

Programming Languages — Ruby

IPA Ruby Standardization WG Draft

February 4, 2010

©Information-technology Promotion Agency, Japan 2010

Contents		Page
1	Scope	1
2	Normative references	1
3	Conformance	1
4	Terms and definitions	2
5	Notational conventions	4
6	Objects	5
6.1	General description	5
6.2	Variables	6
6.3	Methods	6
6.4	Classes, eighclasses, and modules	6
6.4.1	General description	6
6.4.2	Classes	6
6.4.3	Eighclasses	6
6.4.4	Inheritance	7
6.4.5	Modules	7
6.5	Boolean values	7
7	Execution context	7
7.1	Contextual attributes	7
7.2	The initial state	8
8	Lexical structure	9
8.1	General description	9
8.2	Source text	9
8.3	Line terminators	9
8.4	Whitespace	10
8.5	Comments	11
8.6	End-of-program markers	12
8.7	Tokens	12
8.7.1	General description	12
8.7.2	Keywords	12
8.7.3	Identifiers	13
8.7.4	Punctuators	14
8.7.5	Operators	14
8.7.6	Literals	15
8.7.6.1	General description	15
8.7.6.2	Numeric literals	15
8.7.6.3	String literals	18
8.7.6.3.1	General description	18
8.7.6.3.2	Single quoted strings	19
8.7.6.3.3	Double quoted strings	19
8.7.6.3.4	Quoted non-expanded literal strings	22
8.7.6.3.5	Quoted expanded literal strings	24
8.7.6.3.6	Here documents	25
8.7.6.3.7	External command execution	28
8.7.6.4	Array literals	28
8.7.6.5	Regular expression literals	31

8.7.6.6	Symbol literals	33
9	Scope of variables	33
9.1	General description	33
9.2	Local variables	34
9.3	Global variables	34
10	Program structure	35
10.1	Program	35
10.2	Compound statement	35
11	Expressions	36
11.1	General description	36
11.2	Logical expressions	36
11.3	Method invocation expressions	38
11.3.1	General description	38
11.3.2	Method arguments	43
11.3.3	Blocks	46
11.3.4	The <code>super</code> expression	49
11.3.5	The <code>yield</code> expression	51
11.4	Operator expressions	53
11.4.1	General description	53
11.4.2	Assignments	53
11.4.2.1	General description	53
11.4.2.2	Single assignments	54
11.4.2.2.1	General description	54
11.4.2.2.2	Single variable assignments	54
11.4.2.2.3	Scoped constant assignments	56
11.4.2.2.4	Single indexing assignments	57
11.4.2.2.5	Single method assignments	57
11.4.2.3	Abbreviated assignments	58
11.4.2.3.1	General description	58
11.4.2.3.2	Abbreviated variable assignments	59
11.4.2.3.3	Abbreviated indexing assignments	59
11.4.2.3.4	Abbreviated method assignments	60
11.4.2.4	Multiple assignments	61
11.4.2.5	Assignments with <code>rescue</code> modifiers	64
11.4.3	Unary operators	65
11.4.3.1	General description	65
11.4.3.2	The <code>defined?</code> expression	66
11.4.4	Binary operators	67
11.5	Primary expressions	70
11.5.1	General description	70
11.5.2	Control structures	71
11.5.2.1	Conditional expressions	71
11.5.2.1.1	The <code>if</code> expression	71
11.5.2.1.2	The <code>unless</code> expression	72
11.5.2.1.3	The <code>case</code> expression	72
11.5.2.1.4	Conditional operator	74
11.5.2.2	Iteration expressions	74
11.5.2.2.1	General description	74
11.5.2.2.2	The <code>while</code> expression	75
11.5.2.2.3	The <code>until</code> expression	75

11.5.2.2.4	The <code>for</code> expression	76
11.5.2.3	Jump expressions	77
11.5.2.3.1	General description	77
11.5.2.3.2	The <code>return</code> expression	77
11.5.2.3.3	The <code>break</code> expression	79
11.5.2.3.4	The <code>next</code> expression	79
11.5.2.3.5	The <code>redo</code> expression	80
11.5.2.3.6	The <code>retry</code> expression	81
11.5.2.4	Exceptions	81
11.5.2.4.1	The <code>rescue</code> expression	81
11.5.3	Grouping expression	83
11.5.4	Variable references	83
11.5.4.1	General description	83
11.5.4.2	Constants	84
11.5.4.3	Scoped constants	85
11.5.4.4	Global variables	85
11.5.4.5	Class variables	85
11.5.4.6	Instance variables	86
11.5.4.7	Local variables or method invocations	86
11.5.4.7.1	General description	86
11.5.4.7.2	Determination of the type of local variable identifiers	86
11.5.4.7.3	Local variables	87
11.5.4.7.4	Method invocations	87
11.5.4.8	Pseudo variables	87
11.5.4.8.1	General description	87
11.5.4.8.2	The <code>nil</code> expression	88
11.5.4.8.3	The <code>true</code> expression and the <code>false</code> expression	88
11.5.4.8.4	The <code>self</code> expression	88
11.5.5	Object constructors	88
11.5.5.1	Array constructor	88
11.5.5.2	Hash constructor	89
11.5.5.3	Range constructor	90
11.5.6	Literals	91
12	Statements	91
12.1	General description	91
12.2	The expression statement	91
12.3	The <code>if</code> modifier statement	92
12.4	The <code>unless</code> modifier statement	92
12.5	The <code>while</code> modifier statement	92
12.6	The <code>until</code> modifier statement	93
12.7	The <code>rescue</code> modifier statement	93
13	Classes and modules	94
13.1	Modules	94
13.1.1	General description	94
13.1.2	Module definition	95
13.1.3	Module inclusion	96
13.2	Classes	96
13.2.1	General description	96
13.2.2	Class definition	97
13.2.3	Inheritance	98
13.2.4	Instance creation	99

13.3	Methods	99
13.3.1	Method definition	99
13.3.2	Method parameters	100
13.3.3	Method invocation	102
13.3.4	Method lookup	104
13.3.5	Method visibility	105
13.3.5.1	General description	105
13.3.5.2	Public methods	105
13.3.5.3	Private methods	105
13.3.5.4	Protected methods	105
13.3.5.5	Visibility change	106
13.3.6	The <code>alias</code> statement	106
13.3.7	The <code>undef</code> statement	107
13.4	Eigenclasses	108
13.4.1	General description	108
13.4.2	Eigenclass definition	109
13.4.3	Singleton method definition	110
14	Exceptions	111
14.1	General description	111
14.2	Cause of exceptions	111
14.3	Exception handling	111
15	Built-in classes and modules	112
15.1	General description	112
15.2	Built-in classes	113
15.2.1	Object	113
15.2.1.1	General description	113
15.2.1.2	Direct superclass	113
15.2.1.3	Included modules	114
15.2.1.4	Constants	114
15.2.1.5	Instance methods	114
15.2.1.5.1	Object#initialize	114
15.2.2	Module	114
15.2.2.1	General description	114
15.2.2.2	Direct superclass	114
15.2.2.3	Singleton methods	115
15.2.2.3.1	Module.constants	115
15.2.2.3.2	Module.nesting	115
15.2.2.4	Instance methods	115
15.2.2.4.1	Module#<	115
15.2.2.4.2	Module#<=	116
15.2.2.4.3	Module#<=>	116
15.2.2.4.4	Module#==	117
15.2.2.4.5	Module#===	117
15.2.2.4.6	Module#>	117
15.2.2.4.7	Module#>=	117
15.2.2.4.8	Module#alias_method	118
15.2.2.4.9	Module#ancestors	118
15.2.2.4.10	Module#append_features	118
15.2.2.4.11	Module#attr	119
15.2.2.4.12	Module#attr_accessor	119
15.2.2.4.13	Module#attr_reader	120

15.2.2.4.14	Module#attr_writer	120
15.2.2.4.15	Module#class_eval	120
15.2.2.4.16	Module#class_variable_defined?	122
15.2.2.4.17	Module#class_variable_get	122
15.2.2.4.18	Module#class_variable_set	122
15.2.2.4.19	Module#class_variables	123
15.2.2.4.20	Module#const_defined?	123
15.2.2.4.21	Module#const_get	123
15.2.2.4.22	Module#const_missing	124
15.2.2.4.23	Module#const_set	124
15.2.2.4.24	Module#constants	125
15.2.2.4.25	Module#extend_object	125
15.2.2.4.26	Module#extended	125
15.2.2.4.27	Module#include	125
15.2.2.4.28	Module#include?	126
15.2.2.4.29	Module#included	126
15.2.2.4.30	Module#included_modules	126
15.2.2.4.31	Module#initialize	127
15.2.2.4.32	Module#initialize_copy	127
15.2.2.4.33	Module#instance_methods	128
15.2.2.4.34	Module#method_defined?	129
15.2.2.4.35	Module#module_eval	129
15.2.2.4.36	Module#private	129
15.2.2.4.37	Module#protected	129
15.2.2.4.38	Module#public	130
15.2.2.4.39	Module#remove_class_variable	131
15.2.2.4.40	Module#remove_const	131
15.2.2.4.41	Module#remove_method	132
15.2.2.4.42	Module#undef_method	132
15.2.3	Class	132
15.2.3.1	General description	132
15.2.3.2	Direct superclass	133
15.2.3.3	Instance methods	133
15.2.3.3.1	Class#initialize	133
15.2.3.3.2	Class#initialize_copy	133
15.2.3.3.3	Class#new	134
15.2.3.3.4	Class#superclass	134
15.2.4	NilClass	134
15.2.4.1	General description	134
15.2.4.2	Direct superclass	134
15.2.4.3	Instance methods	134
15.2.4.3.1	NilClass#&	134
15.2.4.3.2	NilClass#^	135
15.2.4.3.3	NilClass#nil?	135
15.2.4.3.4	NilClass#	135
15.2.5	TrueClass	135
15.2.5.1	General description	135
15.2.5.2	Direct superclass	136
15.2.5.3	Instance methods	136
15.2.5.3.1	TrueClass#&	136
15.2.5.3.2	TrueClass#^	136
15.2.5.3.3	TrueClass#to_s	136

15.2.5.3.4	TrueClass#	136
15.2.6	FalseClass	137
15.2.6.1	General description	137
15.2.6.2	Direct superclass	137
15.2.6.3	Instance methods	137
15.2.6.3.1	FalseClass#&	137
15.2.6.3.2	FalseClass#^	137
15.2.6.3.3	FalseClass#to_s	137
15.2.6.3.4	FalseClass#	138
15.2.7	Numeric	138
15.2.7.1	General description	138
15.2.7.2	Direct superclass	138
15.2.7.3	Included modules	138
15.2.7.4	Instance methods	138
15.2.7.4.1	Numeric#+@	138
15.2.7.4.2	Numeric#-@	139
15.2.7.4.3	Numeric#abs	139
15.2.7.4.4	Numeric#coerce	139
15.2.8	Integer	140
15.2.8.1	General description	140
15.2.8.2	Direct superclass	140
15.2.8.3	Instance methods	141
15.2.8.3.1	Integer#+	141
15.2.8.3.2	Integer#-	141
15.2.8.3.3	Integer#*	142
15.2.8.3.4	Integer#/	142
15.2.8.3.5	Integer#%	143
15.2.8.3.6	Integer#<=>	144
15.2.8.3.7	Integer#==	144
15.2.8.3.8	Integer#~	145
15.2.8.3.9	Integer#&	145
15.2.8.3.10	Integer#	145
15.2.8.3.11	Integer#^	145
15.2.8.3.12	Integer#<<	146
15.2.8.3.13	Integer#>>	146
15.2.8.3.14	Integer#ceil	146
15.2.8.3.15	Integer#downto	147
15.2.8.3.16	Integer#eql?	147
15.2.8.3.17	Integer#floor	147
15.2.8.3.18	Integer#hash	148
15.2.8.3.19	Integer#next	148
15.2.8.3.20	Integer#round	148
15.2.8.3.21	Integer#succ	148
15.2.8.3.22	Integer#times	148
15.2.8.3.23	Integer#to_f	149
15.2.8.3.24	Integer#to_i	149
15.2.8.3.25	Integer#truncate	149
15.2.8.3.26	Integer#upto	149
15.2.9	Float	150
15.2.9.1	General description	150
15.2.9.2	Direct superclass	150
15.2.9.3	Instance methods	150

15.2.9.3.1	Float#+	150
15.2.9.3.2	Float#-	151
15.2.9.3.3	Float#*	152
15.2.9.3.4	Float#/	152
15.2.9.3.5	Float#%	153
15.2.9.3.6	Float#<=>	154
15.2.9.3.7	Float#==	154
15.2.9.3.8	Float#ceil	155
15.2.9.3.9	Float#finite?	155
15.2.9.3.10	Float#floor	155
15.2.9.3.11	Float#infinite?	156
15.2.9.3.12	Float#round	156
15.2.9.3.13	Float#to_f	156
15.2.9.3.14	Float#to_i	156
15.2.9.3.15	Float#truncate	157
15.2.10	String	157
15.2.10.1	General description	157
15.2.10.2	Direct superclass	157
15.2.10.3	Included modules	157
15.2.10.4	Upper-case and lower-case characters	158
15.2.10.5	Instance methods	158
15.2.10.5.1	String#*	158
15.2.10.5.2	String#+	159
15.2.10.5.3	String#<=>	159
15.2.10.5.4	String#==	160
15.2.10.5.5	String#=~	160
15.2.10.5.6	String#[]	160
15.2.10.5.7	String#capitalize	162
15.2.10.5.8	String#capitalize!	162
15.2.10.5.9	String#chomp	162
15.2.10.5.10	String#chomp!	163
15.2.10.5.11	String#chop	163
15.2.10.5.12	String#chop!	163
15.2.10.5.13	String#downcase	164
15.2.10.5.14	String#downcase!	164
15.2.10.5.15	String#each_line	164
15.2.10.5.16	String#empty?	165
15.2.10.5.17	String#eql?	165
15.2.10.5.18	String#gsub	165
15.2.10.5.19	String#gsub!	167
15.2.10.5.20	String#hash	167
15.2.10.5.21	String#include?	167
15.2.10.5.22	String#index	168
15.2.10.5.23	String#initialize	168
15.2.10.5.24	String#initialize_copy	169
15.2.10.5.25	String#intern	169
15.2.10.5.26	String#length	169
15.2.10.5.27	String#match	169
15.2.10.5.28	String#replace	170
15.2.10.5.29	String#reverse	170
15.2.10.5.30	String#reverse!	170
15.2.10.5.31	String#rindex	170

15.2.10.5.32	String#scan	171
15.2.10.5.33	String#size	172
15.2.10.5.34	String#slice	172
15.2.10.5.35	String#split	172
15.2.10.5.36	String#sub	173
15.2.10.5.37	String#sub!	174
15.2.10.5.38	String#upcase	174
15.2.10.5.39	String#upcase!	175
15.2.10.5.40	String#to_i	175
15.2.10.5.41	String#to_f	176
15.2.10.5.42	String#to_s	176
15.2.10.5.43	String#to_sym	177
15.2.11	Symbol	177
15.2.11.1	General description	177
15.2.11.2	Direct superclass	177
15.2.11.3	Instance methods	177
15.2.11.3.1	Symbol#===	177
15.2.11.3.2	Symbol#id2name	177
15.2.11.3.3	Symbol#to_s	177
15.2.11.3.4	Symbol#to_sym	178
15.2.12	Array	178
15.2.12.1	General description	178
15.2.12.2	Direct superclass	179
15.2.12.3	Included modules	179
15.2.12.4	Singleton methods	179
15.2.12.4.1	Array.[]	179
15.2.12.5	Instance methods	179
15.2.12.5.1	Array#*	179
15.2.12.5.2	Array#+	179
15.2.12.5.3	Array#<<	180
15.2.12.5.4	Array#[]	180
15.2.12.5.5	Array#[]=	181
15.2.12.5.6	Array#clear	182
15.2.12.5.7	Array#collect!	182
15.2.12.5.8	Array#concat	182
15.2.12.5.9	Array#delete_at	182
15.2.12.5.10	Array#each	183
15.2.12.5.11	Array#each_index	183
15.2.12.5.12	Array#empty?	183
15.2.12.5.13	Array#first	184
15.2.12.5.14	Array#index	184
15.2.12.5.15	Array#initialize	185
15.2.12.5.16	Array#initialize_copy	185
15.2.12.5.17	Array#join	186
15.2.12.5.18	Array#last	186
15.2.12.5.19	Array#length	187
15.2.12.5.20	Array#map!	187
15.2.12.5.21	Array#pop	187
15.2.12.5.22	Array#push	188
15.2.12.5.23	Array#replace	188
15.2.12.5.24	Array#reverse	188
15.2.12.5.25	Array#reverse!	188

15.2.12.5.26	Array#rindex	188
15.2.12.5.27	Array#shift	189
15.2.12.5.28	Array#size	189
15.2.12.5.29	Array#slice	189
15.2.12.5.30	Array#unshift	189
15.2.13	Hash	190
15.2.13.1	General description	190
15.2.13.2	Direct superclass	191
15.2.13.3	Included modules	191
15.2.13.4	Instance methods	191
15.2.13.4.1	Hash#==	191
15.2.13.4.2	Hash#[]	191
15.2.13.4.3	Hash#[]=	192
15.2.13.4.4	Hash#clear	192
15.2.13.4.5	Hash#default	192
15.2.13.4.6	Hash#default=	193
15.2.13.4.7	Hash#default_proc	193
15.2.13.4.8	Hash#delete	194
15.2.13.4.9	Hash#each	194
15.2.13.4.10	Hash#each_key	194
15.2.13.4.11	Hash#each_value	195
15.2.13.4.12	Hash#empty?	195
15.2.13.4.13	Hash#has_key?	195
15.2.13.4.14	Hash#has_value?	195
15.2.13.4.15	Hash#include?	196
15.2.13.4.16	Hash#initialize	196
15.2.13.4.17	Hash#initialize_copy	197
15.2.13.4.18	Hash#key?	197
15.2.13.4.19	Hash#keys	197
15.2.13.4.20	Hash#length	197
15.2.13.4.21	Hash#member?	198
15.2.13.4.22	Hash#merge	198
15.2.13.4.23	Hash#replace	198
15.2.13.4.24	Hash#shift	199
15.2.13.4.25	Hash#size	199
15.2.13.4.26	Hash#store	199
15.2.13.4.27	Hash#value?	199
15.2.13.4.28	Hash#values	200
15.2.14	Range	200
15.2.14.1	General description	200
15.2.14.2	Direct superclass	200
15.2.14.3	Included modules	200
15.2.14.4	Instance methods	200
15.2.14.4.1	Range#==	200
15.2.14.4.2	Range#===	201
15.2.14.4.3	Range#begin	201
15.2.14.4.4	Range#each	202
15.2.14.4.5	Range#end	202
15.2.14.4.6	Range#exclude_end?	202
15.2.14.4.7	Range#first	203
15.2.14.4.8	Range#include?	203
15.2.14.4.9	Range#initialize	203

15.2.14.4.10	Range#last	203
15.2.14.4.11	Range#member?	204
15.2.15	Regexp	204
15.2.15.1	General description	204
15.2.15.2	Direct superclass	204
15.2.15.3	Constants	204
15.2.15.4	Patterns	205
15.2.15.5	Matching process	209
15.2.15.6	Singleton methods	210
15.2.15.6.1	Regexp.compile	210
15.2.15.6.2	Regexp.escape	210
15.2.15.6.3	Regexp.last_match	210
15.2.15.6.4	Regexp.quote	212
15.2.15.7	Instance methods	212
15.2.15.7.1	Regexp#initialize	212
15.2.15.7.2	Regexp#initialize_copy	212
15.2.15.7.3	Regexp#===	213
15.2.15.7.4	Regexp#====	213
15.2.15.7.5	Regexp#=~	214
15.2.15.7.6	Regexp#casefold?	214
15.2.15.7.7	Regexp#match	214
15.2.15.7.8	Regexp#source	215
15.2.16	MatchData	215
15.2.16.1	General description	215
15.2.16.2	Direct superclass	215
15.2.16.3	Instance methods	215
15.2.16.3.1	MatchData#[]	215
15.2.16.3.2	MatchData#begin	216
15.2.16.3.3	MatchData#captures	216
15.2.16.3.4	MatchData#end	216
15.2.16.3.5	MatchData#initialize_copy	217
15.2.16.3.6	MatchData#length	217
15.2.16.3.7	MatchData#offset	217
15.2.16.3.8	MatchData#post_match	218
15.2.16.3.9	MatchData#pre_match	218
15.2.16.3.10	MatchData#size	218
15.2.16.3.11	MatchData#string	218
15.2.16.3.12	MatchData#to_a	218
15.2.16.3.13	MatchData#to_s	219
15.2.17	Proc	219
15.2.17.1	General description	219
15.2.17.2	Direct superclass	219
15.2.17.3	Singleton methods	219
15.2.17.3.1	Proc.new	219
15.2.17.4	Instance methods	220
15.2.17.4.1	Proc#[]	220
15.2.17.4.2	Proc#arity	220
15.2.17.4.3	Proc#call	221
15.2.17.4.4	Proc#clone	221
15.2.17.4.5	Proc#dup	222
15.2.18	Struct	222
15.2.18.1	General description	222

15.2.18.2	Direct superclass	222
15.2.18.3	Singleton methods	222
15.2.18.3.1	Struct.new	222
15.2.18.4	Instance methods	224
15.2.18.4.1	Struct#==	224
15.2.18.4.2	Struct#[]	225
15.2.18.4.3	Struct#[]=	225
15.2.18.4.4	Struct#each	226
15.2.18.4.5	Struct#each_pair	226
15.2.18.4.6	Struct#members	227
15.2.18.4.7	Struct#select	227
15.2.18.4.8	Struct#initialize	227
15.2.18.4.9	Struct#initialize_copy	228
15.2.19	Time	228
15.2.19.1	General description	228
15.2.19.2	Direct superclass	229
15.2.19.3	Time computation	229
15.2.19.3.1	Day	229
15.2.19.3.2	Year	229
15.2.19.3.3	Month	230
15.2.19.3.4	Days of month	231
15.2.19.3.5	Hours, Minutes, and Seconds	231
15.2.19.4	Time zone and Local time	231
15.2.19.5	Daylight saving time	232
15.2.19.6	Singleton methods	232
15.2.19.6.1	Time.at	232
15.2.19.6.2	Time.gm	233
15.2.19.6.3	Time.local	234
15.2.19.6.4	Time.mktime	235
15.2.19.6.5	Time.now	235
15.2.19.6.6	Time.utc	235
15.2.19.7	Instance methods	235
15.2.19.7.1	Time#+	235
15.2.19.7.2	Time#-	236
15.2.19.7.3	Time#<=>	236
15.2.19.7.4	Time#asctime	236
15.2.19.7.5	Time#ctime	237
15.2.19.7.6	Time#day	237
15.2.19.7.7	Time#dst?	238
15.2.19.7.8	Time#getgm	238
15.2.19.7.9	Time#getlocal	238
15.2.19.7.10	Time#getutc	238
15.2.19.7.11	Time#gmt?	239
15.2.19.7.12	Time#gmt_offset	239
15.2.19.7.13	Time#gmtime	239
15.2.19.7.14	Time#gmtoff	239
15.2.19.7.15	Time#hour	239
15.2.19.7.16	Time#localtime	240
15.2.19.7.17	Time#mday	240
15.2.19.7.18	Time#min	240
15.2.19.7.19	Time#mon	241
15.2.19.7.20	Time#month	241

15.2.19.7.21	Time#sec	241
15.2.19.7.22	Time#to_f	241
15.2.19.7.23	Time#to_i	242
15.2.19.7.24	Time#usec	242
15.2.19.7.25	Time#utc	242
15.2.19.7.26	Time#utc?	243
15.2.19.7.27	Time#utc_offset	243
15.2.19.7.28	Time#wday	243
15.2.19.7.29	Time#yday	243
15.2.19.7.30	Time#year	244
15.2.19.7.31	Time#zone	244
15.2.19.7.32	Time#initialize	244
15.2.19.7.33	Time#initialize_copy	245
15.2.20	IO	245
15.2.20.1	General description	245
15.2.20.2	Direct superclass	246
15.2.20.3	Included modules	246
15.2.20.4	Singleton methods	246
15.2.20.4.1	IO.open	246
15.2.20.5	Instance methods	246
15.2.20.5.1	IO#close	246
15.2.20.5.2	IO#closed?	247
15.2.20.5.3	IO#each	247
15.2.20.5.4	IO#each_byte	247
15.2.20.5.5	IO#each_line	248
15.2.20.5.6	IO#eof?	248
15.2.20.5.7	IO#flush	248
15.2.20.5.8	IO#getc	249
15.2.20.5.9	IO#gets	249
15.2.20.5.10	IO#initialize_copy	249
15.2.20.5.11	IO#print	249
15.2.20.5.12	IO#putc	250
15.2.20.5.13	IO#puts	250
15.2.20.5.14	IO#read	251
15.2.20.5.15	IO#readchar	252
15.2.20.5.16	IO#readline	252
15.2.20.5.17	IO#readlines	252
15.2.20.5.18	IO#sync	253
15.2.20.5.19	IO#sync=	253
15.2.20.5.20	IO#write	253
15.2.21	File	254
15.2.21.1	General description	254
15.2.21.2	Direct superclass	254
15.2.21.3	Singleton methods	254
15.2.21.3.1	File.exist?	254
15.2.21.4	Instance methods	255
15.2.21.4.1	File#initialize	255
15.2.21.4.2	File#path	255
15.2.22	Exception	256
15.2.22.1	General description	256
15.2.22.2	Direct superclass	256
15.2.22.3	Built-in exception classes	256

15.2.22.4	Singleton methods	256
15.2.22.4.1	Exception.exception	256
15.2.22.5	Instance methods	256
15.2.22.5.1	Exception#exception	256
15.2.22.5.2	Exception#message	257
15.2.22.5.3	Exception#to_s	258
15.2.22.5.4	Exception#initialize	258
15.2.23	StandardError	258
15.2.23.1	General description	258
15.2.23.2	Direct superclass	258
15.2.24	ArgumentError	258
15.2.24.1	General description	258
15.2.24.2	Direct superclass	258
15.2.25	LocalJumpError	259
15.2.25.1	Direct superclass	259
15.2.25.2	Instance methods	259
15.2.25.2.1	LocalJumpError#exit_value	259
15.2.25.2.2	LocalJumpError#reason	259
15.2.26	RangeError	259
15.2.26.1	General description	259
15.2.26.2	Direct superclass	259
15.2.27	RegexpError	259
15.2.27.1	General description	259
15.2.27.2	Direct superclass	259
15.2.28	RuntimeError	260
15.2.28.1	General description	260
15.2.28.2	Direct superclass	260
15.2.29	TypeError	260
15.2.29.1	General description	260
15.2.29.2	Direct superclass	260
15.2.30	ZeroDivisionError	260
15.2.30.1	General description	260
15.2.30.2	Direct superclass	260
15.2.31	NameError	260
15.2.31.1	Direct superclass	260
15.2.31.2	Instance methods	260
15.2.31.2.1	NameError#name	260
15.2.31.2.2	NameError#initialize	261
15.2.32	NoMethodError	261
15.2.32.1	Direct superclass	261
15.2.32.2	Instance methods	261
15.2.32.2.1	NoMethodError#args	261
15.2.32.2.2	NoMethodError#initialize	262
15.2.33	IndexError	262
15.2.33.1	General description	262
15.2.33.2	Direct superclass	262
15.2.34	StopIteration	262
15.2.34.1	General description	262
15.2.34.2	Direct superclass	262
15.2.35	IOError	262
15.2.35.1	General description	262
15.2.35.2	Direct superclass	262

15.2.36	EOFError	262
15.2.36.1	General description	262
15.2.36.2	Direct superclass	263
15.2.37	SystemCallError	263
15.2.37.1	General description	263
15.2.37.2	Direct superclass	263
15.2.38	ScriptError	263
15.2.38.1	General description	263
15.2.38.2	Direct superclass	263
15.2.39	SyntaxError	263
15.2.39.1	General description	263
15.2.39.2	Direct superclass	263
15.2.40	LoadError	263
15.2.40.1	General description	263
15.2.40.2	Direct superclass	263
15.3	Built-in modules	264
15.3.1	Kernel	264
15.3.1.1	General description	264
15.3.1.2	Singleton methods	264
15.3.1.2.1	Kernel.`	264
15.3.1.2.2	Kernel.block_given?	264
15.3.1.2.3	Kernel.eval	264
15.3.1.2.4	Kernel.global_variables	265
15.3.1.2.5	Kernel.iterator?	265
15.3.1.2.6	Kernel.lambda	265
15.3.1.2.7	Kernel.local_variables	266
15.3.1.2.8	Kernel.loop	267
15.3.1.2.9	Kernel.method_missing	267
15.3.1.2.10	Kernel.p	267
15.3.1.2.11	Kernel.print	268
15.3.1.2.12	Kernel.puts	268
15.3.1.2.13	Kernel.raise	268
15.3.1.2.14	Kernel.require	269
15.3.1.3	Instance methods	270
15.3.1.3.1	Kernel#==	270
15.3.1.3.2	Kernel#===	270
15.3.1.3.3	Kernel#_id_	270
15.3.1.3.4	Kernel#_send_	271
15.3.1.3.5	Kernel#`	271
15.3.1.3.6	Kernel#block_given?	271
15.3.1.3.7	Kernel#class	271
15.3.1.3.8	Kernel#clone	271
15.3.1.3.9	Kernel#dup	272
15.3.1.3.10	Kernel#eql?	272
15.3.1.3.11	Kernel#equal?	273
15.3.1.3.12	Kernel#eval	273
15.3.1.3.13	Kernel#extend	273
15.3.1.3.14	Kernel#global_variables	274
15.3.1.3.15	Kernel#hash	274
15.3.1.3.16	Kernel#initialize_copy	274
15.3.1.3.17	Kernel#inspect	274
15.3.1.3.18	Kernel#instance_eval	275

15.3.1.3.19	Kernel#instance_of?	275
15.3.1.3.20	Kernel#instance_variable_defined?	275
15.3.1.3.21	Kernel#instance_variable_get	276
15.3.1.3.22	Kernel#instance_variable_set	276
15.3.1.3.23	Kernel#instance_variables	277
15.3.1.3.24	Kernel#is_a?	277
15.3.1.3.25	Kernel#iterator?	277
15.3.1.3.26	Kernel#kind_of?	277
15.3.1.3.27	Kernel#lambda	278
15.3.1.3.28	Kernel#local_variables	278
15.3.1.3.29	Kernel#loop	278
15.3.1.3.30	Kernel#method_missing	278
15.3.1.3.31	Kernel#methods	278
15.3.1.3.32	Kernel#nil?	279
15.3.1.3.33	Kernel#object_id	279
15.3.1.3.34	Kernel#p	279
15.3.1.3.35	Kernel#print	279
15.3.1.3.36	Kernel#private_methods	280
15.3.1.3.37	Kernel#protected_methods	281
15.3.1.3.38	Kernel#public_methods	281
15.3.1.3.39	Kernel#puts	281
15.3.1.3.40	Kernel#raise	281
15.3.1.3.41	Kernel#remove_instance_variable	281
15.3.1.3.42	Kernel#require	282
15.3.1.3.43	Kernel#respond_to?	282
15.3.1.3.44	Kernel#send	283
15.3.1.3.45	Kernel#singleton_methods	283
15.3.1.3.46	Kernel#to_s	283
15.3.2	Enumerable	284
15.3.2.1	General description	284
15.3.2.2	Instance methods	284
15.3.2.2.1	Enumerable#all?	284
15.3.2.2.2	Enumerable#any?	284
15.3.2.2.3	Enumerable#collect	285
15.3.2.2.4	Enumerable#detect	285
15.3.2.2.5	Enumerable#each_with_index	286
15.3.2.2.6	Enumerable#entries	286
15.3.2.2.7	Enumerable#find	286
15.3.2.2.8	Enumerable#find_all	286
15.3.2.2.9	Enumerable#grep	287
15.3.2.2.10	Enumerable#include?	287
15.3.2.2.11	Enumerable#inject	288
15.3.2.2.12	Enumerable#map	288
15.3.2.2.13	Enumerable#max	288
15.3.2.2.14	Enumerable#min	289
15.3.2.2.15	Enumerable#member?	290
15.3.2.2.16	Enumerable#partition	290
15.3.2.2.17	Enumerable#reject	291
15.3.2.2.18	Enumerable#select	291
15.3.2.2.19	Enumerable#sort	291
15.3.2.2.20	Enumerable#to_a	292
15.3.3	Comparable	292

15.3.3.1	General description	292
15.3.3.2	Instance methods	292
15.3.3.2.1	Comparable#<	292
15.3.3.2.2	Comparable#<=	293
15.3.3.2.3	Comparable#==	293
15.3.3.2.4	Comparable#>	293
15.3.3.2.5	Comparable#>=	294
15.3.3.2.6	Comparable#between?	294
Annex A (informative)	Grammar Summary	295

Information technology — Programming Languages — Ruby

1 Scope

This document specifies the syntax and semantics of the computer programming language Ruby by specifying requirements for a conforming processor and for a conforming program.

This document does not specify:

- the size or complexity of a program text that exceeds the capacity of any specific data processing system or the capacity of a particular processor;
- the minimal requirements of a data processing system that is capable of supporting a conforming processor;
- the method for activating the execution of programs on a data processing system;
- the method for reporting syntactic and runtime errors.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 646:1991 *Information technology – ISO 7-bit coded character set for information interchange.*

IEC 60559:1989 *Binary floating-point arithmetic for microprocessor systems.*

3 Conformance

A conforming Ruby program shall:

- use only those features of the language specified in this document;
- not rely on unspecified features;

A conforming Ruby processor shall:

- 1 • accept any conforming programs and behave as specified in this document;
- 2 • report any unhandled exceptions raised during execution of the conforming program;

3 A conforming Ruby processor may:

- 4 • use an internal model for the Ruby language other than the one described in this document,
5 if it does not change the behavior of a conforming program;
- 6 • support syntax not described in this document, and accept any programs which use features
7 not specified in this document;

8 **4 Terms and definitions**

9 For the purposes of this document, the following terms and definitions apply. Other terms are
10 defined where they appear in ***bold slant face*** or on the left side of a syntax rule.

11 **4.1**

12 **block**

13 procedure which is passed to a method invocation

14 **4.2**

15 **class**

16 object which defines the behavior of a set of other objects called its instances

17 NOTE The behavior is a set of methods which can be invoked on an instance.

18 **4.3**

19 **class variable**

20 variable whose value is shared by all the instances of a class

21 **4.4**

22 **constant**

23 variable which is defined in a class or a module and is accessible outside the class or module

24 NOTE The value of a constant is ordinarily expected to remain constant during the execution of a
25 program, but Ruby does not force it. In some implementations, an assignment to a constant which
26 already exists causes a warning, but this document does not specify it.

27 **4.5**

28 **eigenclass**

29 special class which, when associated to an object, can modify the behavior of that object

30 **4.6**

31 **exception**

32 object which represents an unexpected event

33 **4.7**

34 **global variable**

35 variable which is accessible everywhere in a conforming program

36 **4.8**

1 **implementation-defined**
2 possibly differing between implementations, but defined for every implementation

3 **4.9**
4 **unspecified**
5 possibly differing between implementations, and not necessarily defined for any particular im-
6 plementation

7 **4.10**
8 **instance method**
9 method which can be invoked on all the instances of a class

10 **4.11**
11 **instance variable**
12 variable which belongs to a single object

13 **4.12**
14 **local variable**
15 variable which is accessible only in a certain scope introduced by a program construct such as
16 a method definition, a block, a class definition, a module definition, an eigenclass definition, or
17 the toplevel of a program

18 **4.13**
19 **method**
20 procedure which, when invoked on an object, performs a set of computations on the object

21 **4.14**
22 **method visibility**
23 attribute of a method which determines the conditions under which a method invocation is
24 allowed

25 **4.15**
26 **module**
27 object which provides features to be included into a class or another module

28 **4.16**
29 **object**
30 computational entity which has states and a behavior

31 **4.17**
32 **singleton method**
33 instance method of the eigenclass of an object

34 **4.18**
35 **variable**
36 computational entity which stores a reference to an object

1 5 Notational conventions

2 5.1 Syntax

3 The syntax of the language is presented as a series of productions. Each production consists
4 of the name of the nonterminal symbol being defined followed by “::”, followed by one or more
5 alternatives separated by “[”.

6 Terminal symbols are shown in **typewriter face**, and represent sequences of characters as they
7 appear in a program text. Non-terminal symbols are shown in *italic face*.

8 Each alternative in a production consists of a sequence of terminal and/or nonterminal symbols
9 separated by whitespace.

10 If the same nonterminal symbol occurs on the right side of a production more than once, each
11 occurrence is subscripted with a number to distinguish it from the other occurrences of the same
12 name.

13 An optional symbol is denoted by postfixing the symbol with “?”.

14 A sequence of zero or more repetitions of a symbol is denoted by postfixing the symbol with “*”.

15 A sequence of one or more repetitions of a symbol is denoted by postfixing the symbol with “+”.

16 Parentheses are used to treat a sequence of symbols as a single symbol.

17 A symbol followed by the phrase **but not** and another symbol represents all sequences of charac-
18 ters represented by the first symbol except for sequences of characters represented by the second
19 symbol.

20 EXAMPLE 1 The following example means that *non-escaped-character* is any member of *source-character*
21 except *escape-character*:

```
22 non-escaped-character ::  
23 source-character but not escape-character
```

24 Text enclosed by “[” and “]” is used to describe a sequence of characters which is difficult to
25 express by production rules, or to describe a location in a program text.

26 EXAMPLE 2 The following example means that *source-character* is any character specified in ISO/IEC
27 646:1991 IRV:

```
28 source-character ::  
29 [ any character in ISO/IEC 646:1991 IRV ]
```

30 In particular, the notation “[lookahead \notin *set*]” indicates that the token immediately following
31 the notation shall not begin with a sequence of characters represented by one of the members
32 of *set*. The *set* is represented as a list of one or more terminal symbols separated by commas,
33 and the list is enclosed by “{” and “}”.

1 EXAMPLE 3 The following example means that the *argument* following the *method-identifier* shall not
2 begin with “{”:

3 *command* ::
4 *method-identifier* [lookahead \notin { { }] *argument*
5

6 In this document, use of the words **of** and **in**, when expressing a relationship between nonter-
7 minal symbols, has the following meanings:

8 **X of Y:** refers to the *X* occurring directly in a production defining *Y*.

9 **X in Y:** refers to any *X* occurring in a sequence which is derived directly or indirectly
10 from *Y*.

11 5.2 Conceptual names

12 A **conceptual name** is a common name given to a set of semantically related nonterminal
13 symbols in the grammar in order to refer to this set in a semantic description. A conceptual
14 name is defined by a **conceptual name definition**. A conceptual name definition consists
15 of the conceptual name to be defined followed by “::=”, followed by one or more nonterminal
16 symbols or conceptual names, separated by “|”.

17 EXAMPLE The following example defines the conceptual name *assignment*, which can be used to refer
18 either *assignment-expression* or *assignment-statement*.

19 *assignment* ::=
20 *assignment-expression*
21 | *assignment-statement*

22 5.3 Attributes of execution contexts

23 The names of the attributes of execution contexts are enclosed in double square brackets “[”
24 and “]” (see §7.1).

25 EXAMPLE `[[self]]` is one of the attributes of execution contexts.

26 6 Objects

27 6.1 General description

28 Ruby is a pure object-oriented language. It is pure in the sense that every value manipulated
29 in a conforming program is an object.

30 An object is a computational entity which has states and a behavior.

1 6.2 Variables

2 A variable is a computational entity which stores a reference to an object. A variable has a
3 name. A variable is said to be **bound** to an object if the variable stores a reference to the
4 object. This association of a variable with an object is called a **variable binding**. When a
5 variable with name N is bound to an object O , N is called the name of the binding, and O is
6 called the value of the binding.

7 An object has a set of variable bindings. A variable whose binding is in this set is an instance
8 variable of that object. This set of bindings of instance variables represents a state of that object
9 and is encapsulated in that object.

10 6.3 Methods

11 A method is a procedure which, when invoked, performs a set of computations. The behavior
12 of an object is defined by a set of methods which can be invoked on that object. A method
13 has one or more (when aliased) names associated with it. An association between a name and
14 a method is called a **method binding**. When a name N is bound to a method M , N is called
15 the name of the binding, and M is called the value of the binding. A name to which a method
16 is bound is called the **method name**. A method can be invoked on an object by specifying one
17 of its names. The object on which the method is invoked is called the **receiver** of the method
18 invocation.

19 Methods are described further in §13.3.

20 6.4 Classes, eigenclasses, and modules

21 6.4.1 General description

22 There are three constructs which define behaviors of objects: classes, eigenclasses, and modules.
23 A class defines methods shared by objects of the same class. An eigenclass is a special class
24 which, when associated to an object, can modify the behavior of that object. A module defines,
25 and provides methods to be included into a class or another module. All these three constructs
26 are themselves objects, which are dynamically created and modified at run-time.

27 6.4.2 Classes

28 A class creates objects, and the created objects are called **direct instances** of the class (see
29 §13.2.4). A class defines a set of methods which can be invoked on all the instances of the class.
30 These methods are instance methods of the class. A class is itself an object, and created by an
31 evaluation of a program construct such as a class definition (see §13.2.2). A class has two sets
32 of variable bindings besides a set of bindings of instance variables. The one is a set of bindings
33 of constants. The other is a set of bindings of class variables, which represents the state shared
34 by all the instances of the class.

35 Classes are described further in §13.2.

36 6.4.3 Eigenclasses

37 Every object, including classes, can be associated with at most one special class to the object.
38 This special class is called the eigenclass of the object. The eigenclass defines methods which
39 can be invoked on that object. Those methods are singleton methods of the object. If the object

1 is not a class, the singleton methods of the object can be invoked on only that object. If the
2 object is a class, singleton methods of the class are similar to so-called class methods because
3 they can be invoked on that class. An eigenclass is created, and associated with an object by
4 an eigenclass definition (see §13.4.2) or a singleton method definition (see §13.4.3).

5 Eigenclasses are described further in §13.4.

6 **6.4.4 Inheritance**

7 A class has at most one single class as its *direct superclass*. If a class *A* has a class *B* as its
8 direct superclass, *A* is called a *direct subclass* of *B*. Classes form a rooted tree called a *class*
9 *inheritance tree*, where the parent of a class is its direct superclass, and the children of a class
10 are all its direct subclasses. There is only one class which does not have a superclass. It is the
11 root of the tree. All the ancestors of a class in the tree are called *superclasses* of the class.
12 All the descendants of a class in the tree are called *subclasses* of the class. A class inherits
13 constants, class variables, singleton methods, and instance methods from its superclasses, if any
14 (see §13.2.3). If an object *C* is a direct instance of a class *D*, *C* is called an instance of *D* and
15 all its superclasses.

16 **6.4.5 Modules**

17 Ruby does not support multiple inheritance; that is, a class can have only one direct superclass.
18 However, Ruby supports module inclusion, which is a mechanism to append features into a class
19 from multiple sources.

20 A module is an object which has the same structure as a class except that it cannot create an
21 instance of itself and cannot be inherited. As with classes, a module has a set of class variables
22 and instance methods. Instance methods and class variables defined in a module can be used
23 by other classes, modules and eigenclasses by including the module into them. While a class
24 can have only one direct superclass, a class or a module can include multiple modules. Instance
25 methods defined in a module can be invoked on an instance of a class which includes the module.
26 A module is created by a module definition (see §13.1.2).

27 Modules are described further in §13.1.

28 **6.5 Boolean values**

29 An object is classified into either a *trueish value* or a *falseish value*.

30 Only **false** and **nil** are falseish values. **false** is the only instance of the class `FalseClass` (see
31 §15.2.6), to which a *false-expression* evaluates (see §11.5.4.8.3). **nil** is the only instance of the
32 class `NilClass` (see §15.2.4), to which a *nil-expression* evaluates (see §11.5.4.8.2).

33 Objects other than **false** and **nil** are classified into trueish values. **true** is the only instance of
34 the class `TrueClass` (see §15.2.5), to which a *true-expression* evaluates (see §11.5.4.8.3).

35 **7 Execution context**

36 **7.1 Contextual attributes**

37 An *execution context* is a set of attributes which affects an evaluation of a program.

1 An execution context is not a part of the language. It is defined in this document only for the
2 description of the semantics of a program. A conforming processor shall evaluate a program as
3 if the processor acted upon an execution context in the manner described in this document.

4 An execution context consists of a set of attributes as described below. Each attribute of
5 an execution context except `[[global-variable-bindings]]` forms a logical stack. Attributes of an
6 execution context are changed when a program construct is evaluated.

7 The following are the attributes of an execution context:

8 `[[self]]` : A logical stack of objects. The object at the top of the stack is called the **current**
9 **self**, to which a *self-expression* evaluates (see §11.5.4.8.4).

10 `[[class-module-list]]` : A logical stack of lists of classes or modules. The class or module at
11 the head of the list which is on the top of the stack is called the **current class or module**.

12 `[[default-method-visibility]]` : A logical stack of visibilities of methods, each of which is one
13 of the **public**, **private**, and **protected** visibility. The top of the stack is called the **current**
14 **visibility**.

15 `[[local-variable-bindings]]` : A logical stack of sets of bindings of local variables. The element
16 at the top of the stack is called the **current set of local variable bindings**. A set of
17 bindings is pushed onto the stack on every entry into a local variable scope (see §9.2), and
18 the top element is removed from the stack on every exit from the scope. The scope with
19 which an element in the stack is associated is called the **scope of the set of local variable**
20 **bindings**.

21 `[[invoked-method-name]]` : A logical stack of names by which methods are invoked.

22 `[[defined-method-name]]` : A logical stack of names with which the invoked methods are
23 defined.

24 `[[block]]` : A logical stack of blocks passed to method invocations. An element of the stack
25 may be **block-not-given**, which indicates that no block is passed to a method invocation.

26 `[[global-variable-bindings]]` : A set of bindings of global variables.

27 The term **unset** is used to describe the state of an attribute which is set to nothing.

28 7.2 The initial state

29 Immediately prior to an execution of a program, the attributes of the execution context is
30 initialized as follows:

31 a) Create an empty set of variable bindings, and set `[[global-variable-bindings]]` to the set of
32 variable bindings.

33 b) Create built-in classes and modules as described in §15.

34 c) Create an empty stack for each attribute of the execution context except `[[global-variable-`
35 `bindings]]`.

36 d) Create a direct instance of the class `Object` and push it onto `[[self]]`.

- 1 e) Create a list containing only the class `Object` and push the list onto `[[class-module-list]]`.
- 2 f) Push the private visibility onto `[[default-visibility]]`.
- 3 g) Push `block-not-given` onto `[[block]]`.

4 8 Lexical structure

5 8.1 General description

6 The source text of a program is first converted into a sequence of input elements, which are
7 either tokens, line terminators, comments, end of program markers, or whitespace. When several
8 prefixes of the input under the converting process have matching productions, the production
9 that matches the longest prefix is selected.

10 8.2 Source text

11 Syntax

12 *source-character* ::
13 [any character in ISO/IEC 646:1991 IRV]

14 A program is represented as a sequence of characters. A conforming processor shall accept any
15 conforming program which consists of characters in ISO/IEC 646:1991 IRV (the International
16 Reference Version), encoded with the octet values as specified in ISO/IEC 646:1991. The support
17 for any other character sets and encodings is unspecified.

18 Terminal symbols are sequences of those characters in ISO/IEC 646:1991 IRV. Control characters
19 in ISO/IEC 646:1991 IRV are represented by two digits in hexadecimal notation prefixed by “0x”,
20 where the first and the second digits respectively represent x and y of the notations of the form
21 x/y in §5.1 of ISO/IEC 646:1991.

22 EXAMPLE “0x0a” represents the character LF, whose bit combination specified in ISO/IEC 646:1991
23 is 0/10.

24 8.3 Line terminators

25 Syntax

26 *line-terminator* ::
27 0x0d? 0x0a

28 A *line-terminator* is ignored when it is used to separate *tokens*. For this reason, except in §8,
29 *line-terminators* are omitted from productions. However, in some cases, the presence or absence
30 of a *line-terminator* changes the meaning of a program.

1 A location of program text where a *line-terminator* shall occur is indicated by the notation “[
2 *line-terminator* here]”. A location of program text where a *line-terminator* shall not occur is
3 indicated by the notation “[no *line-terminator* here]”; however, a conforming processor may
4 ignore the notation where the ignorance does not introduce ambiguity.

5 **EXAMPLE** *statements* are separated by *separators* (see §10.2). The syntax of the *separators* is as
6 follows:

```
7 separator ::  
8     ;  
9     | [line-terminator here]
```

10 The source

```
11     x = 1 + 2  
12     puts x
```

13 is therefore separated into the two statements `x = 1 + 2` and `puts x` by a line-terminator.

14 The source

```
15     x =  
16     1 + 2
```

17 is parsed as the single statement `x = 1 + 2` because `x =` is not a *statement*. However, the source

```
18     x  
19     = 1 + 2
```

20 is not a conforming program because a *line-terminator* shall not occur before `=` in a *single-variable-*
21 *assignment-expression*, and `= 1 + 2` is not a *statement*. The fact that a *line-terminator* shall not occur
22 before `=` is indicated in the syntax of the *single-variable-assignment-expression* as follows (see §11.4.2.2.2):

```
23 single-variable-assignment-expression ::  
24     variable [no line-terminator here] = operator-expression
```

25 8.4 Whitespace

26 Syntax

```
27 whitespace ::  
28     0x09 | 0x0b | 0x0c | 0x0d | 0x20 | line-terminator-escape-sequence
```

```
29 line-terminator-escape-sequence ::  
30     \ 0x0d? 0x0a
```

1 *whitespace* is ignored when it is used to separate *tokens*. For this reason, except in §8, *whitespace*
2 is omitted from productions. However, in some cases, the presence or absence of *whitespace*
3 changes the meaning of a program.

4 A location of program text where a *whitespace* shall occur is indicated by the notation “[*whites-*
5 *pace* here]”. A location of program text where a *whitespace* shall not occur is indicated by the
6 notation “[no *whitespace* here]”. A *line-terminator* shall not occur in the location where a
7 *whitespace* shall not occur. Therefore, this notation also indicates that a *line-terminator* shall
8 not occur.

9 8.5 Comments

10 Syntax

11 *comment* ::

12 *single-line-comment*

13 | *multi-line-comment*

14 *single-line-comment* ::

15 # *comment-content*?

16 *comment-content* ::

17 *line-content*

18 *line-content* ::

19 *source-character* +

20 *multi-line-comment* ::

21 *multi-line-comment-begin-line* *multi-line-comment-line*?

22 *multi-line-comment-end-line*

23 *multi-line-comment-begin-line* ::

24 [beginning of a line] =**begin** *rest-of-begin-end-line*? *line-terminator*

25 *multi-line-comment-end-line* ::

26 [beginning of a line] =**end** *rest-of-begin-end-line*?

27 (*line-terminator* | [end of a program])

28 *rest-of-begin-end-line* ::

29 *whitespace* + *comment-content*

30 *line* ::

31 *comment-content* *line-terminator*

32 *multi-line-comment-line* ::

33 *line* **but not** *multi-line-comment-end-line*

1 The notation “[beginning of a line]” indicates the beginning of a program or the position
2 immediately after a *line-terminator*.

3 Any characters that are considered as *line-terminators* are not allowed within a *line-content*.

4 A *comment* is either a *single-line-comment* or a *multi-line-comment*. A *comment* is considered
5 to be *whitespace*.

6 A *single-line-comment* begins with “#” and continues to the end of the line. A *line-terminator*
7 at the end of the line is not considered to be a part of the comment. A *single-line-comment* can
8 contain any characters except *line-terminators*.

9 A *multi-line-comment* begins with a line beginning with `=begin`, and continues until and in-
10 cluding a line that begins with `=end`. Unlike *single-line-comments*, a *line-terminator* on a
11 *multi-line-comment-end-line*, if any, is considered to be part of the comment.

12 8.6 End-of-program markers

13 Syntax

14 *end-of-program-marker* ::
15 [beginning of a line] `__END__` (*line-terminator* | [end of a program])

16 An *end-of-program-marker* indicates the logical end of a program. Any source characters after
17 an *end-of-program-marker* are not treated as program text.

18 8.7 Tokens

19 8.7.1 General description

20 Syntax

21 *token* ::
22 *keyword*
23 | *identifier*
24 | *punctuator*
25 | *operator*
26 | *literal*

27 8.7.2 Keywords

28 Syntax

```
1  keyword ::
2      __LINE__ | __ENCODING__ | __FILE__ | BEGIN | END | alias | and | begin
3      | break | case | class | def | defined? | do | else | elsif | end
4      | ensure | for | false | if | in | module | next | nil | not | or | redo
5      | rescue | retry | return | self | super | then | true | undef | unless
6      | until | when | while | yield
```

7 Keywords are case-sensitive.

8 8.7.3 Identifiers

9 Syntax

```
10 identifier ::
11     local-variable-identifier
12     | global-variable-identifier
13     | class-variable-identifier
14     | instance-variable-identifier
15     | constant-identifier
16     | method-identifier

17 local-variable-identifier ::
18     ( lowercase-character | _ ) identifier-character*

19 global-variable-identifier ::
20     $ identifier-start-character identifier-character*

21 class-variable-identifier ::
22     @@ identifier-start-character identifier-character*

23 instance-variable-identifier ::
24     @ identifier-start-character identifier-character*

25 constant-identifier ::
26     uppercase-character identifier-character*

27 method-identifier ::
28     method-only-identifier
29     | assignment-like-method-identifier
30     | constant-identifier
31     | local-variable-identifier

32 method-only-identifier ::
33     ( constant-identifier | local-variable-identifier ) ( ! | ? )
```

```

1  assignment-like-method-identifier ::
2      ( constant-identifier | local-variable-identifier ) =

3  identifier-character ::
4      lowercase-character
5      | uppercase-character
6      | decimal-digit
7      | _

8  identifier-start-character ::
9      lowercase-character
10     | uppercase-character
11     | _

12 uppercase-character ::
13     A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R
14     | S | T | U | V | W | X | Y | Z

15 lowercase-character ::
16     a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r
17     | s | t | u | v | w | x | y | z

18 decimal-digit ::
19     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

20 An *identifier* is a sequence of *identifier-characters* optionally prefixed by one of “\$”, “@@”, or
21 “@”, and optionally postfixed by one of “?”, “!”, or “=”.

22 A *global-variable-identifier* begins with “\$”. A *class-variable-identifier* begins with “@@”. An
23 *instance-variable-identifier* begins with “@”. A *constant-identifier* begins with an *uppercase-*
24 *character*.

25 A *local-variable-identifier* begins with a *lowercase-character* or “_”. A *method-identifier* is a
26 *constant-identifier* or a *local-variable-identifier* optionally followed by one of “?”, “!”, or “=”.

27 8.7.4 Punctuators

28 Syntax

```

29 punctuator ::
30     [ | ] | ( | ) | { | } | :: | , | ; | .. | ... | ? | : | =>

```

31 8.7.5 Operators

32 Syntax

```

1  operator ::
2      ! | != | !~ | && | ||
3      | operator-method-name
4      | =
5      | assignment-operator

6  operator-method-name ::
7      ^ | & | | | <=> | == | === | =~ | > | >= | < | <= | << | >> | + | -
8      | * | / | % | ** | ~ | +@ | -@ | [] | []= | '

9  assignment-operator ::
10     assignment-operator-name =

11  assignment-operator-name ::
12     + | - | * | ** | / | ^ | % | << | >> | & | && | || | |

```

13 8.7.6 Literals

14 8.7.6.1 General description

```

15  literal ::
16     numeric-literal
17     | string-literal
18     | array-literal
19     | regular-expression-literal
20     | symbol

```

21 8.7.6.2 Numeric literals

22 Syntax

```

23  numeric-literal ::
24     signed-number
25     | unsigned-number

26  unsigned-number ::
27     integer-literal
28     | float-literal

29  integer-literal ::
30     decimal-integer-literal
31     | binary-integer-literal

```

```

1      | octal-integer-literal
2      | hexadecimal-integer-literal

3  decimal-integer-literal ::
4      unprefixed-decimal-integer-literal
5      | prefixed-decimal-integer-literal

6  unprefixed-decimal-integer-literal ::
7      0
8      | decimal-digit-without-zero ( _? decimal-digit )*

9  prefixed-decimal-integer-literal ::
10     0 ( d | D ) digit-decimal-part

11  digit-decimal-part ::
12     decimal-digit ( _? decimal-digit )*

13  binary-integer-literal ::
14     0 ( b | B ) binary-digit ( _? binary-digit )*

15  octal-integer-literal ::
16     0 ( _ | o | O )? octal-digit ( _? octal-digit )*

17  hexadecimal-integer-literal ::
18     0 ( x | X ) hexadecimal-digit ( _? hexadecimal-digit )*

19  float-literal ::
20     float-literal-without-exponent
21     | float-literal-with-exponent

22  float-literal-without-exponent ::
23     unprefixed-decimal-integer-literal . digit-decimal-part

24  float-literal-with-exponent ::
25     significand-part exponent-part

26  significand-part ::
27     float-literal-without-exponent
28     | unprefixed-decimal-integer-literal

29  exponent-part ::
30     ( e | E ) ( + | - )? digit-decimal-part

31  signed-number ::
32     ( + | - ) unsigned-number

```

1 *decimal-digit-without-zero* ::
2 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3 *octal-digit* ::
4 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

5 *binary-digit* ::
6 0 | 1

7 *hexadecimal-digit* ::
8 *decimal-digit* | a | b | c | d | e | f | A | B | C | D | E | F

9 If the previous token of a *signed-number* is a *local-variable-identifier*, both of the following
10 conditions shall hold:

- 11 • There is at least one *whitespace* between the *local-variable-identifier* and the *signed-number*.
- 12 • The *local-variable-identifier* is not a reference to a local variable (see §11.5.4.7.2).

13 EXAMPLE -123 in the following program is a *signed-number*.

14 x -123

15 In the above program, the method x is invoked with the value of -123 as the argument.

16 However, -123 in the following program is separated into the two tokens - and 123 because there is no
17 *whitespace* between x and -123.

18 x-123

19 In the above program, the method - is invoked on the value of x with the value of 123 as the argument.

20 -123 in the following program is also separated into the two tokens - and 123 because x is a reference to
21 a local variable.

22 x = 1
23 x -123

24 Semantics

25 A *numeric-literal* evaluates to either an instance of the class `Integer` or a direct instance of the
26 class `Float`.

27 An *unsigned-number* of the form *integer-literal* evaluates to an instance of the class `Integer`
28 whose value is the value of one of the alternatives in the *integer-literal* production.

29 An *unsigned-number* of the form *float-literal* evaluates to a direct instance of the class `Float`
30 whose value is the value of one of the alternatives in the *float-literal* production.

31 A *signed-number* which begins with “+” evaluates to an instance represented by the *unsigned-*
32 *number*. A *signed-number* which begins with “-” evaluates to an instance of the class `Integer` or

1 a direct instance of the class `Float` whose value is the negated value of the instance represented
2 by the *unsigned-number*.

3 The value of an *integer-literal*, a *decimal-integer-literal*, a *float-literal*, or a *significand-part* is the
4 value of one of the alternatives in their production.

5 The value of a *unprefixed-decimal-integer-literal* is either 0 or the value of a sequence of charac-
6 ters, which consist of a *decimal-digit-without-zero* followed by sequence of *decimal-digits*, ignoring
7 interleaving “_”s, computed using base 10.

8 The value of a *prefixed-decimal-integer-literal* is the value of the *digit-decimal-part*.

9 The value of a *digit-decimal-part* is the value of the sequence of *decimal-digits*, ignoring inter-
10 leaving “_”s, computed using base 10.

11 The value of a *binary-integer-literal* is the value of the sequence of *binary-digits*, ignoring inter-
12 leaving “_”s, computed using base 2.

13 The value of an *octal-integer-literal* is the value of the sequence of *octal-digits*, ignoring inter-
14 leaving “_”s, computed using base 8.

15 The value of a *hexadecimal-integer-literal* is the value of the sequence of *hexadecimal-digits*,
16 ignoring interleaving “_”s, computed using base 16.

17 The value of a *float-literal-without-exponent* is the value of the *unprefixed-decimal-integer-literal*
18 plus the value of the *digit-decimal-part* times 10^{-n} where n is the number of *decimal-digits* of
19 the *digit-decimal-part*.

20 The value of an *float-literal-with-exponent* is the value of the *significand-part* times 10^n where n
21 is the value of the *exponent-part*.

22 The value of an *exponent-part* is the negative value of the *digit-decimal-part* if “-” occurs,
23 otherwise, it is the value of the *digit-decimal-part*.

24 There is no limitation on the maximum magnitude for the value of an *integer-literal*; however the
25 actual values computable depend on resource limitations, and the behavior when the resource
26 limits are exceeded is implementation-defined. The precision of the value of a *float-literal* is
27 implementation-defined; however, if the underlying platform of a conforming processor supports
28 IEC 60559:1989, the representation of an instance of the class `Float` shall be the 64-bit double
29 format as specified in §3.2.2 of IEC 60559:1989. The value of a *float-literal* is rounded to fit in
30 the representation of an instance of the class `Float` in an implementation-defined way.

31 8.7.6.3 String literals

32 8.7.6.3.1 General description

33 Syntax

34 *string-literal* ::
35 *single-quoted-string*
36 | *double-quoted-string*
37 | *quoted-non-expanded-literal-string*

1 | *quoted-expanded-literal-string*
2 | *here-document*
3 | *external-command-execution*

4 **Semantics**

5 A *string-literal* evaluates to a direct instance of the class `String`.

6 **8.7.6.3.2 Single quoted strings**

7 **Syntax**

8 *single-quoted-string* ::
9 ' *single-quoted-string-character** '

10 *single-quoted-string-character* ::
11 *single-quoted-string-non-escaped-character*
12 | *single-quoted-escape-sequence*

13 *single-quoted-escape-sequence* ::
14 *single-escape-character-sequence*
15 | *single-quoted-string-non-escaped-character-sequence*

16 *single-escape-character-sequence* ::
17 *single-quoted-string-meta-character*

18 *single-quoted-string-non-escaped-character-sequence* ::
19 *single-quoted-string-non-escaped-character*

20 *single-quoted-string-meta-character* ::
21 ' | \

22 *single-quoted-string-non-escaped-character* ::
23 *source-character* **but not** *single-escaped-character*

24 **Semantics**

25 A *single-quoted-string* consists of zero or more characters enclosed by single quotes. The sequence
26 of *single-quoted-string-characters* within the pair of single quotes represents the content of a
27 string as it occurs in program text literally, except for *single-escape-character-sequences*. The
28 sequence “\” represents “\”. The sequence “\” represents “”.

29 **8.7.6.3.3 Double quoted strings**

30 **Syntax**

```

1  double-quoted-string ::
2      " double-quoted-string-character* "

3  double-quoted-string-character ::
4      source-character but not ( " | # | \ )
5      | # [lookahead ∉ { $, @, { }]
6      | double-escape-sequence
7      | interpolated-character-sequence

8  double-escape-sequence ::
9      simple-escape-sequence
10     | non-escaped-sequence
11     | line-terminator-escape-sequence
12     | octal-escape-sequence
13     | hex-escape-sequence
14     | control-escape-sequence

15 simple-escape-sequence ::
16     \ double-escaped-character

17 non-escaped-sequence ::
18     \ non-escaped-double-quoted-string-character

19 line-terminator-escape-sequence ::
20     \ line-terminator

21 non-escaped-double-quoted-string-character ::
22     source-character but not ( alpha-numeric-character | line-terminator )

23 double-escaped-character ::
24     \ | n | t | r | f | v | a | e | b | s

25 octal-escape-sequence ::
26     \ octal-digit ( octal-digit octal-digit? )?

27 hex-escape-sequence ::
28     \ x hexadecimal-digit hexadecimal-digit?

29 control-escape-sequence ::
30     \ ( C - | c ) control-escaped-character

31 control-escaped-character ::
32     double-escape-sequence
33     | ?

```

1 | *source-character* **but not** (\ | ?)

2 *interpolated-character-sequence* ::
3 # *global-variable-identifier*
4 | # *class-variable-identifier*
5 | # *instance-variable-identifier*
6 | # { *compound-statement* }

7 *alpha-numeric-character* ::
8 *uppercase-character*
9 | *lowercase-character*
10 | *decimal-digit*

11 Semantics

12 A *double-quoted-string* consists of zero or more characters enclosed by double quotes. The se-
13 quence of *double-quoted-string-characters* within the pair of double quotes represents the content
14 of a string.

15 Except for a *double-escape-sequence* and an *interpolated-character-sequence*, a *double-quoted-*
16 *string-character* represents a character as it occurs in program text.

17 A *simple-escape-sequence* represents a character as shown in Table 1.

Table 1 – Simple escape sequences

Escape sequence	Character code
\\	0x5c
\n	0x0a
\t	0x09
\r	0x0d
\f	0x0c
\v	0x0b
\a	0x07
\e	0x1b
\b	0x08
\s	0x20

18 An *octal-escape-sequence* represents a character the code of which is the value of the sequence
19 of *octal-digits* computed using base 8.

20 A *hex-escape-sequence* represents a character the code of which is the value of the sequence of
21 *hexadecimal-digits* computed using base 16.

22 A *non-escaped-sequence* represents an implementation-defined character.

23 A *line-terminator-escape-sequence* is used to break the content of a string into separate lines

1 in program text without inserting a *line-terminator* into the string. A *line-terminator-escape-*
2 *sequence* does not count as a character of the string.

3 A *control-escape-sequence* represents a character the code of which is computed by performing a
4 bitwise AND operation between 0x9f and the code of the character represented by the *control-*
5 *escaped-character*, except when the *control-escaped-character* is ?, in which case, the *control-*
6 *escape-sequence* represents a character the code of which is 127.

7 An *interpolated-character-sequence* is a part of a *string-literal* which is dynamically evalu-
8 ated when the *string-literal* in which it is embedded is evaluated. The *interpolated-character-*
9 *sequences* within a *string-literal* are evaluated in the order in which they occur in program
10 text.

11 The value of a *string-literal* which contains *interpolated-character-sequences* is a direct instance
12 of the class `String` the content of which is made from the *string-literal* where each occurrence
13 of *interpolated-character-sequence* is replaced by the content of an instance of the class `String`
14 which is the dynamically evaluated value of the *interpolated-character-sequence*.

15 An *interpolated-character-sequence* is evaluated as follows:

16 a) If it is of the form `# global-variable-identifier`, evaluate the *global-variable-identifier* (see
17 §11.5.4.4). Let *V* be the resulting value.

18 b) If it is of the form `# class-variable-identifier`, evaluate the *class-variable-identifier* (see
19 §11.5.4.5). Let *V* be the resulting value.

20 c) If it is of the form `# instance-variable-identifier`, evaluate the *instance-variable-identifier*
21 (see §11.5.4.6). Let *V* be the resulting value.

22 d) If it is of the form `# { compound-statement }`, evaluate the *compound-statement* (see §10.2).
23 Let *V* be the resulting value.

24 e) If *V* is an instance of the class `String`, *V* is the value of *interpolated-character-sequence*.

25 f) Otherwise, invoke the method `to_s` on *V* with an empty list of arguments. Let *S* be the
26 resulting value.

27 g) If *S* is an instance of the class `String`, *S* is the value of *interpolated-character-sequence*.

28 h) Otherwise, the value of *interpolated-character-sequence* is an instance of the class `String`,
29 the content of which is implementation-defined.

30 8.7.6.3.4 Quoted non-expanded literal strings

31 Syntax

32 *quoted-non-expanded-literal-string* ::
33 `%q literal-beginning-delimiter non-expanded-literal-string* literal-ending-delimiter`

34 *non-expanded-literal-string* ::
35 *non-expanded-literal-character*
36 | *non-expanded-delimited-string*


```

1  non-expanded-delimited-string ::
2      literal-beginning-delimiter non-expanded-literal-string* literal-ending-delimiter

3  non-expanded-literal-character ::
4      non-escaped-literal-character
5      | non-expanded-literal-escape-sequence

6  non-escaped-literal-character ::
7      source-character but not quoted-literal-escape-character

8  non-expanded-literal-escape-sequence ::
9      non-expanded-literal-escape-character-sequence
10     | non-escaped-non-expanded-literal-character-sequence

11 non-expanded-literal-escape-character-sequence ::
12     \ non-expanded-literal-escaped-character

13 non-expanded-literal-escaped-character ::
14     literal-beginning-delimiter
15     | literal-ending-delimiter
16     | \

17 quoted-literal-escape-character ::
18     non-expanded-literal-escaped-character

19 non-escaped-non-expanded-literal-character-sequence ::
20     \ non-escaped-non-expanded-literal-character

21 non-escaped-non-expanded-literal-character ::
22     source-character but not non-expanded-literal-escaped-character

```

23 The *literal-beginning-delimiter* of a *non-expanded-delimited-string* shall be the same character
24 as the *literal-beginning-delimiter* of the *quoted-non-expanded-literal-string*.

25 A *literal-ending-delimiter* shall be the same character as the corresponding *literal-beginning-*
26 *delimiter*, except when the *literal-beginning-delimiter* is one of the characters on the left in
27 Table 2. In that case, the *literal-ending-delimiter* is the corresponding character on the right in
28 Table 2.

29 The production *non-expanded-delimited-string* applies only when the *literal-beginning-delimiter*
30 is one of the characters of *matching-literal-beginning-delimiter*.

31 Semantics

32 A *non-expanded-literal-string* represents the content of a string as it occurs in program text
33 literally, except for *non-expanded-literal-escape-character-sequences*.

Table 2 – Matching *literal-beginning-delimiter* *literal-ending-delimiter*

<i>literal-beginning-delimiter</i>	<i>literal-ending-delimiter</i>
{	}
()
[]
<	>

1 A *non-expanded-literal-escape-character-sequence* represents a character as follows. The se-
 2 quence “\” represents “\”; the sequence *\literal-beginning-delimiter*, a *literal-beginning-delimiter*;
 3 the sequence *\literal-ending-delimiter*, a *literal-ending-delimiter*.

4 8.7.6.3.5 Quoted expanded literal strings

5 Syntax

6 *quoted-expanded-literal-string* ::
 7 % Q? *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

8 *expanded-literal-string* ::
 9 *expanded-literal-character*
 10 | *expanded-delimited-string*

11 *expanded-literal-character* ::
 12 *non-escaped-literal-character* **but not** #
 13 | # [lookahead ∉ { \$, @, { }]
 14 | *double-escape-sequence*
 15 | *interpolated-character-sequence*

16 *expanded-delimited-string* ::
 17 *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

18 *literal-beginning-delimiter* ::
 19 *source-character* **but not** *alpha-numeric-character*

20 *literal-ending-delimiter* ::
 21 *source-character* **but not** *alpha-numeric-character*

22 *matching-literal-beginning-delimiter* ::
 23 (| { | < | [

24 The *literal-beginning-delimiter* of an *expanded-delimited-string* shall be the same character as
 25 the *literal-beginning-delimiter* of the *quoted-expanded-literal-string*.

1 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.7.6.3.4.

2 The production *expanded-delimited-string* applies only when the *literal-beginning-delimiter* is
3 one of the characters of *matching-literal-beginning-delimiter*.

4 **Semantics**

5 A *expanded-literal-string* represents the content of a string.

6 A character in an *expanded-literal-string* other than a *double-escape-sequence* or an *interpolated-*
7 *character-sequence* represents a character as it occurs in program text. A *double-escape-sequence*
8 and an *interpolated-character-sequence* represent characters as described in §8.7.6.3.3.

9 **8.7.6.3.6 Here documents**

10 **Syntax**

11 *here-document* ::
12 *heredoc-start-line* *heredoc-body* *heredoc-end-line*

13 *heredoc-start-line* ::
14 *heredoc-signifier* *rest-of-line*

15 *heredoc-signifier* ::
16 << *heredoc-delimiter-specifier*

17 *rest-of-line* ::
18 *line-content?* *line-terminator*

19 *heredoc-body* ::
20 *heredoc-body-line**

21 *heredoc-body-line* ::
22 *line* **but not** *heredoc-end-line*

23 **Semantics**

24 A *here-document* is represented by several lines of program text, and evaluates to a direct
25 instance of the class **String** or the value of the invocation of the method ‘.

26 The *heredoc-signifier*, the *heredoc-body*, and the *heredoc-end-line* in a *here-document* are treated
27 as a unit and considered to be a single token occurring at the place where the *heredoc-signifier*
28 occurs. The first character of the *rest-of-line* becomes the head of the input after the *here-*
29 *document* has been processed.

30 The object to which *here-document* evaluates is either a direct instance *S* of the class **String**
31 whose content is represented by the *heredoc-body* or the value of the invocation of the method
32 ‘ with *S* as the only argument.

1 The form of the *heredoc-delimiter-specifier* determines both the form of the *heredoc-end-line* and
2 the way in which the *here-document* is processed, as described below.

3 **Syntax**

4 *heredoc-delimiter-specifier* ::
5 -? *heredoc-delimiter*

6 *heredoc-delimiter* ::
7 *non-quoted-delimiter*
8 | *single-quoted-delimiter*
9 | *double-quoted-delimiter*
10 | *command-quoted-delimiter*

11 *non-quoted-delimiter* ::
12 *non-quoted-delimiter-identifier*

13 *non-quoted-delimiter-identifier* ::
14 *identifier-character**

15 *single-quoted-delimiter* ::
16 ' *single-quoted-delimiter-identifier* '

17 *single-quoted-delimiter-identifier* ::
18 (*source-character* **but not** ')*

19 *double-quoted-delimiter* ::
20 " *double-quoted-delimiter-identifier* "

21 *double-quoted-delimiter-identifier* ::
22 (*source-character* **but not** ")*

23 *command-quoted-delimiter* ::
24 ' *command-quoted-delimiter-identifier* '

25 *command-quoted-delimiter-identifier* ::
26 (*source-character* **but not** ')*

27 *heredoc-end-line* ::
28 *indented-heredoc-end-line*
29 | *non-indented-heredoc-end-line*

30 *indented-heredoc-end-line* ::
31 [beginning of a line] *whitespace** *heredoc-delimiter-identifier* *line-terminator*

```

1  non-indented-heredoc-end-line ::
2      [ beginning of a line ] heredoc-delimiter-identifier line-terminator

3  heredoc-delimiter-identifier ::
4      non-quoted-delimiter-identifier
5      | single-quoted-delimiter-identifier
6      | double-quoted-delimiter-identifier
7      | command-quoted-delimiter-identifier

```

8 Any characters that are considered as *line-terminators* are not allowed within a *single-quoted-delimiter-identifier*, a *double-quoted-delimiter-identifier*, or a *command-quoted-delimiter-identifier*.

10 The form of a *heredoc-end-line* depends on the presence or absence of the beginning “-” of the *heredoc-delimiter-specifier*.

12 If the *heredoc-delimiter-specifier* begins with “-”, a line of the form *indented-heredoc-end-line* is treated as the *heredoc-end-line*, otherwise, a line of the form *non-indented-heredoc-end-line* is treated as the *heredoc-end-line*. In both forms, the *heredoc-delimiter-identifier* shall be the same sequence of characters as it occurs in the corresponding part of *heredoc-delimiter*.

16 If the *heredoc-delimiter* is of the form *non-quoted-delimiter*, the *heredoc-delimiter-identifier* shall be the same sequence of characters as the *non-quoted-delimiter-identifier*; if it is of the form *single-quoted-delimiter*, the *single-quoted-delimiter-identifier*; if it is of the form of *double-quoted-delimiter*, the *double-quoted-delimiter-identifier*; if it is of the form of *command-quoted-delimiter*, the *command-quoted-delimiter-identifier*.

21 Semantics

22 The object to which a *here-document* evaluates is created as follows:

- 23 a) Create a direct instance of the class **String** from the *heredoc-body*, the treatment of which
24 depends on the form of the *heredoc-delimiter* as follows:
- 25 • If *heredoc-delimiter* is of the form *single-quoted-delimiter*, the *heredoc-body* is treated
26 as a sequence of *source-characters* as it occurs in program text literally.
 - 27 • If *heredoc-delimiter* is in any of the forms *non-quoted-delimiter*, *double-quoted-delimiter*,
28 or *command-quoted-delimiter*, the *heredoc-body* is treated as a sequence of *double-*
29 *quoted-string-characters* as described in §8.7.6.3.3.

30 Let S be that instance of the class **String**.

- 31 b) If the *heredoc-delimiter* is not of the form *command-quoted-delimiter*, let V be S .
- 32 c) Otherwise, invoke the method ‘ ‘ on the current self with the list of arguments whose only
33 element is S . Let V be the resulting value of the method invocation.
- 34 d) V is the object to which the *here-document* evaluates.

1 8.7.6.3.7 External command execution

2 Syntax

3 *external-command-execution* ::
4 *backquoted-external-command-execution*
5 | *quoted-external-command-execution*

6 *backquoted-external-command-execution* ::
7 ‘ *backquoted-external-command-execution-character** ‘

8 *backquoted-external-command-execution-character* ::
9 *source-character* **but not** (‘ | # | \)
10 | # [lookahead ∉ { \$, @, { }]
11 | *double-escape-sequence*
12 | *interpolated-character-sequence*

13 *quoted-external-command-execution* ::
14 %*x* *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

15 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.7.6.3.

16 Semantics

17 An *external-command-execution* is a form to invoke the method “‘”.

18 An *external-command-execution* is evaluated as follows:

- 19 a) If the *external-command-execution* is of the form *backquoted-external-command-execution*,
20 construct a direct instance of the class **String** *S* whose content is a sequence of characters
21 represented by *backquoted-external-command-execution-characters*. A *backquoted-external-*
22 *command-execution-character* other than a *double-escape-sequence* or an *interpolated-character-*
23 *sequence* represents a character as it occurs in program text. A *double-escape-sequence* and
24 an *interpolated-character-sequence* represent characters as described in §8.7.6.3.3.
- 25 b) If the *external-command-execution* is of the form *quoted-external-command-execution*, con-
26 struct a direct instance of the class **String** *S* by replacing “%*x*” with “%Q” and evaluating
27 the resulting *quoted-expanded-literal-string* as described in §8.7.6.3.5.
- 28 c) Invoke the method “‘” on the current self with a list of arguments whose only element is *S*.
- 29 d) The resulting value is the value of the *external-command-execution*.

30 8.7.6.4 Array literals

31 Syntax

```

1  array-literal ::
2      quoted-non-expanded-array-constructor
3      | quoted-expanded-array-constructor

4  quoted-non-expanded-array-constructor ::
5      %w literal-beginning-delimiter non-expanded-array-content literal-ending-delimiter

6  non-expanded-array-content ::
7      quoted-array-item-separator-list? non-expanded-array-item-list?
8      quoted-array-item-separator-list?

9  non-expanded-array-item-list ::
10     non-expanded-array-item ( quoted-array-item-separator-list non-expanded-array-item )*

11 quoted-array-item-separator-list ::
12     quoted-array-item-separator +

13 quoted-array-item-separator ::
14     whitespace
15     | line-terminator

16 non-expanded-array-item ::
17     non-expanded-array-item-character +

18 non-expanded-array-item-character ::
19     non-escaped-array-item-character
20     | non-expanded-array-escape-sequence

21 non-escaped-array-item-character ::
22     non-escaped-array-character
23     | matching-literal-delimiter

24 non-escaped-array-character ::
25     non-escaped-literal-character but not quoted-array-item-separator

26 matching-literal-delimiter ::
27     ( | { | < | [ | ) | } | > | ]

28 non-expanded-array-escape-sequence ::
29     non-expanded-literal-escape-sequence but not escaped-quoted-array-item-separator
30     | escaped-quoted-array-item-separator

31 escaped-quoted-array-item-separator ::
32     \ quoted-array-item-separator

```

```

1  quoted-expanded-array-constructor ::
2      %W literal-beginning-delimiter expanded-array-content literal-ending-delimiter

3  expanded-array-content ::
4      quoted-array-item-separator-list? expanded-array-item-list?
5      quoted-array-item-separator-list?

6  expanded-array-item-list ::
7      expanded-array-item ( quoted-array-item-separator-list expanded-array-item )*

8  expanded-array-item ::
9      expanded-array-item-character +

10 expanded-array-item-character ::
11     non-escaped-array-item-character but not #
12     | # [lookahead ∉ { $, @, { }]
13     | expanded-array-escape-sequence
14     | interpolated-character-sequence

15 expanded-array-escape-sequence ::
16     double-escape-sequence but not escaped-quoted-array-item-separator
17     | escaped-quoted-array-item-separator

```

18 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.7.6.3.

19 When the *literal-beginning-delimiter* is one of the *matching-literal-beginning-delimiter*, the *quoted-*
20 *non-expanded-array-constructor* and the *quoted-expanded-array-constructor* is determined as fol-
21 lows.

22 Let N be 0. For each character C which appears after “%w” or “%W”, take the following steps.

- 23 a) If C is a *literal-beginning-delimiter* which is not prefixed by a “\”, increment N by 1.
- 24 b) If C is a *literal-ending-delimiter* which is not prefixed by a “\”, decrement N by 1.
- 25 c) If N is 0 and C is the *literal-ending-delimiter*, terminate these steps.

26 The *literal-ending-delimiter* in Step c) is the *literal-ending-delimiter* of the *quoted-non-expanded-*
27 *array-constructor* or the *quoted-expanded-array-constructor*.

28 Semantics

29 An *array-literal* evaluates to a direct instance of the class **Array** as follows:

- 30 a) A *quoted-non-expanded-array-constructor* is evaluated as follows:
 - 31 1) Create an empty direct instance of the class **Array**. Let A be the instance.

1 2) If *non-expanded-array-item-list* is present, for each *non-expanded-array-item* of the *non-*
2 *expanded-array-item-list*, take the following steps:

3 i) Create a direct instance of the class **String** *S*, the content of which is represented
4 by the sequence of *non-expanded-array-item-characters*.

5 A *non-expanded-array-item-character* represents itself, except in the case of a
6 *non-expanded-array-escape-sequence*. A *non-expanded-array-escape-sequence* rep-
7 resents a character as described in §8.7.6.3.4, except in the case of an *escaped-*
8 *quoted-array-item-separator*. An *escaped-quoted-array-item-separator* represents a
9 *quoted-array-item-separator*.

10 ii) Append *S* to *A*.

11 3) The value of the *quoted-non-expanded-array-constructor* is *A*.

12 b) A *quoted-expanded-array-constructor* is evaluated as follows:

13 1) Create an empty direct instance of the class **Array**. Let *A* be the instance.

14 2) If *expanded-array-item-list* is present, process each *expanded-array-item* of the *expanded-*
15 *array-item-list* as follows:

16 i) Create a direct instance of the class **String** *S*, the content of which is represented
17 by the sequence of *expanded-array-item-characters*.

18 An *expanded-array-item-character* represents itself, except in the case of an *expanded-*
19 *array-escape-sequence* and an *interpolated-character-sequence*. An *expanded-array-*
20 *escape-sequence* represents a character as described in §8.7.6.3.3, except in the
21 case of an *escaped-quoted-array-item-separator*. An *escaped-quoted-array-item-*
22 *separator* represents a *quoted-array-item-separator*. An *interpolated-character-*
23 *sequence* represents a sequence of characters as described in §8.7.6.3.3.

24 ii) Append *S* to *A*.

25 3) The value of the *quoted-expanded-array-constructor* is *A*.

26 8.7.6.5 Regular expression literals

27 Syntax

28 *regular-expression-literal* ::
29 / *regular-expression-body* / *regular-expression-option**
30 | %**r** *literal-beginning-delimiter* *expanded-literal-string**
31 *literal-ending-delimiter* *regular-expression-option**

32 *regular-expression-body* ::
33 *regular-expression-character**

34 *regular-expression-character* ::
35 *source-character* **but not** (/ | # | \)

```
1      | \ \
2      | # [lookahead ∉ { $, @, { }]
3      | line-terminator-escape-sequence
4      | interpolated-character-sequence
```

```
5 regular-expression-option ::
6     i | m
```

7 Within an *expanded-literal-string*, a *literal-beginning-delimiter* shall be the same character as
8 the *literal-beginning-delimiter* of a *regular-expression-literal*.

9 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.7.6.3.4.

10 If a *regular-expression-literal* of the form / *regular-expression-body* / *regular-expression-option**
11 is the first argument (see §11.3.2), the first character of the *regular-expression-body* shall not be
12 *whitespace*.

13 Semantics

14 A *regular-expression-literal* evaluates to a direct instance of the class **Regexp**.

15 The pattern of an instance of the class **Regexp** resulting from a *regular-expression-literal* is the
16 string which *regular-expression-characters* or *expanded-literal-strings* represent. If the string
17 cannot be derived from the *pattern* (see §15.2.15.4), the evaluation of the program shall be
18 terminated and a syntax error shall be reported.

19 A *regular-expression-character* other than the sequence \ \, a *line-terminator-escape-sequence*,
20 or *interpolated-character-sequence* represents themselves. A *expanded-literal-string* other than a
21 *line-terminator-escape-sequence* or *interpolated-character-sequence* represents themselves.

22 The sequence \ \ of *regular-expression-character* represents a single character \.

23 A *line-terminator-escape-sequence* in a *regular-expression-character* and an *expanded-literal-*
24 *string* is ignored in the resulting pattern of an instance of the class **Regexp**.

25 An *interpolated-character-sequence* in a *regular-expression-literal* and an *expanded-literal-string*
26 is evaluated as described in §8.7.6.3.3, and represents a string which is the content of the resulting
27 an instance of the class **String**.

28 A *regular-expression-option* specifies the ignorecase and the multiline properties of an instance of
29 the class **Regexp** resulting from a *regular-expression-literal*. If **i** is present in a *regular-expression-*
30 *option*, the ignorecase property of the resulting instance of the class **Regexp** is set to true. If **m**
31 is present in a *regular-expression-option*, the multiline property of the resulting instance of the
32 class **Regexp** is set to true.

33 The grammar for a pattern of an instance of the class **Regexp** created from a *regular-expression-*
34 *literal* is described in §15.2.15.

1 8.7.6.6 Symbol literals

2 Syntax

```
3  symbol ::  
4      symbol-literal  
5      | dynamic-symbol  
  
6  symbol-literal ::  
7      : symbol-name  
  
8  dynamic-symbol ::  
9      : single-quoted-string  
10     | : double-quoted-string  
11     | %s literal-beginning-delimiter non-expanded-literal-string* literal-ending-delimiter  
  
12  symbol-name ::  
13     method-identifier  
14     | operator-method-name  
15     | keyword  
16     | instance-variable-identifier  
17     | global-variable-identifier  
18     | class-variable-identifier
```

19 *single-quoted-strings*, *double-quoted-strings*, and *non-expanded-literal-strings* shall not contain
20 any sequences which represent the character 0x00.

21 Within a *non-expanded-literal-string*, *literal-beginning-delimiter* shall be the same character as
22 the *literal-beginning-delimiter* of the *dynamic-symbol*.

23 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.7.6.3.4.

24 Semantics

25 A *symbol* evaluates to a direct instance of the class `Symbol`. A *symbol-literal* evaluates to a direct
26 instance of the class `Symbol` whose name is the *symbol-name*. A *dynamic-symbol* evaluates to a
27 direct instance of the class `Symbol` whose name is the content of an instance of the class `String`
28 which is the value of the *single-quoted-string*(see §8.7.6.3.2), *double-quoted-string*(see §8.7.6.3.3),
29 or *non-expanded-literal-string*(see §8.7.6.3.4).

30 9 Scope of variables

31 9.1 General description

32 A **scope** is a region of a program text with which a set of bindings of variables is associated.

1 9.2 Local variables

2 A local variable is referred to by a *local-variable-identifier*.

3 Scopes for local variables are introduced by the following program constructs:

- 4 • *program* (see §10.1)
- 5 • *class-body* (see §13.2.2)
- 6 • *module-body* (see §13.1.2)
- 7 • *eigenclass-body* (see §13.4.2)
- 8 • *method-definition* (see §13.3.1) and *singleton-method-definition* (see §13.4.3), for both of
9 which the scope starts with the *method-parameter-part* and continues up to and including
10 the *method-body*.
- 11 • *block* (see §11.3.3)

12 Let P be any of the above program constructs. Let S be the region of P excluding all the regions
13 of any of the above program constructs (except *block*) nested within P . Then, S is the **local**
14 **variable scope** which corresponds to the program construct P .

15 The scope of a local variable is the local variable scope whose set of local variable bindings
16 contains the binding of the local variable, which is resolved as described below.

17 When a *local-variable-identifier* which is a reference to a local variable occurs, the binding of
18 the local variable is resolved as follows:

- 19 a) Let N be the *local-variable-identifier*. Let B be the current set of local variable bindings.
- 20 b) Let S be the scope of B .
- 21 c) If a binding with name N exists in B , that binding is the resolved binding.
- 22 d) If a binding with name N does not exist in B :
 - 23 • If S is a local variable scope which corresponds to a *block*:
 - 24 1) If the *local-variable-identifier* occurs as a *left-hand-side* of a *block-parameter-list*,
25 whether to proceed to the next step or not is implementation-defined.
 - 26 2) Replace B with the element immediately below the current B on \llbracket local-variable-
27 bindings \rrbracket , and continue searching for a binding with name N from Step b.
 - 28 • Otherwise, a binding is considered not resolved.

29 9.3 Global variables

30 The scope of global variables is global in the sense that they are accessible everywhere in a
31 conforming program. Global variable bindings are created in \llbracket global-variable-bindings \rrbracket .

1 10 Program structure

2 10.1 Program

3 Syntax

4 *program* ::
5 *compound-statement*

6 Semantics

7 A *program* is evaluated as follows:

- 8 a) Push an empty set of bindings onto \llbracket local-variable-bindings \rrbracket .
- 9 b) Evaluate the *compound-statement*.
- 10 c) The resulting value is the value of the *program*.
- 11 d) Restore the execution context by removing the element from the top of \llbracket local-variable-
- 12 bindings \rrbracket , even when an exception is raised and not handled during Step b.

13 10.2 Compound statement

14 Syntax

15 *compound-statement* ::
16 *statement-list?* *separator-list?*

17 *statement-list* ::
18 *statement* (*separator-list* *statement*)*

19 *separator-list* ::
20 *separator* ;*

21 *separator* ::
22 ;
23 | [*line-terminator* here]

24 Semantics

25 A *compound-statement* is evaluated as follows:

- 26 a) If the *statement-list* is omitted, the value of the *compound-statement* is **nil**.
- 27 b) If the *statement-list* is present, evaluate each *statement* of the *statement-list* in the order it
- 28 appears in the program text.

- 1 c) If one of the *statements* of the *statement-list* is terminated by a *jump-expression*, terminate
2 the evaluation of the *statement-list* immediately. None of the following *statements* of the
3 *statement-list* is evaluated. In this case, the value of the *compound-statement* is undefined.
- 4 d) If none of the *statements* of the *statement-list* is terminated by a *jump-expression*, the value
5 of the *compound-statement* is the value of the last *statement* of the *statement-list*.

6 11 Expressions

7 11.1 General description

8 Syntax

9 *expression* ::
10 *keyword-logical-expression*

11 Semantics

12 See §11.2 for *keyword-logical-expression*.

13 11.2 Logical expressions

14 Syntax

15 *keyword-logical-expression* ::
16 *keyword-NOT-expression*
17 | *keyword-AND-expression*
18 | *keyword-OR-expression*

19 *keyword-NOT-expression* ::
20 *method-invocation-without-parentheses*
21 | *operator-expression*
22 | *logical-NOT-with-method-invocation-without-parentheses*
23 | **not** *keyword-NOT-expression*

24 *logical-NOT-expression* ::= *logical-NOT-with-method-invocation-without-parentheses*
25 | *logical-NOT-with-unary-expression*

27 *logical-NOT-with-method-invocation-without-parentheses* ::
28 ! *method-invocation-without-parentheses*

29 *logical-NOT-with-unary-expression* ::
30 ! *unary-expression*

1 *keyword-AND-expression* ::
 2 *expression* **and** *keyword-NOT-expression*

3 *keyword-OR-expression* ::
 4 *expression* **or** *keyword-NOT-expression*

5 *logical-OR-expression* ::
 6 *logical-AND-expression*
 7 | *logical-OR-expression* || *logical-AND-expression*

8 *logical-AND-expression* ::
 9 *equality-expression*
 10 | *logical-AND-expression* **&&** *equality-expression*

11 Semantics

12 A *keyword-logical-expression* is evaluated as follows:

- 13 a) A *logical-NOT-expression* or a *keyword-NOT-expression* of the form **not** *keyword-NOT-*
 14 *expression* is evaluated as follows:
- 15 1) If it is of the form **not** *keyword-NOT-expression*, evaluate the *keyword-NOT-expression*.
 16 Let X be the resulting value.
 - 17 2) If it is a *logical-NOT-expression*, evaluate its *method-invocation-without-parentheses* or
 18 *unary-expression*. Let X be the resulting value.
 - 19 3) If X is a trueish value, the value of the *keyword-NOT-expression* or the *logical-NOT-*
 20 *expression* is **false**.
 - 21 4) Otherwise, the value of the *keyword-NOT-expression* or the *logical-NOT-expression* is
 22 **true**.

23 Instead of the above process, a conforming processor may evaluate a *logical-NOT-expression*
 24 as follows:

- 25 1) Evaluate the *unary-expression* or the *method-invocation-without-parentheses*. Let V be
 26 the resulting value.
- 27 2) Create an empty list of arguments L . Invoke the method !@ on V with L as the list of
 28 arguments. The resulting value is the value of the *logical-NOT-expression*.

29 In this case, the processor shall:

- 30 • include the operator !@ in *operator-method-name*.
- 31 • define an instance method !@ in the class `Object`, one of its superclasses, or a module
 32 included in the class `Object`. The method !@ shall not take any arguments. The method
 33 !@ shall return **true** if the receiver is **false** or **nil**, and shall return **false** otherwise.

1 b) A *logical-AND-expression* of the form *logical-AND-expression* **&&** *equality-expression* or a
2 *keyword-AND-expression* is evaluated as follows:

- 3 1) Evaluate the *expression* or the *logical-AND-expression*. Let *X* be the resulting value.
- 4 2) If *X* is a trueish value, evaluate the *keyword-NOT-expression* or *equality-expression*.
5 Let *Y* be the resulting value. The value of the *keyword-AND-expression* or the *logical-*
6 *AND-expression* is *Y*.
- 7 3) Otherwise, the value of the *keyword-AND-expression* or the *logical-AND-expression* is
8 *X*.

9 c) A *keyword-OR-expression* or a *logical-OR-expression* of the form *logical-OR-expression* **||**
10 *logical-AND-expression* is evaluated as follows:

- 11 1) Evaluate the *expression* or the *logical-OR-expression*. Let *X* be the resulting value.
- 12 2) If *X* is a falseish value, evaluate the *keyword-NOT-expression* or the *logical-AND-*
13 *expression*. Let *Y* be the resulting value. The value of the *keyword-OR-expression* or
14 *logical-OR-expression* is *Y*.
- 15 3) Otherwise, the value of the *keyword-OR-expression* or *logical-OR-expression* is *X*.

16 11.3 Method invocation expressions

17 11.3.1 General description

18 Syntax

19 *method-invocation-expression* ::=
20 *primary-method-invocation*
21 | *method-invocation-without-parentheses*

22 *primary-method-invocation* ::
23 *super-with-optional-argument*
24 | *indexing-method-invocation*
25 | *method-only-identifier*
26 | *method-identifier* ([no whitespace here] *argument-with-parentheses*)? *block*?
27 | *primary-expression* [no line-terminator here]
28 . *method-name* ([no whitespace here] *argument-with-parentheses*)? *block*?
29 | *primary-expression* [no line-terminator here]
30 :: *method-name* [no whitespace here] *argument-with-parentheses* *block*?
31 | *primary-expression* [no line-terminator here] :: *method-name-without-constant*
32 *block*?

33 *indexing-method-invocation* ::
34 *primary-expression* [no line-terminator here] *optional-whitespace*?
35 [*indexing-argument-list*?]


```

1  optional-whitespace ::
2      [ whitespace here ]

3  method-name-without-constant ::
4      method-name but not constant-identifier

5  method-invocation-without-parentheses ::
6      command
7      | chained-command-with-do-block
8      | chained-command-with-do-block ( . | :: ) method-name argument
9      | return-with-argument
10     | break-with-argument
11     | next-with-argument

12  command ::
13     super-with-argument
14     | yield-with-argument
15     | method-identifier argument
16     | primary-expression [no line-terminator here] ( . | :: ) method-name argument

17  chained-command-with-do-block ::
18     command-with-do-block chained-method-invocation*

19  chained-method-invocation ::
20     ( . | :: ) method-name
21     | ( . | :: ) method-name [no whitespace here]
22     [lookahead ∉ { { } } argument-with-parentheses

23  command-with-do-block ::
24     super-with-argument-and-do-block
25     | method-identifier argument do-block
26     | primary-expression [no line-terminator here] ( . | :: ) method-name argument
27     do-block

```

28 The *primary-expression* of a *primary-method-invocation*, *command*, and *indexing-method-invocation*
29 shall not be a *jump-expression*.

30 If the *argument-with-parentheses* of a *primary-method-invocation* is present, and the *block-*
31 *argument* of the *argument* of the *argument-with-parentheses* is present, the *block* of the *primary-*
32 *method-invocation* shall be omitted.

33 If the *argument* of a *command-with-do-block* is present, and the *block-argument* of the *argument-*
34 *in-parentheses* of the *argument* (see §11.3.2) is present, the *do-block* of the *command-with-do-*
35 *block* shall be omitted.

36 The *optional-whitespace* of an *indexing-method-invocation* shall be omitted if its *primary-expression*
37 is any of the following constructs:

- 1 • A *primary-method-invocation* of the form *method-only-identifier*, *method-identifier*, *primary-expression* . *method-name*, or *primary-expression* :: *method-name-without-constant*
- 2
- 3 • A *method-invocation-without-parentheses* of the form *chained-command-with-do-block* which
- 4 satisfies all of the following conditions:
- 5 — Let *M* be the *chained-command-with-do-block*. One or more *chained-method-invocation*
- 6 of *M* is present.
- 7 — Let *I* be the last *chained-method-invocation* of *M*, in the order they appear in program
- 8 text. *I* is of the form (*.*|*:*) *method-name*.

9 Semantics

10 A *method-invocation-expression* is evaluated as follows:

11 a) A *primary-method-invocation* is evaluated as follows:

12 1) If the *primary-method-invocation* is a *super-with-optional-argument* or an *indexing-*

13 *method-invocation*, evaluate it. The resulting value is the value of the *primary-method-*

14 *invocation*.

15 2) i) If the *primary-method-invocation* is a *method-only-identifier*, let *O* be the current

16 self and let *M* be the *method-only-identifier*. Create an empty list of arguments

17 *L*.

18 ii) If the *method-identifier* of the *primary-method-invocation* is present:

19 I) Let *O* be the current self and let *M* be the *method-identifier*.

20 II) If the *argument-with-parentheses* is present, construct a list of arguments and

21 a block from the *argument-with-parentheses* as described in §11.3.2. Let *L* be

22 the resulting list. Let *B* be the resulting block, if any.

23 If the *argument-with-parentheses* is omitted, create an empty list of arguments

24 *L*.

25 III) If the *block* is present, let *B* be the *block*.

26 iii) If the *.* of the *primary-method-invocation* is present:

27 I) Evaluate the *primary-expression* and let *O* be the resulting value. Let *M* be

28 the *method-name*.

29 II) If the *argument-with-parentheses* is present, construct a list of arguments and

30 a block from the *argument-with-parentheses* as described in §11.3.2. Let *L* be

31 the resulting list. Let *B* be the resulting block, if any.

32 If the *argument-with-parentheses* is omitted, create an empty list of arguments

33 *L*.

34 III) If the *block* is present, let *B* be the *block*.

- 1 iv) If the `::` and *method-name* of the *primary-method-invocation* are present:
- 2 I) Evaluate the *primary-expression* and let *O* be the resulting value. Let *M* be
3 the *method-name*.
- 4 II) Construct a list of arguments and a block from the *argument-with-parentheses*
5 as described in §11.3.2. Let *L* be the resulting list. Let *B* be the resulting
6 block, if any.
- 7 III) If the *block* is present, let *B* be the *block*.
- 8 v) If the `::` and *method-name-without-constant* of the *primary-method-invocation* are
9 present:
- 10 I) Evaluate the *primary-expression* and let *O* be the resulting value. Let *M* be
11 the *method-name-without-constant*.
- 12 II) Create an empty list of arguments *L*.
- 13 III) If the *block* is present, let *B* be the *block*.
- 14 3) Invoke the method *M* on *O* with *L* as the list of arguments and *B*, if any, as the block.
15 (see §13.3.3). The resulting value is the value of the *primary-method-invocation*.
- 16 b) An *indexing-method-invocation* is evaluated as follows:
- 17 1) Evaluate the *primary-expression*. Let *O* be the resulting value.
- 18 2) If the *indexing-argument-list* is present, construct a list of arguments from the *indexing-*
19 *argument-list* as described in §11.3.2. Let *L* be the resulting list.
- 20 3) If the *indexing-argument-list* is omitted, Create an empty list of arguments *L*.
- 21 4) Invoke the method `[]` on *O* with *L* as the list of arguments. The resulting value is the
22 value of the *indexing-method-invocation*.
- 23 c) A *method-invocation-without-parentheses* is evaluated as follows:
- 24 1) If the *method-invocation-without-parentheses* is a *command*, evaluate it. The resulting
25 value is the value of the *method-invocation-without-parentheses*.
- 26 2) If the *method-invocation-without-parentheses* is a *return-with-argument*, *break-with-*
27 *argument* or *next-with-argument*, evaluate it (see §11.5.2.3).
- 28 3) If the *chained-command-with-do-block* of the *method-invocation-without-parentheses* is
29 present:
- 30 i) Evaluate the *chained-command-with-do-block*. Let *V* be the resulting value.
- 31 ii) If the *method-name* and the *argument* of the *method-invocation-without-parentheses*
32 are present:

- 1 I) Let M be the *method-name*.
- 2 II) Construct a list of arguments from the *argument* as described in §11.3.2 and let
3 L be the resulting list. If the *block-argument* of the *argument-in-parentheses*
4 of the *argument* is present, let B be the *block* to which the *block-argument*
5 corresponds.
- 6 III) Invoke the method M on V with L as the list of arguments and B , if any, as
7 the block.
- 8 IV) Replace V with the resulting value.
- 9 iii) The value of the *method-invocation-without-parentheses* is V .
- 10 d) A *command* is evaluated as follows:
- 11 1) If the *command* is a *super-with-argument* or a *yield-with-argument*, evaluate it.
- 12 2) Otherwise:
- 13 i) If the *method-identifier* of the *command* is present:
- 14 I) Let O be the current self and let M be the *method-identifier*.
- 15 II) Construct a list of arguments from the *argument* as described in §11.3.2 and
16 let L be the resulting list.
- 17 If the *block-argument* of the *argument-in-parentheses* of the *argument* is present,
18 let B be the *block* to which the *block-argument* corresponds.
- 19 ii) If the *primary-expression*, *method-name*, and the *argument* of the *command* is
20 present:
- 21 I) Evaluate the *primary-expression*. Let O be the resulting value. Let M be the
22 *method-name*.
- 23 II) Construct a list of arguments from the *argument* as described in §11.3.2 and
24 let L be the resulting list.
- 25 If the *block-argument* of the *argument-in-parentheses* of the *argument* is present,
26 let B be the *block* to which the *block-argument* corresponds.
- 27 iii) Invoke the method M on O with L as the list of arguments and B , if any, as the
28 block. The resulting value is the value of the *command*.
- 29 e) A *chained-command-with-do-block* is evaluated as follows:
- 30 1) Evaluate the *command-with-do-block* and let V be the resulting value.
- 31 2) For each *chained-method-invocation*, in the order they appears in the program text,
32 take the following steps:

- 1 i) Let M be the *method-name* of the *chained-method-invocation*.
- 2 ii) If the *argument-with-parentheses* is present, construct a list of arguments and a
3 block from the *argument-with-parentheses* as described in §11.3.2 and let L be the
4 resulting list. Let B be the resulting block, if any.
- 5 If the *argument-with-parentheses* is omitted, create an empty list of arguments L .
- 6 iii) Invoke the method M on V with L as the list of arguments and B , if any, as the
7 block.
- 8 iv) Replace V with the resulting value.
- 9 3) The value of the *chained-command-with-do-block* is V .
- 10 f) A *command-with-do-block* is evaluated as follows:
- 11 1) If the *command-with-do-block* is a *super-with-argument-and-do-block*, evaluate it. The
12 resulting value is the value of the *command-with-do-block*.
- 13 2) Otherwise:
- 14 i) If the *method-identifier* of the *command* is present, let O be the current self and
15 let M be the *method-name*.
- 16 If the *method-identifier* of the *command* is omitted, evaluate the *primary-expression*,
17 let O be the resulting value and let M be the *method-name*.
- 18 ii) Construct a list of arguments from the *argument* of the *command-with-do-block*
19 and let L be the resulting list.
- 20 iii) Invoke the method M on O with L as the list of arguments and the *do-block* as
21 the block. The resulting value is the value of the *command-with-do-block*.

22 11.3.2 Method arguments

23 Syntax

24 *method-argument* ::=

25 *indexing-argument-list*

26 | *argument-with-parentheses*

27 | *argument*

28 *indexing-argument-list* ::=

29 *command*

30 | *operator-expression-list* ,?

31 | *operator-expression-list* , *splatting-argument*

32 | *association-list* ,?

33 | *splatting-argument*

```

1  splattling-argument ::
2      * operator-expression

3  operator-expression-list ::
4      operator-expression ( , operator-expression )*

5  argument-with-parentheses ::
6      ( )
7      | ( argument-in-parentheses )
8      | ( operator-expression-list , chained-command-with-do-block )
9      | ( chained-command-with-do-block )

10 argument ::
11     [no line-terminator here] [lookahead ∉ { { }] optional-whitespace?
12     argument-in-parentheses

13 argument-in-parentheses ::
14     command
15     | ( operator-expression-list | association-list )
16         ( , splattling-argument )? ( , block-argument )?
17     | operator-expression-list , association-list
18         ( , splattling-argument )? ( , block-argument )?
19     | splattling-argument ( , block-argument )?
20     | block-argument

21 block-argument ::
22     & operator-expression

```

23 The *operator-expression* of a *splattling-argument*, *operator-expression-list*, and *block-argument*
24 shall not be a *jump-expression*.

25 If the *operator-expression-list* of an *argument-in-parentheses* is present, the first *operator-expression*
26 of the *operator-expression-list* is called the **first argument**.

27 If an *argument-in-parentheses* is of the form *splattling-argument* (, *block-argument*)?, then the
28 *splattling-argument* is called the **first argument**, and *whitespaces* shall be omitted between its
29 * and *operator-expression*.

30 If an *argument-in-parentheses* is of the form *block-argument*, then the *splattling-argument* is called
31 the **first argument**, and *whitespaces* shall be omitted between its & and *operator-expression*.

32 If the first argument of an *argument* is other than the following constructs, the *optional-*
33 *whitespace* shall be present.

- 34 • A *variable-reference* of the form *global-variable-identifier*, *class-variable-identifier* or *instance-*
35 *variable-identifier* (see §11.5.4).
- 36 • A *single-quoted-string* or *double-quoted-string* (see §8.7.6.3).

- 1 • A *symbol-literal*, or a *dynamic-symbol* of the form : [no *whitespace* here] *single-quoted-string*
2 or : [no *whitespace* here] *double-quoted-string* (see §8.7.6.6).
- 3 • An *external-command-execution* of the form *backquoted-external-command-execution* (see
4 §8.7.6.3.7).
- 5 • A *scoped-constant-reference* whose *primary-expression* is present and the *primary-expression*
6 is any of these constructs.
- 7 • A *primary-method-invocation* whose *primary-expression* is present and the *primary-expression*
8 is any of these constructs.

9 Semantics

10 The list of arguments used for method invocation is constructed from a *method-argument* as
11 follows:

- 12 a) An *indexing-argument-list* is processed as follows:
 - 13 1) Create an empty list of arguments *L*.
 - 14 2) Evaluate the *command*, *operator-expressions* of *operator-expression-lists*, and the *association-*
15 *list* and append their values to *L* in the order they appear in the program text.
 - 16 3) If the *splatting-argument* is present, construct a list of arguments from it and concate-
17 nate the resulting list to *L*.
- 18 b) A *splatting-argument* is processed as follows:
 - 19 1) Create an empty list of arguments *L*.
 - 20 2) Evaluate the *operator-expression*. Let *V* be the resulting value.
 - 21 3) If *V* is not an instance of the class **Array**, the behavior is unspecified.
 - 22 4) Append each element of *V*, in the indexing order, to *L*.
- 23 c) An *argument-with-parentheses* is processed as follows:
 - 24 1) Create an empty list of arguments *L*.
 - 25 2) If the *argument-in-parentheses* is present, construct a list of arguments from it and
26 concatenate the resulting list to *L*. If *block-argument* of *argument-in-parentheses* is
27 present, the *block* to which the *block-argument* corresponds is the block which is passed
28 to the method invocation with *L*.
 - 29 3) If the *operator-expression-list* is present, for each *operator-expression* of the *operator-*
30 *expression-list*, in the order they appears in the program text, take the following steps:
 - 31 i) Evaluate the *operator-expression*. Let *V* be the resulting value.
 - 32 ii) Append *V* to *L*.

- 1 4) If the *chained-command-with-do-block* is present, evaluate it. Append the resulting
2 value to *L*.
- 3 d) An *argument* is processed as follows:
- 4 1) Evaluate the *argument-in-parentheses*.
- 5 2) Let *L* be the resulting list.
- 6 e) An *argument-in-parentheses* is processed as follows:
- 7 1) Create an empty list of arguments *L*.
- 8 2) If the *command* is present, evaluate it. Append the resulting value to *L*.
- 9 3) If the *operator-expression-list* is present, for each *operator-expression* of the *operator-*
10 *expression-list*, in the order they appears in the program text, take the following steps:
- 11 i) Evaluate the *operator-expression*. Let *V* be the resulting value.
- 12 ii) Append *V* to *L*.
- 13 4) If the *association-list* is present, evaluate it. Append the resulting value to *L*.
- 14 5) If the *splating-argument* is present, construct a list of arguments from it and concate-
15 nate the resulting list to *L*.
- 16 6) If the *block-argument* is present, construct a *block* which is passed to a method invo-
17 cation as described below.
- 18 f) A block which is passed to a method invocation is constructed from the *block-argument* as
19 follows:
- 20 1) Evaluate the *operator-expression*. Let *P* be the resulting value.
- 21 2) If *P* is not an instance of the class `Proc`, the behavior is unspecified.
- 22 3) Otherwise, the resulting block is the block which *P* represents.

23 11.3.3 Blocks

24 Syntax

25 *block* ::
26 *brace-block*
27 | *do-block*

28 *brace-block* ::
29 { *block-parameter?* *block-body* }


```

1  do-block ::
2      do block-parameter? block-body end

3  block-parameter ::
4      | |
5      | ||
6      | | block-parameter-list |

7  block-parameter-list ::
8      left-hand-side
9      | multiple-left-hand-side

10 block-body ::
11     compound-statement

```

12 Whether the *left-hand-side* (see §11.4.2.4) in the *block-parameter-list* is allowed to be of the
13 following forms is implementation-defined.

- 14 • *constant-identifier*
- 15 • *global-variable-identifier*
- 16 • *instance-variable-identifier*
- 17 • *class-variable-identifier*
- 18 • *primary-expression* [*indexing-argument-list?*]
- 19 • *primary-expression* (. | ::) (*local-variable-identifier* | *constant-identifier*)
- 20 • :: *constant-identifier*

21 Whether the *grouped-left-hand-side* in the *block-parameter-list* is allowed to be the following
22 form is implementation-defined.

- 23 • ((*multiple-left-hand-side-item* ,)+);

24 Semantics

25 A *block* is a procedure which is passed to a method invocation.

26 A *block* can be called either by a *yield-expression* (see §11.3.5) or by invoking the method `call`
27 on an instance of the class `Proc` which is created by an invocation of the method `Proc.new` to
28 which the block is passed (see §15.2.17.4.3).

29 A *block* can be called with arguments. If a *block* is called by a *yield-expression*, the arguments
30 to the *yield-expression* are used as the arguments to the *block* call. If a *block* is called by an
31 invocation of the method `call`, the arguments to the method invocation is used as the arguments
32 to the *block* call.

1 A *block* is evaluated under the execution context as it exists just before the method invocation to
2 which the *block* is passed. However, the changes of variable bindings in `[[local-variable-bindings]]`
3 after the *block* is passed to the method invocation affect the execution context. Let E_b be the
4 affected execution context.

5 Both the *do-block* and the *brace-block* of the *block* are evaluated as follows:

6 a) Let E_o be the current execution context. Let L be the list of arguments passed to the block.

7 b) Set the execution context to E_b .

8 c) Push an empty set of local variable bindings onto `[[local-variable-bindings]]`.

9 d) If the *block-parameter-list* in the *do-block* or the *brace-block* is present:

10 1) If the *block-parameter-list* is of the form *left-hand-side* or *grouped-left-hand-side*:

11 i) If the length of L is 0, let X be **nil**.

12 ii) If the length of L is 1, let X be the only element of L .

13 iii) If the length of L is larger than 1, the result of this step is unspecified.

14 iv) If the *block-parameter-list* is of the form *left-hand-side*, evaluate a *single-variable-*
15 *assignment-expression* E (see §11.4.2.2.2), where the *variable* of E is the *left-hand-*
16 *side* and the value of the *operator-expression* of E is X .

17 v) If the *block-parameter-list* is of the form *grouped-left-hand-side*, evaluate a *many-*
18 *to-many-assignment-statement* E (see §11.4.2.4), where the *multiple-left-hand-side*
19 of E is the *grouped-left-hand-side* and the value of the *method-invocation-without-*
20 *parentheses* or *operator-expression* of E is X .

21 2) If the *block-parameter-list* is of the form *multiple-left-hand-side* and the *multiple-left-*
22 *hand-side* is not a *grouped-left-hand-side*:

23 i) If the length of L is 1:

24 I) If the only element of L is not an instance of the class **Array**, the result of
25 this step is unspecified.

26 II) Create a list of arguments Y which contains the elements of L , preserving
27 their order.

28 ii) If the length of L is 0 or larger than 1, let Y be L .

29 iii) Evaluate the *many-to-many-assignment-statement* E as described in §11.4.2.4,
30 where the *multiple-left-hand-side* of E is the *block-parameter-list* and the list of
31 arguments constructed from the *multiple-right-hand-side* of E is Y .

32 e) Evaluate the *block-body*. If the evaluation of the *block-body*:

33 1) is terminated by a *break-expression*:

- 1 i) If the method invocation with which *block* is passed has already terminated when
2 the *block* is called:
- 3 I) Let *S* be an instance of the class `Symbol` with name `break`.
- 4 II) If the *jump-argument* of the *break-expression* is present, let *V* be the value of
5 the *jump-argument*. Otherwise, let *V* be `nil`.
- 6 III) Raise a direct instance of the class `LocalJumpError` which has two instance
7 variable bindings, one named `@reason` with the value *S* and the other named
8 `@exit_value` with the value *V*.
- 9 ii) Otherwise, restore the execution context to *E_o* and terminate Step i and take Step
10 j of the current method invocation (see §13.3.3).
- 11 If the *jump-argument* of the *break-expression* is present, the value of the current
12 method invocation is the value of the *jump-argument*. Otherwise, the value of the
13 current method invocation is `nil`.
- 14 2) is terminated by a *redo-expression*, repeat Step e.
- 15 3) is terminated by a *next-expression*:
- 16 i) If the *jump-argument* of the *next-expression* is present, let *V* be the value of the
17 *jump-argument*.
- 18 ii) Otherwise, let *V* be `nil`.
- 19 4) is terminated by a *return-expression*, remove the element from the top of `[[local-variable-`
20 bindings]].
- 21 5) terminates otherwise, let *V* be the resulting value of the evaluation of the *block-body*.
- 22 f) Unless Step e is terminated by a *return-expression*, restore the execution context to *E_o*,
23 even when an exception is raised and not handled in Step d or e.
- 24 g) The value of calling the *do-block* or the *brace-block* is *V*.

25 11.3.4 The super expression

26 Syntax

27 *super-expression* ::=

28 *super-with-optional-argument*

29 | *super-with-argument*

30 | *super-with-argument-and-do-block*

31 *super-with-optional-argument* ::

32 **super** ([no whitespace here] *argument-with-parentheses*)? *block*?

```

1  super-with-argument ::
2      super argument

3  super-with-argument-and-do-block ::
4      super argument do-block

```

5 The *block-argument* of the *argument-in-parentheses* of the *argument* (see §11.3.2) of a *super-*
6 *with-argument-and-do-block* shall be omitted.

7 Semantics

8 A *super-expression* is evaluated as follows:

- 9 a) If the current self is pushed by an *eigenclass-definition* (see §13.4.2), or an invocation of
10 one of the following methods, the behavior is unspecified:
- 11 • the method `class_eval` of the class `Module` (see §15.2.2.4.15)
 - 12 • the method `module_eval` of the class `Module` (see §15.2.2.4.35)
 - 13 • the method `instance_eval` of the class `Kernel` (see §15.3.1.3.18)
- 14 b) Let A be an empty list. Let B be the top of `[[block]]`.
- 15 1) If the *super-expression* is a *super-with-optional-argument*, and neither the *argument-*
16 *with-parentheses* nor the *block* is present, construct a list of arguments as follows:
- 17 i) Let M be the method which correspond to the current method invocation. Let L
18 be the *parameter-list* of the *method-parameter-part* of M . Let S be the set of local
19 variable bindings in `[[local-variable-bindings]]` which corresponds to the current
20 method invocation.
 - 21 ii) If the *mandatory-parameter-list* is present in L , for each *mandatory-parameter* p ,
22 take the following steps:
 - 23 I) Let v be the value of the binding with name p in S .
 - 24 II) Append v to A .
 - 25 iii) If the *optional-parameter-list* is present in L , for each *optional-parameter* p , take
26 the following steps:
 - 27 I) Let n be the *optional-parameter-name* of p .
 - 28 II) Let v be the value of the binding with name n in S .
 - 29 III) Append v to A .
 - 30 iv) If the *array-parameter* is present in L :

- 1 I) Let n be the *array-parameter-name* of the *array-parameter*.
- 2 II) Let v be the value of the binding with name n in S . Append each element of
3 v , in the indexing order, to A .
- 4 2) If the *super-expression* is a *super-with-optional-argument* with either or both of the
5 *argument-with-parentheses* and the *block*:
- 6 i) If the *argument-with-parentheses* is present, construct a list of arguments and a
7 block as described in §11.3.2. Let A be the resulting list. Let B be the resulting
8 block, if any.
- 9 ii) If the *block* is present, Let B be the *block*.
- 10 3) If the *super-expression* is a *super-with-argument*, construct the list of arguments from
11 the *argument* as described in §11.3.2. Let A be the resulting list. If *block-argument*
12 of the *argument-in-parentheses* of *argument* is present, let B be the *block* constructed
13 from the *block-argument*.
- 14 4) If the *super-expression* is a *super-with-argument-and-do-block*, construct a list of argu-
15 ments from the *argument* as described in §11.3.2. Let A be the resulting list. Let B
16 be the *do-block*.
- 17 c) Determine the method to be invoked as follows:
- 18 1) Let C be the current class or module. Let N be the top of `[[defined-method-name]]`.
- 19 2) Search for a method binding with name N from Step b in §13.3.4, assuming that C in
20 §13.3.4 to be C .
- 21 3) If a binding is found and its value is not `undef` (see §13.1.1), let V be the value of the
22 binding.
- 23 4) Otherwise, add a direct instance of the class `Symbol` with name N to the head of A ,
24 and invoke the method `method_missing` on the current self with A as arguments and
25 B as the block. Then, terminate the evaluation of the *super-expression*. The value of
26 the *super-expression* is the resulting value of the method invocation.
- 27 d) Take Step g, h, i, and j of §13.3.3, assuming that A , B , M , R , and V in §13.3.3 to be A , B ,
28 N , the current self, and V in this subclause respectively. The value of the *super-expression*
29 is the resulting value.

30 11.3.5 The yield expression

31 Syntax

32 *yield-expression* ::=

33 *yield-with-optional-argument*

34 | *yield-with-argument*

35 *yield-with-optional-argument* ::

36 *yield-with-parentheses-and-argument*

```

1      | yield-with-parentheses-without-argument
2      | yield

3      yield-with-parentheses-and-argument ::
4      yield [no whitespace here] ( argument-in-parentheses )

5      yield-with-parentheses-without-argument ::
6      yield [no whitespace here] ( )

7      yield-with-argument ::
8      yield argument

```

9 The *block-argument* of the *argument-in-parentheses* (see §11.3.2) of a *yield-with-parentheses-and-*
10 *argument* shall be omitted.

11 The *block-argument* of the *argument-in-parentheses* of the *argument* (see §11.3.2) of a *yield-with-*
12 *argument* shall be omitted.

13 Semantics

14 A *yield-expression* calls the block at the top of `[[block]]` as follows:

- 15 a) A *yield-with-optional-argument* is evaluated as follows:
- 16 1) Let *B* be the top of `[[block]]`. If *B* is block-not-given:
 - 17 i) Let *S* be a direct instance of the class `Symbol` with name `noreason`.
 - 18 ii) Let *V* be an implementation-defined value.
 - 19 iii) Raise a direct instance of the class `LocalJumpError` which has two instance vari-
20 able bindings, one named `@reason` with the value *S* and the other named `@exit_value`
21 with the value *V*.
 - 22 2) If the *yield-with-optional-argument* is of the form *yield-with-parentheses-and-argument*,
23 create a list of arguments from the *argument* as described in §11.3.2. Let *L* be the list.
 - 24 3) If the *yield-with-optional-argument* is of the form *yield-with-parentheses-without-argument*
25 or `yield`, create an empty list of argument *L*.
 - 26 4) Call *B* with *L* as described in §11.3.3.
 - 27 5) The value of *yield-with-optional-argument* is the value of the block call.
- 28 b) A *yield-with-argument* is evaluated as follows:
- 29 1) Let *B* be the top of `[[block]]`. If *B* is block-not-given:
 - 30 i) Let *S* be a direct instance of the class `Symbol` with name `noreason`.

- 1 ii) Let V be an implementation-defined value.
- 2 iii) Raise a direct instance of the class `LocalJumpError` which has two instance vari-
3 able bindings, one named `@reason` with the value S and the other named `@exit_value`
4 with the value V .
- 5 2) Create a list of arguments from the *argument* as described in §11.3.2. Let L be the list.
- 6 3) Call B with L as described in §11.3.3.
- 7 4) The value of the *yield-with-argument* is the value of the block call.

8 11.4 Operator expressions

9 11.4.1 General description

10 Syntax

11 *operator-expression* ::
12 *assignment-expression*
13 | *defined?-without-parentheses*
14 | *conditional-operator-expression*

15 11.4.2 Assignments

16 11.4.2.1 General description

17 Syntax

18 *assignment* ::=
19 *assignment-expression*
20 | *assignment-statement*

21 *assignment-expression* ::
22 *single-assignment-expression*
23 | *abbreviated-assignment-expression*
24 | *assignment-with-rescue-modifier*

25 *assignment-statement* ::
26 *single-assignment-statement*
27 | *abbreviated-assignment-statement*
28 | *multiple-assignment-statement*

29 Semantics

30 Assignments create or update variable bindings, or invoke a method whose name ends with `=`.

1 Evaluation of each construct is described below.

2 11.4.2.2 Single assignments

3 11.4.2.2.1 General description

4 Syntax

5 *single-assignment* ::=
6 *single-assignment-expression*
7 | *single-assignment-statement*

8 *single-assignment-expression* ::
9 *single-variable-assignment-expression*
10 | *scoped-constant-assignment-expression*
11 | *single-indexing-assignment-expression*
12 | *single-method-assignment-expression*

13 *single-assignment-statement* ::
14 *single-variable-assignment-statement*
15 | *scoped-constant-assignment-statement*
16 | *single-indexing-assignment-statement*
17 | *single-method-assignment-statement*

18 11.4.2.2.2 Single variable assignments

19 Syntax

20 *single-variable-assignment* ::=
21 *single-variable-assignment-expression*
22 | *single-variable-assignment-statement*

23 *single-variable-assignment-expression* ::
24 *variable* [no line-terminator here] = *operator-expression*

25 *single-variable-assignment-statement* ::
26 *variable* [no line-terminator here] = *method-invocation-without-parentheses*

27 Semantics

28 A *single-variable-assignment* is evaluated as follows:

- 29 a) Evaluate the *operator-expression* or the *method-invocation-without-parentheses*. Let V be
30 the resulting value.
- 31 b) 1) If the *variable* is a *constant-identifier*:

- 1 i) Let N be the *constant-identifier*.
- 2 ii) If a binding with name N exists in the set of bindings of constants of the current
3 class or module, replace the value of the binding with V .
- 4 iii) Otherwise, create a variable binding with name N and value V in the set of
5 bindings of constants of the current class or module.
- 6 2) If the *variable* is a *global-variable-identifier*:
- 7 i) Let N be the *global-variable-identifier*.
- 8 ii) If a binding with name N exists in \llbracket global-variable-bindings \rrbracket , replace the value of
9 the binding with V .
- 10 iii) Otherwise, create a variable binding with name N and value V in \llbracket global-variable-
11 bindings \rrbracket .
- 12 3) If the *variable* is a *class-variable-identifier*:
- 13 i) Let C be the first class or module in the list at the top of \llbracket class-module-list \rrbracket which
14 is not an eigenclass.
- 15 Let CS be the set of classes which consists of C and all the superclasses of C . Let
16 MS be the set of modules which consists of all the modules in the included module
17 lists of all classes in CS . Let CM be the union of CS and MS .
- 18 Let N be the *class-variable-identifier*.
- 19 ii) If one of the classes or modules in CM has a binding with name N in the set of
20 bindings of class variables, let B be that binding.
- 21 If more than one class or module in CM has bindings with name N in the set
22 of bindings of class variables, let B be one of those bindings. Which binding is
23 selected is implementation-defined.
- 24 Replace the value of B with V .
- 25 iii) If none of the classes or modules in CM has a binding with name N in the set of
26 bindings of class variables, create a variable binding with name N and value V in
27 the set of bindings of class variables of C .
- 28 4) If the *variable* is an *instance-variable-identifier*:
- 29 i) Let N be the *instance-variable-identifier*.
- 30 ii) If a binding with name N exists in the set of bindings of instance variables of the
31 current self, replace the value of the binding with V .
- 32 iii) Otherwise, create a variable binding with name N and value V in the set of
33 bindings of instance variables of the current self.

- 1 5) If the *variable* is a *local-variable-identifier*:
- 2 i) Let N be the *local-variable-identifier*.
- 3 ii) Search for a binding of a local variable with name N as described in §9.2.
- 4 iii) If a binding is found, replace the value of the binding with V .
- 5 iv) Otherwise, create a variable binding with name N and value V in the current set
- 6 of local variable bindings.
- 7 c) The value of the *single-variable-assignment* is V .

8 11.4.2.2.3 Scoped constant assignments

9 Syntax

10 *scoped-constant-assignment* ::=

11 *scoped-constant-assignment-expression*

12 | *scoped-constant-assignment-statement*

13 *scoped-constant-assignment-expression* ::

14 *primary-expression* [no *whitespace* here] :: *constant-identifier*

15 [no *line-terminator* here] = *operator-expression*

16 | :: *constant-identifier* [no *line-terminator* here] = *operator-expression*

17 *scoped-constant-assignment-statement* ::

18 *primary-expression* [no *whitespace* here] :: *constant-identifier*

19 [no *line-terminator* here] = *method-invocation-without-parentheses*

20 | :: *constant-identifier* [no *line-terminator* here] = *method-invocation-without-parentheses*

21 Semantics

22 A *scoped-constant-assignment* is evaluated as follows:

- 23 a) If the *primary-expression* is present, evaluate it and let M be the resulting value. Otherwise,
- 24 let M be the class `Object`.
- 25 b) If M is an instance of the class `Module`:
- 26 1) Let N be the *constant-identifier*.
- 27 2) Evaluate the *operator-expression* or the *method-invocation-without-parentheses*. Let V
- 28 be the resulting value.
- 29 3) Create a variable binding with name N and value V in the set of bindings of constants
- 30 of M .
- 31 4) The value of the *scoped-constant-assignment* is V .

1 c) If M is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.

2 11.4.2.2.4 Single indexing assignments

3 Syntax

4 *single-indexing-assignment* ::=
5 *single-indexing-assignment-expression*
6 | *single-indexing-assignment-statement*

7 *single-indexing-assignment-expression* ::
8 *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
9 [no *line-terminator* here] = *operator-expression*

10 *single-indexing-assignment-statement* ::
11 *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
12 [no *line-terminator* here] = *method-invocation-without-parentheses*

13 Semantics

14 A *single-indexing-assignment* is evaluated as follows:

15 a) Evaluate the *primary-expression*. Let O be the resulting value.

16 b) Construct a list of arguments from the *indexing-argument-list* as described in §11.3.2. Let
17 L be the resulting list.

18 c) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let V be the
19 resulting value.

20 d) Append V to L .

21 e) Invoke the method `[]=` on O with L as the list of arguments.

22 f) The value of the *single-indexing-assignment* is V .

23 11.4.2.2.5 Single method assignments

24 Syntax

25 *single-method-assignment* ::=
26 *single-method-assignment-expression*
27 | *single-method-assignment-statement*

28 *single-method-assignment-expression* ::
29 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
30 [no *line-terminator* here] = *operator-expression*

1 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
2 | [no *line-terminator* here] = *operator-expression*

3 *single-method-assignment-statement* ::
4 | *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
5 | [no *line-terminator* here] = *method-invocation-without-parentheses*
6 | | *primary-expression* [no *line-terminator* here] . *constant-identifier*
7 | [no *line-terminator* here] = *method-invocation-without-parentheses*

8 Semantics

9 A *single-method-assignment* is evaluated as follows:

- 10 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 11 b) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let V be the
12 resulting value.
- 13 c) Let M be the *local-variable-identifier* or *constant-identifier*. Let N be the concatenation of
14 M and =.
- 15 d) Invoke the method whose name is N on O with a list of arguments which contains only one
16 value V .
- 17 e) The value of the *single-method-assignment* is V .

18 11.4.2.3 Abbreviated assignments

19 11.4.2.3.1 General description

20 Syntax

21 *abbreviated-assignment* ::=
22 | *abbreviated-assignment-expression*
23 | *abbreviated-assignment-statement*

24 *abbreviated-assignment-expression* ::
25 | *abbreviated-variable-assignment-expression*
26 | | *abbreviated-indexing-assignment-expression*
27 | | *abbreviated-method-assignment-expression*

28 *abbreviated-assignment-statement* ::
29 | *abbreviated-variable-assignment-statement*
30 | | *abbreviated-indexing-assignment-statement*
31 | | *abbreviated-method-assignment-statement*

1 11.4.2.3.2 Abbreviated variable assignments

2 Syntax

3 *abbreviated-variable-assignment* ::=
4 *abbreviated-variable-assignment-expression*
5 | *abbreviated-variable-assignment-statement*

6 *abbreviated-variable-assignment-expression* ::
7 *variable* [no *line-terminator* here] *assignment-operator* *operator-expression*

8 *abbreviated-variable-assignment-statement* ::
9 *variable* [no *line-terminator* here] *assignment-operator*
10 *method-invocation-without-parentheses*

11 Semantics

12 An *abbreviated-variable-assignment* is evaluated as follows:

- 13 a) Evaluate the *variable* as a variable reference (see §11.5.4). Let *V* be the resulting value.
- 14 b) Evaluate the *operator-expression* or the *method-invocation-without-parentheses*. Let *W* be
15 the resulting value.
- 16 c) Let *OP* be the *assignment-operator-name* of the *assignment-operator*.
- 17 d) Evaluate the *operator-expression* of the form *L OP R*, where the value of *L* is *V* and the
18 value of *R* is *W*. Let *X* be the resulting value.
- 19 e) Let *I* be the *variable* of the *abbreviated-variable-assignment-expression* or the *abbreviated-*
20 *variable-assignment-statement*.
- 21 f) Evaluate a *single-variable-assignment-expression* (see §11.4.2.2.2) where its *variable* is *I* and
22 the value of the *operator-expression* is *X*.
- 23 g) The value of the *abbreviated-variable-assignment* is *X*.

24 11.4.2.3.3 Abbreviated indexing assignments

25 Syntax

26 *abbreviated-indexing-assignment* ::=
27 *abbreviated-indexing-assignment-expression*
28 | *abbreviated-indexing-assignment-statement*

29 *abbreviated-indexing-assignment-expression* ::
30 *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
31 [no *line-terminator* here] *assignment-operator* *operator-expression*

1 *abbreviated-indexing-assignment-statement* ::
 2 *primary-expression* [no *line-terminator* here] [*indexing-argument-list*?]
 3 [no *line-terminator* here] *assignment-operator method-invocation-without-parentheses*

4 **Semantics**

5 An *abbreviated-indexing-assignment* is evaluated as follows:

- 6 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 7 b) Construct a list of arguments from the *indexing-argument-list* as described in §11.3.2. Let
 8 L be the resulting list.
- 9 c) Invoke the method [] on O with L as the list of arguments. Let V be the resulting value.
- 10 d) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let W be the
 11 resulting value.
- 12 e) Let OP be the *assignment-operator-name* of the *assignment-operator*.
- 13 f) Evaluate the *operator-expression* of the form $V OP W$. Let X be the resulting value.
- 14 g) Append X to L .
- 15 h) Invoke the method []= on O with L as the list of arguments.
- 16 i) The value of the *abbreviated-indexing-assignment* is X .

17 **11.4.2.3.4 Abbreviated method assignments**

18 **Syntax**

19 *abbreviated-method-assignment* ::=
 20 *abbreviated-method-assignment-expression*
 21 | *abbreviated-method-assignment-statement*

22 *abbreviated-method-assignment-expression* ::
 23 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
 24 [no *line-terminator* here] *assignment-operator operator-expression*
 25 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
 26 [no *line-terminator* here] *assignment-operator operator-expression*

27 *abbreviated-method-assignment-statement* ::
 28 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
 29 [no *line-terminator* here] *assignment-operator method-invocation-without-parentheses*
 30 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
 31 [no *line-terminator* here] *assignment-operator method-invocation-without-parentheses*

1 Semantics

2 An *abbreviated-method-assignment* is evaluated as follows:

- 3 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 4 b) Create an empty list of arguments L . Invoke the method whose name is the *local-variable-identifier* on O with L as the list of arguments. Let V be the resulting value.
- 5
- 6 c) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let W be the
- 7 resulting value.
- 8 d) Let OP be the *assignment-operator-name* of the *assignment-operator*.
- 9 e) Evaluate the *single-method-assignment* of the form $V OP W$. Let X be the resulting value.
- 10 f) Let M be the *local-variable-identifier* or the *constant-identifier*. Let N be the concatenation
- 11 of M and $=$.
- 12 g) Invoke the method whose name is N on O with X as the argument.
- 13 h) The value of the *abbreviated-method-assignment* is X .

14 11.4.2.4 Multiple assignments

15 Syntax

16 *multiple-assignment-statement* ::

17 *many-to-one-assignment-statement*
18 | *one-to-packing-assignment-statement*
19 | *many-to-many-assignment-statement*

20 *many-to-one-assignment-statement* ::

21 *left-hand-side* [no *line-terminator* here] = *multiple-right-hand-side*

22 *one-to-packing-assignment-statement* ::

23 *packing-left-hand-side* [no *line-terminator* here] =
24 (*method-invocation-without-parentheses* | *operator-expression*)

25 *many-to-many-assignment-statement* ::

26 *multiple-left-hand-side* [no *line-terminator* here] = *multiple-right-hand-side*
27 | (*multiple-left-hand-side* **but not** *packing-left-hand-side*)
28 [no *line-terminator* here] =
29 (*method-invocation-without-parentheses* | *operator-expression*)

30 *left-hand-side* ::

31 *variable*
32 | *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
33 | *primary-expression* [no *line-terminator* here]

```

1      ( . | :: ) ( local-variable-identifier | constant-identifier )
2      | :: constant-identifier

3  multiple-left-hand-side ::
4      ( multiple-left-hand-side-item , )+ multiple-left-hand-side-item?
5      | ( multiple-left-hand-side-item , )+ packing-left-hand-side?
6      | packing-left-hand-side
7      | grouped-left-hand-side

8  packing-left-hand-side ::
9      * left-hand-side?

10 grouped-left-hand-side ::
11     ( multiple-left-hand-side )

12 multiple-left-hand-side-item ::
13     left-hand-side
14     | grouped-left-hand-side

15 multiple-right-hand-side ::
16     operator-expression-list ( , splattng-right-hand-side )?
17     | splattng-right-hand-side

18 splattng-right-hand-side ::
19     splattng-argument

```

20 Any of the *operator-expressions* in a *multiple-assignment-statement* or *splattng-right-hand-side*
21 shall not be a *jump-expression*.

22 Semantics

23 A *multiple-assignment-statement* is evaluated as follows:

24 a) A *many-to-one-assignment-statement* is evaluated as follows:

- 25 1) Construct a list of values from the *multiple-right-hand-side* (see below). Let L be the
26 resulting list.
- 27 2) If the length of L is 0 or 1, let A be an implementation-defined value.
- 28 3) If the length of L is larger than 1, create a direct instance of the class **Array** and store
29 the elements of L in it, preserving their order. Let A be the instance of the class **Array**.
- 30 4) Evaluate a *single-variable-assignment-expression* (see §11.4.2.2.2) where its *variable* is
31 the *left-hand-side* and the value of its *operator-expression* is A .
- 32 5) The value of the *many-to-one-assignment-statement* is A .

- 1 b) A list of values is constructed from a *multiple-right-hand-side* as follows:
- 2 1) If the *operator-expression-list* is present, evaluate its *operator-expressions* in the order
3 they appear in the program text. Let *L1* be a list which contains the resulting values,
4 preserving their order.
 - 5 2) If the *operator-expression-list* is omitted, create an empty list of values *L1*.
 - 6 3) If the *splatting-right-hand-side* is present, construct a list of values from its *splatting-*
7 *argument* as described in §11.3.2 and let *L2* be the resulting list.
 - 8 4) If the *splatting-right-hand-side* is omitted, create an empty list of values *L2*.
 - 9 5) The result is the concatenation of *L1* and *L2*.
- 10 c) A *one-to-packing-assignment-statement* is evaluated as follows:
- 11 1) Evaluate the *method-invocation-without-parentheses* or the *operator-expression*. Let *V*
12 be the value.
 - 13 2) If *V* is an instance of the class `Array`, let *A* be an implementation-defined value.
 - 14 3) If *V* is not an instance of the class `Array`, create an instance of the class `Array` *A*
15 which contains only one value *V*.
 - 16 4) If the *left-hand-side* of the *packing-left-hand-side* is present, evaluate a *single-variable-*
17 *assignment-expression* (see §11.4.2.2.2) where its *variable* is the *left-hand-side* and the
18 value of the *operator-expression* is *A*.
 - 19 5) The value of the *one-to-packing-assignment-statement* is *A*.
- 20 d) A *many-to-many-assignment-statement* is evaluated as follows:
- 21 1) If the *multiple-right-hand-side* is present, construct a list of values from it (see above)
22 and let *R* be the resulting list.
 - 23 2) If the *multiple-right-hand-side* is omitted:
 - 24 i) Evaluate the *method-invocation-without-parentheses* or the *operator-expression*.
25 Let *V* be the resulting value.
 - 26 ii) If *V* is not an instance of the class `Array`, the behavior is unspecified.
 - 27 iii) Create a list of arguments *R* which contains all the elements of *V*, preserving their
28 order.
 - 29 3) Create an empty list of variables *L*.
- 30 For each *multiple-left-hand-side-item*, in the order they appear in the program text,
31 append the *left-hand-side* or the *grouped-left-hand-side* of the *multiple-left-hand-side-*
32 *item* to *L*.

- 1 If the *packing-left-hand-side* of the *multiple-left-hand-side* is present, append it to L .
- 2 If the *multiple-left-hand-side* is a *grouped-left-hand-side*, append the *grouped-left-hand-*
3 *side* to L .
- 4 4) For each element L_i of L , in the same order in L , take the following steps:
- 5 i) Let i be the index of L_i within L . Let N_R be the number of elements of R .
- 6 ii) If L_i is a *left-hand-side*:
- 7 I) If i is larger than N_R , let V be **nil**.
- 8 II) Otherwise, let V be the i th element of R .
- 9 III) Evaluate the *single-variable-assignment* of the form $L_i = V$.
- 10 iii) If L_i is a *packing-left-hand-side* and its *left-hand-side* is present:
- 11 I) If i is larger than N_R , create an empty direct instance of the class **Array**. Let
12 A be the instance.
- 13 II) Otherwise, create a direct instance of the class **Array** which contains elements
14 in R whose index is equal to, or larger than i , in the same order they are store
15 in R . let A be the instance.
- 16 III) Evaluate a *single-variable-assignment-expression* (see §11.4.2.2.2) where its
17 *variable* is the *left-hand-side* and the value of the *operator-expression* is A .
- 18 iv) If L_i is a *grouped-left-hand-side*:
- 19 I) If i is larger than N_R , let V be **nil**.
- 20 II) Otherwise, let V be the i th element of R .
- 21 III) Evaluate a *many-to-many-assignment-statement* where its *multiple-left-hand-*
22 *side* is the *multiple-left-hand-side* of the *grouped-left-hand-side* and its *multiple-*
23 *right-hand-side* is V .

24 11.4.2.5 Assignments with rescue modifiers

25 Syntax

26 *assignment-with-rescue-modifier* ::
27 *left-hand-side* [no *line-terminator* here] =
28 *operator-expression*₁ **rescue** *operator-expression*₂

29 Semantics

30 An *assignment-with-rescue-modifier* is evaluated as follows:

- 1 a) Evaluate the *operator-expression*₁. Let *V* be the resulting value.
- 2 If an exception is raised and not handled during the evaluation of the *operator-expression*₁,
 3 and if the exception is an instance of the class `StandardError`, evaluate the *operator-*
 4 *expression*₂ and let *V* be the resulting value.
- 5 b) Evaluate a *single-variable-assignment-expression* (see §11.4.2.2.2) where its *variable* is the
 6 *left-hand-side* and the value of the *operator-expression* is *V*. The value of the *assignment-*
 7 *with-rescue-modifier* is the resulting value of the evaluation.

8 11.4.3 Unary operators

9 11.4.3.1 General description

10 Syntax

11 *unary-operator-expression* ::=
 12 *unary-minus-expression*
 13 | *unary-expression*

14 *unary-minus-expression* ::
 15 *power-expression*₁
 16 | - *power-expression*₂

17 *unary-expression* ::
 18 *primary-expression*
 19 | *logical-NOT-with-unary-expression*
 20 | ~ *unary-expression*₁
 21 | + *unary-expression*₂

22 If a *unary-minus-expression* of the form - *power-expression*₂ is the first argument (see §11.3.2),
 23 *whitespaces* shall be omitted between its - and *power-expression*₂.

24 If a *unary-expression* of the form + *unary-expression*₂ is the first argument (see §11.3.2), *whites-*
 25 *paces* shall be omitted between its + and *unary-expression*₂.

26 Semantics

27 An *unary-operator-expression* is evaluated as follows:

- 28 a) A *logical-NOT-with-unary-expression* is evaluated as described in §11.2.
- 29 b) An *unary-expression* of the form ~ *unary-expression*₁ is evaluated as follows:
- 30 1) Evaluate the *unary-expression*₁. Let *X* be the resulting value.
- 31 2) Create an empty list of arguments *L*. Invoke the method ~ on *X* with *L* as the list of
 32 arguments. The value of the *unary-expression* is the resulting value of the invocation.

- 1 c) An *unary-expression* of the form `+ unary-expression2` is evaluated as follows:
- 2 1) Evaluate the *unary-expression₂*. Let *X* be the resulting value.
- 3 2) Create an empty list of arguments *L*. Invoke the method `+@` on *X* with *L* as the list of
4 arguments. The value of the *unary-expression* is the resulting value of the invocation.
- 5 3) If the *unary-expression₂* is a *numeric-literal* (see §8.7.6.2), instead of the above process,
6 a conforming processor may evaluate the *unary-expression* to the value of the *numeric-*
7 *literal*.
- 8 d) An *unary-expression* of the form `- power-expression2` is evaluated as follows:
- 9 1) Evaluate the *power-expression₂*. Let *X* be the resulting value.
- 10 2) Create an empty list of arguments *L*. Invoke the method `-@` on *X* with *L* as the list of
11 arguments. The value of the *unary-expression* is the resulting value of the invocation.

12 11.4.3.2 The `defined?` expression

13 Syntax

14 *defined?-expression* ::=
15 *defined?-with-parentheses*
16 | *defined?-without-parentheses*

17 *defined?-with-parentheses* ::
18 `defined? (expression)`

19 *defined?-without-parentheses* ::
20 `defined? operator-expression`

21 Semantics

22 A *defined?-expression* is evaluated as follows:

- 23 a) If the *defined?-expression* is a *defined?-with-parentheses*, let *E* be the *expression*.
- 24 If the *defined?-expression* is a *defined?-without-parentheses*, let *E* be the *operator-expression*.
- 25 b) If *E* is a *constant-identifier*:
- 26 1) Search for a binding of a constant with name *E* with the same evaluation steps for
27 *constant-identifier* as described in §11.5.4.2. However, a direct instance of the class
28 `NameError` shall not be raised when a binding is not found.
- 29 2) If a binding is found, the value of the *defined?-expression* is an implementation-defined
30 value, which shall be a trueish value.

- 1 3) Otherwise, the value of the *defined?-expression* is **nil**.
- 2 c) If *E* is a *global-variable-identifier*:
- 3 1) If a binding with name *E* exists in `[[global-variable-bindings]]`, the value of the *defined?-*
4 *expression* is an implementation-defined value, which shall be a trueish value.
- 5 2) Otherwise, the value of the *defined?-expression* is **nil**.
- 6 d) If *E* is a *class-variable-identifier*:
- 7 1) Let *C* be the current class or module. Let *CS* be the set of classes which consists of *C*
8 and all the superclasses of *C*. Let *MS* be the set of modules which consists of all the
9 modules in the included module lists of all classes in *CS*. Let *CM* be the union of *CS*
10 and *MS*.
- 11 2) If any of the classes or modules in *CM* has a binding with name *E* in the set of bindings
12 of class variables, the value of the *defined?-expression* is an implementation-defined
13 value, which shall be a trueish value.
- 14 3) Otherwise, the value of the *defined?-expression* is **nil**.
- 15 e) If *E* is an *instance-variable-identifier*:
- 16 1) If a binding with name *E* exists in the set of bindings of instance variables of the
17 current self, the value of the *defined?-expression* is an implementation-defined value,
18 which shall be a trueish value.
- 19 2) Otherwise, the value of the *defined?-expression* is **nil**.
- 20 f) If *E* is a *local-variable-identifier*:
- 21 1) If the *local-variable-identifier* is a reference (see §11.5.4.7.2), the value of the *defined?-*
22 *expression* is an implementation-defined value, which shall be a trueish value.
- 23 2) Otherwise, search for a method binding with name *E*, starting from the current class
24 or module as described in §13.3.4.
- 25 i) If the binding is found and its value is not undef, the value of the *defined?-*
26 *expression* is an implementation-defined value, which shall be a trueish value.
- 27 ii) Otherwise, the value of the *defined?-expression* is **nil**.
- 28 g) Otherwise, the value of the *defined?-expression* is implementation-defined.

29 11.4.4 Binary operators

30 Syntax

```

1  equality-expression ::
2      relational-expression
3      | relational-expression <=> relational-expression
4      | relational-expression == relational-expression
5      | relational-expression === relational-expression
6      | relational-expression != relational-expression
7      | relational-expression =~ relational-expression
8      | relational-expression !~ relational-expression

9  relational-expression ::
10     bitwise-OR-expression
11     | relational-expression > bitwise-OR-expression
12     | relational-expression >= bitwise-OR-expression
13     | relational-expression < bitwise-OR-expression
14     | relational-expression <= bitwise-OR-expression

15  bitwise-OR-expression ::
16     bitwise-AND-expression
17     | bitwise-OR-expression | bitwise-AND-expression
18     | bitwise-OR-expression ^ bitwise-AND-expression

19  bitwise-AND-expression ::
20     bitwise-shift-expression
21     | bitwise-AND-expression whitespace-before-operator? & bitwise-shift-expression

22  bitwise-shift-expression ::
23     additive-expression
24     | bitwise-shift-expression whitespace-before-operator? << additive-expression
25     | bitwise-shift-expression >> additive-expression

26  additive-expression ::
27     multiplicative-expression
28     | additive-expression whitespace-before-operator? + multiplicative-expression
29     | additive-expression whitespace-before-operator? - multiplicative-expression

30  multiplicative-expression ::
31     unary-minus-expression
32     | multiplicative-expression whitespace-before-operator? * unary-minus-expression
33     | multiplicative-expression whitespace-before-operator? / unary-minus-expression
34     | multiplicative-expression whitespace-before-operator? % unary-minus-expression

35  power-expression ::
36     unary-expression
37     | - ( numeric-literal ) ** power-expression
38     | unary-expression ** power-expression

```

```

1  binary-operator ::=
2      <=> | == | === | =~ | > | >= | < | <= | | | ^
3      | & | << | >> | + | - | * | / | % | ** | != | !~

4  whitespace-before-operator ::
5      [ whitespace here ]

```

6 If a *whitespace-before-operator* is present, *whitespaces* shall be omitted between the operator
7 after the *whitespace-before-operator* (i.e. &, <<, +, -, *, /, or %) and the nonterminal after that
8 operator.

9 Semantics

10 An *equality-expression* is evaluated as follows:

11 a) If the *equality-expression* is of the form $x != y$, take the following steps:

12 1) Evaluate x . Let X be the resulting value.

13 2) Evaluate y . Let Y be the resulting value.

14 3) Invoke the method `==` on X with a list of arguments which contains only one value Y .
15 If the resulting value is a trueish value, the value of the *equality-expression* is **false**.
16 Otherwise, the value of the *equality-expression* is **true**.

17 A conforming processor may take the steps in Step c instead of the above steps. In this case,
18 the processor shall define an instance method `!=` in the class `Object`, one of its superclasses,
19 or a module included in the class `Object`. The method `!=` shall take one argument and shall
20 return the value of the *equality-expression* in Step a-3, where let X and Y be the receiver
21 and the argument, respectively.

22 b) If the *equality-expression* is of the form $x !~ y$, take the following steps:

23 1) Evaluate x . Let X be the resulting value.

24 2) Evaluate y . Let Y be the resulting value.

25 3) Invoke the method `=~` on X with a list of arguments which contains only one value Y .
26 If the resulting value is a trueish value, the value of the *equality-expression* is **false**.
27 Otherwise, the value of the *equality-expression* is **true**.

28 A conforming processor may take the steps in Step c instead of the above steps. In this case,
29 the processor shall define an instance method `!~` in the class `Object`, one of its superclasses,
30 or a module included in the class `Object`. The method `!~` shall take one argument and shall
31 return the value of the *equality-expression* in Step a-3, where let X and Y be the receiver
32 and the argument, respectively.

33 c) Otherwise, if an *equality-expression* of the form $x \text{ binary-operator } y$, take the following
34 steps:

35 1) Evaluate x . Let X be the resulting value.

- 1 2) Evaluate y . Let Y be the resulting value.
- 2 3) Invoke the method whose name is the *binary-operator* on X with a list of arguments
3 which contains only one value Y . The value of the *equality-expression* is the resulting
4 value of the invocation.

5 11.5 Primary expressions

6 11.5.1 General description

7 Syntax

8 *primary-expression* ::
9 *class-definition*
10 | *eigenclass-definition*
11 | *module-definition*
12 | *method-definition*
13 | *singleton-method-definition*
14 | *yield-with-optional-argument*
15 | *if-expression*
16 | *unless-expression*
17 | *case-expression*
18 | *while-expression*
19 | *until-expression*
20 | *for-expression*
21 | *return-without-argument*
22 | *break-without-argument*
23 | *next-without-argument*
24 | *redo-expression*
25 | *retry-expression*
26 | *rescue-expression*
27 | *grouping-expression*
28 | *variable-reference*
29 | *scoped-constant-reference*
30 | *array-constructor*
31 | *hash-constructor*
32 | *literal*
33 | *defined?-with-parentheses*
34 | *primary-method-invocation*

35 Semantics

- 36 See §13.2.2 for *class-definition*.
- 37 See §13.4.2 for *eigenclass-definition*.
- 38 See §13.1.2 for *module-definition*.
- 39 See §13.3.1 for *method-definition*.
- 40 See §13.4.3 for *singleton-method-definition*.

1 See §11.3.5 for *yield-with-optional-argument*.

2 See §11.4.3.2 for *defined?-with-parentheses*.

3 See §11.3 for *primary-method-invocation*.

4 11.5.2 Control structures

5 11.5.2.1 Conditional expressions

6 11.5.2.1.1 The if expression

7 Syntax

8 *if-expression* ::
9 **if** *expression* *then-clause* *elsif-clause** *else-clause*? **end**

10 *then-clause* ::
11 *separator* *compound-statement*
12 | *separator*? **then** *compound-statement*

13 *else-clause* ::
14 **else** *compound-statement*

15 *elsif-clause* ::
16 **elsif** *expression* *then-clause*

17 The *expression* of an *if-expression* or *elsif-clause* shall not be a *jump-expression*.

18 Semantics

19 The *if-expression* is evaluated as follows:

- 20 a) Evaluate *expression*. Let *V* be the resulting value.
- 21 b) If *V* is a trueish value, evaluate the *compound-statement* of the *then-clause*. The value of
22 the *if-expression* is the resulting value. In this case, *elsif-clauses* and the *else-clause*, if any,
23 are not evaluated.
- 24 c) If *V* is a falseish value, and if there is no *elsif-clause* and no *else-clause*, then the value of
25 the *if-expression* is **nil**.
- 26 d) If *V* is a falseish value, and if there is no *elsif-clause* but there is an *else-clause*, then
27 evaluate the *compound-statement* of the *else-clause*. The value of the *if-expression* is the
28 resulting value.
- 29 e) If *V* is a falseish value, and if there are one or more *elsif-clauses*, evaluate the sequence of
30 *elsif-clauses* as follows:

- 1) Evaluate the *expression* of each *elsif-clause* in the order they appear in the program text, until there is an *elsif-clause* for which *expression* evaluates to a trueish value. Let *T* be this *elsif-clause*.
- 2) If *T* exists, evaluate the *compound-statement* of its *then-clause*. The value of the *if-expression* is the resulting value. Other *elsif-clauses* and an *else-clause* following *T*, if any, are not evaluated.
- 3) If *T* does not exist, and if there is an *else-clause*, then evaluate the *compound-statement* of the *else-clause*. The value of the *if-expression* is the resulting value.
- 4) If *T* does not exist, and if there is no *else-clause*, then the value of the *if-expression* is **nil**.

11.5.2.1.2 The unless expression

Syntax

unless-expression ::
unless *expression then-clause else-clause?* **end**

The *expression* of an *unless-expression* shall not be a *jump-expression*.

Semantics

The *unless-expression* is evaluated as follows:

- a) Evaluate the *expression*. Let *V* be the resulting value.
- b) If *V* is a falseish value, evaluate the *compound-statement* of the *then-clause*. The value of the *unless-expression* is the resulting value. In this case, the *else-clause*, if any, is not evaluated.
- c) If *V* is a trueish value, and if there is no *else-clause*, then the value of the *unless-expression* is **nil**.
- d) If *V* is a trueish value, and if there is an *else-clause*, then evaluate the *compound-statement* of the *else-clause*. The value of the *unless-expression* is the resulting value.

11.5.2.1.3 The case expression

Syntax

case-expression ::
case-expression-with-expression
| *case-expression-without-expression*

case-expression-with-expression ::
case *expression separator-list?* *when-clause* + *else-clause?* **end**

```

1  case-expression-without-expression ::
2      case separator-list? when-clause + else-clause? end

3  when-clause ::
4      when when-argument then-clause

5  when-argument ::
6      operator-expression-list ( , splating-argument )?
7      | splating-argument

```

8 The *expression* of a *case-expression-with-expression* shall not be a *jump-expression*.

9 Semantics

10 A *case-expression* is evaluated as follows:

- 11 a) If the *case-expression* is a *case-expression-with-expression*, evaluate the *expression*. Let V
12 be the resulting value.
- 13 b) The meaning of the phrase “ O is matching” in this step is defined as follows:

- 14 1) If the *case-expression* is a *case-expression-with-expression*, invoke the method `===` on
15 O with a list of arguments which contains only one value V . O is matching if and only
16 if the resulting value is a trueish value.
- 17 2) If the *case-expression* is a *case-expression-without-expression*, O is matching if and only
18 if O is a trueish value.

19 Search the *when-clauses* in the order they appear in the program text for a matching *when-*
20 *clause* as follows:

- 21 1) If the *operator-expression-list* of the *when-argument* is present:
- 22 i) For each of its *operator-expressions*, evaluate it and test if the resulting value is
23 matching.
- 24 ii) If a matching value is found, other *operator-expressions*, if any, are not evaluated.
- 25 2) If no matching value is found, and the *splating-argument* is present:
- 26 i) Construct a list of values from it as described in §11.3.2. For each element of the
27 resulting list, in the indexing order, test if it is matching.
- 28 ii) If a matching value is found, other values, if any, are not evaluated.
- 29 3) A *when-clause* is considered to be matching if and only if a matching value is found in
30 its *when-argument*. Later *when-clauses*, if any, are not tested in this case.
- 31 c) If one of the *when-clauses* is matching, evaluate the *compound-statement* of the *then-clause*
32 of this *when-clause*. The value of the *case-expression* is the resulting value.

- 1 d) If none of the *when-clauses* is matching, and if there is an *else-clause*, then evaluate the
2 *compound-statement* of the *else-clause*. The value of the *case-expression* is the resulting
3 value.
- 4 e) Otherwise, the value of the *case-expression* is **nil**.

5 11.5.2.1.4 Conditional operator

6 Syntax

7 *conditional-operator-expression* ::
8 *range-constructor*
9 | *range-constructor* ? *operator-expression*₁ : *operator-expression*₂

10 Semantics

11 A *conditional-operator-expression* of the form *range-constructor* ? *operator-expression*₁ : *operator-*
12 *expression*₂ is evaluated as follows:

- 13 a) Evaluate the *range-constructor*.
- 14 b) If the resulting value is a trueish value, evaluate the *operator-expression*₁. The value of the
15 *conditional-operator-expression* is the resulting value of the evaluation.
- 16 c) Otherwise, evaluate the *operator-expression*₂. The value of the *conditional-operator-expression*
17 is the resulting value of the evaluation.

18 11.5.2.2 Iteration expressions

19 11.5.2.2.1 General description

20 Syntax

21 *iteration-expression* ::=
22 *while-expression*
23 | *until-expression*
24 | *for-expression*
25 | *while-modifier-statement*
26 | *until-modifier-statement*

27 Each *iteration-expression* has a **body**. The body of a *while-expression* or an *until-expression* is
28 its *compound-statement*. The body of a *while-modifier-statement* or an *until-modifier-statement*
29 is its *statement*.

30 See §12.5 for *while-modifier-statement*.

31 See §12.6 for *until-modifier-statement*.

1 11.5.2.2.2 The while expression

2 Syntax

3 *while-expression* ::
4 **while** *expression do-clause* **end**

5 *do-clause* ::
6 *separator compound-statement*
7 | **do** *compound-statement*

8 The *expression* of a *while-expression* shall not be a *jump-expression*.

9 Semantics

10 A *while-expression* is evaluated as follows:

- 11 a) Evaluate the *expression*. Let *V* be the resulting value.
- 12 b) If *V* is a falseish value, terminate the evaluation of the *while-expression*. The value of the
13 *while-expression* is **nil**.
- 14 c) Otherwise, evaluate the *compound-statement* of the *do-clause*. If this evaluation:
 - 15 1) is terminated by a *break-expression*, terminate the evaluation of the *while-expression*.
16 If the *jump-argument* of the *break-expression* is present, the value of the *while-expression*
17 is the value of the *jump-argument*. Otherwise, the value of the *while-expression* is **nil**.
 - 18 2) is terminated by a *next-expression*, continue processing from Step a.
 - 19 3) is terminated by a *redo-expression*, continue processing from Step c.
- 20 Otherwise, unless this evaluation is terminated by a *return-expression*, continue processing
21 from Step a.

22 11.5.2.2.3 The until expression

23 Syntax

24 *until-expression* ::
25 **until** *expression do-clause* **end**

26 The *expression* of an *until-expression* shall not be a *jump-expression*.

27 Semantics

28 An *until-expression* is evaluated as follows:

- 1 a) Evaluate the *expression*. Let V be the resulting value.
- 2 b) If V is a trueish value, terminate the evaluation of the *until-expression*. The value of the
3 *until-expression* is **nil**.
- 4 c) Otherwise, evaluate the *compound-statement* of the *do-clause*. If this evaluation:
 - 5 1) is terminated by a *break-expression*, terminate the evaluation of the *until-expression*.
6 If the *jump-argument* of the *break-expression* is present, the value of the *until-expression*
7 is the value of the *jump-argument*. Otherwise, the value of the *until-expression* is **nil**.
 - 8 2) is terminated by a *next-expression*, continue processing from Step a.
 - 9 3) is terminated by a *redo-expression*, continue processing from Step c.
- 10 Otherwise, unless this evaluation is terminated by a *return-expression*, continue processing
11 from Step a.

12 11.5.2.2.4 The for expression

13 Syntax

14 *for-expression* ::
15 **for** *for-variable* **in** *expression* *do-clause* **end**

16 *for-variable* ::
17 *left-hand-side*
18 | *multiple-left-hand-side*

19 The *expression* of a *for-expression* shall not be a *jump-expression*.

20 Semantics

21 A *for-expression* is evaluated as follows:

- 22 a) Evaluate the *expression*. Let O be the resulting value.
- 23 b) Let E be the *primary-method-invocation* of the form *primary-expression* [no *line-terminator*
24 here] . **each do** | *block-parameter-list* | *block-body* **end**, where the value of the *primary-*
25 *expression* is O , the *block-parameter-list* is the *for-variable*, the *block-body* is the *compound-*
26 *statement* of the *do-clause*.
27 Evaluate E , but skip Step c of §11.3.3.
- 28 c) The value of the *for-expression* is the resulting value of the invocation.

1 11.5.2.3 Jump expressions

2 11.5.2.3.1 General description

3 Syntax

```
4 jump-expression ::=
5     return-expression
6     | break-expression
7     | next-expression
8     | redo-expression
9     | retry-expression
```

10 Semantics

11 Jump expressions are used to terminate the evaluation of a *method-body*, a *block-body*, the body
12 of an *iteration-expression*, or the *compound-statement*₂ of a *rescue-clause*.

13 In this document, the **current block** or the **current iteration-expression** refers to either of
14 the following:

- 15 • If the current method invocation exists, the current *block* or the current *iteration-expression*
16 is the *block* or the *iteration-expression* whose evaluation started most recently among the
17 *blocks* or *iteration-expressions* which are being evaluated on during the evaluation of the
18 current method invocation.
- 19 • Otherwise, the current *block* or the current *iteration-expression* is the *block* or the *iteration-*
20 *expression* whose evaluation started most recently among the *blocks* or *iteration-expressions*
21 which are under evaluation.

22 11.5.2.3.2 The return expression

23 Syntax

```
24 return-expression ::=
25     return-without-argument
26     | return-with-argument
```

```
27 return-without-argument ::
28     return
```

```
29 return-with-argument ::
30     return jump-argument
```

```
31 jump-argument ::
32     argument
```

1 The *block-argument* of the *argument-in-parentheses* of the *argument* (see §11.3.2) of a *jump-*
2 *argument* shall be omitted.

3 Semantics

4 *return-expressions* and *jump-arguments* are evaluated as follows:

5 a) A *return-expression* is evaluated as follows:

6 1) Let M be the *method-body* which corresponds to the current method invocation. If
7 such an invocation does not exist, or has already terminated:

8 i) Let S be a direct instance of the class `Symbol` with name `return`.

9 ii) If the *jump-argument* of the *return-expression* is present, let V be the value of the
10 *jump-argument*. Otherwise, let V be `nil`.

11 iii) Raise a direct instance of the class `LocalJumpError` which has two instance vari-
12 able bindings, one named `@reason` with the value S and the other named `@exit_value`
13 with the value V .

14 2) Evaluate the *jump-argument*, if any, as described below.

15 3) If there are *block-bodys* which include the *return-expression* and are included in M , ter-
16minate the evaluations of such *block-bodys*, from innermost to outermost (see §11.3.3).

17 4) Terminate the evaluation of M (see §13.3.3).

18 b) A *jump-argument* is evaluated as follows:

19 1) If the *jump-argument* is a *splatting-argument*:

20 i) Construct a list of values from the *splatting-argument* as described in §11.3.2 and
21 let L be the resulting list.

22 ii) If the length of L is 0 or 1, the value of the *jump-argument* is an implementation-
23 defined value.

24 iii) If the length of L is larger than 1, create a direct instance of the class `Array`
25 and store the elements of L in it, preserving their order. The value of the *jump-*
26 *argument* is the instance of the class `Array`.

27 2) Otherwise:

28 i) Construct a list of values from the *argument* as described in §11.3.2 and let L be
29 the resulting list.

30 ii) If the length of L is 1, the value of the *jump-argument* is the only element of L .

31 iii) If the length of L is larger than 1, create a direct instance of the class `Array`
32 and store the elements of L in it, preserving their order. The value of the *jump-*
33 *argument* is the instance of the class `Array`.

1 11.5.2.3.3 The break expression

2 Syntax

3 *break-expression* ::=
4 *break-without-argument*
5 | *break-with-argument*

6 *break-without-argument* ::
7 **break**

8 *break-with-argument* ::
9 **break** *jump-argument*

10 Semantics

11 A *break-expression* is evaluated as follows:

- 12 a) Evaluate the *jump-argument*, if any, as described in §11.5.2.3.2.
- 13 b) Let *E* be the current *block* or the current *iteration-expression*. If such a *block* or an *iteration-*
14 *expression* does not exist:
- 15 1) Let *S* be a direct instance of the class `Symbol` with name `break`.
- 16 2) If the *jump-argument* of the *break-expression* is present, let *V* be the value of the
17 *jump-argument*. Otherwise, let *V* be `nil`.
- 18 3) Raise a direct instance of the class `LocalJumpError` which has two instance variable
19 bindings, one named `@reason` with the value *S* and the other named `@exit_value` with
20 the value *V*.
- 21 c) If *E* is a *block*, terminate the evaluation of the *block-body* of *E* (see §11.3.3).
- 22 d) If *E* is an *iteration-expression*, terminate the evaluation of the body of *E* (see §11.5.2.2).

23 11.5.2.3.4 The next expression

24 Syntax

25 *next-expression* ::=
26 *next-without-argument*
27 | *next-with-argument*

28 *next-without-argument* ::
29 **next**

1 *next-with-argument* ::
2 **next** *jump-argument*

3 **Semantics**

4 A *next-expression* is evaluated as follows:

- 5 a) Evaluate the *jump-argument*, if any, as described in §11.5.2.3.2.
- 6 b) Let E be the current *block* or the current *iteration-expression*. If such a *block* or an *iteration-expression* does not exist:
 - 7
 - 8 1) Let S be a direct instance of the class `Symbol` with name `next`.
 - 9 2) If the *jump-argument* of the *next-expression* is present, let V be the value of the *jump-argument*. Otherwise, let V be `nil`.
 - 10
 - 11 3) Raise a direct instance of the class `LocalJumpError` which has two instance variable
 - 12 bindings, one named `@reason` with the value S and the other named `@exit_value` with
 - 13 the value V .
- 14 c) If E is a *block*, terminate the evaluation of the *block-body* of E (see §11.3.3).
- 15 d) If E is an *iteration-expression*, terminate the evaluation of the body of E (see §11.5.2.2).

16 **11.5.2.3.5 The redo expression**

17 **Syntax**

18 *redo-expression* ::
19 **redo**

20 **Semantics**

21 A *redo-expression* is evaluated as follows:

- 22 a) Let E be the current *block* or the current *iteration-expression*. If such a *block* or an *iteration-expression* does not exist,
 - 23
 - 24 1) Let S be a direct instance of the class `Symbol` with name `redo`.
 - 25 2) Raise a direct instance of the class `LocalJumpError` which has two instance variable
 - 26 bindings, one named `@reason` with the value S and the other named `@exit_value` with
 - 27 the value `nil`.
- 28 b) If E is a *block*, terminate the evaluation of the *block-body* of E (see §11.3.3).
- 29 c) If E is an *iteration-expression*, terminate the evaluation of the body of E (see §11.5.2.2).

1 11.5.2.3.6 The retry expression

2 Syntax

3 *retry-expression* ::
4 **retry**

5 Semantics

6 A *retry-expression* is evaluated as follows:

- 7 a) If the current method invocation exists, let M be the *method-body* which corresponds to
8 the current method invocation. Otherwise, let M be the *program*.
- 9 b) Let E be the innermost *rescue-clause* in M which encloses the *retry-expression*. If such a
10 *rescue-clause* does not exist, the behavior is unspecified.
- 11 c) Terminate the evaluation of the *compound-statement*₂ of E (see §11.5.2.4.1).

12 11.5.2.4 Exceptions

13 11.5.2.4.1 The rescue expression

14 Syntax

15 *rescue-expression* ::
16 **begin** *body-statement* **end**

17 *body-statement* ::
18 *compound-statement* *rescue-clause** *else-clause*? *ensure-clause*?

19 *rescue-clause* ::
20 **rescue** [no *line-terminator* here] *exception-class-list*?
21 *exception-variable-assignment*? *then-clause*

22 *exception-class-list* ::
23 *operator-expression*
24 | *multiple-right-hand-side*

25 *exception-variable-assignment* ::
26 **=>** *left-hand-side*

27 *ensure-clause* ::
28 **ensure** *compound-statement*

1 The *operator-expression* of an *exception-class-list* shall not be a *jump-expression*.

2 Semantics

3 The value of a *rescue-expression* is the value of the *body-statement*.

4 A *body-statement* is evaluated as follows:

5 a) Evaluate the *compound-statement* of the *body-statement*.

6 b) If no exception is raised, or all the raised exceptions are handled during Step a:

7 1) If the *else-clause* is present, evaluate the *else-clause* as described in §11.5.2.1.1.

8 2) If the *ensure-clause* is present, evaluate its *compound-statement*. The value of this
9 evaluation is the value of the *ensure-clause*.

10 If both the *else-clause* and the *ensure-clause* are present, the value of the *body-statement*
11 is the value of the *ensure-clause*. If only one of these clauses is present, the value of the
12 *body-statement* is the value of the clause.

13 If neither the *else-clause* nor the *ensure-clause* is present, the value of the *body-statement*
14 is the value of its *compound-statement*.

15 c) If an exception is raised and not handled during Step a, test each *rescue-clause*, if any, in
16 the order it occurs in the program text. The test determines whether the *rescue-clause* can
17 handle the exception as follows:

18 1) Let E be the exception raised.

19 2) If the *exception-class-list* is omitted in the *rescue-clause*, and if E is an instance of the
20 class `StandardError`, the *rescue-clause* handles E .

21 3) If the *exception-class-list* of the *rescue-clause* is present:

22 i) If the *exception-class-list* is of the form *operator-expression*, evaluate the *operator-*
23 *expression*. Create an empty list of values, and append the value of the *operator-*
24 *expression* to the list.

25 ii) If the *exception-class-list* is of the form *multiple-right-hand-side*, construct a list
26 of values from the *multiple-right-hand-side* (see §11.4.2.4).

27 Let L be the list created by evaluating the *exception-class-list* as above. Compare each
28 element D of L with E as follows:

29 i) If D is neither the class `Exception` nor a subclass of the class `Exception`, raise a
30 direct instance of the class `TypeError`.

31 ii) If E is an instance of D , the *rescue-clause* handles E . In this case, any remaining
32 *rescue-clauses* in the *body-statement* are not tested.

33 d) If a *rescue-clause* R which can handle E is found:

- 1 1) If the *exception-variable-assignment* of R is present, evaluate it in the same way as
2 a *multiple-assignment-statement* of the form *left-hand-side* = *multiple-right-hand-side*
3 where the value of *multiple-right-hand-side* is E .
- 4 2) Evaluate the *compound-statement* of the *then-clause* of R . If this evaluation is termi-
5 nated by a *retry-expression*, continue processing from Step a. Otherwise, let V be the
6 value of this evaluation.
- 7 3) If the *ensure-clause* is present, evaluate it. The value of the *body-statement* is the value
8 of the *ensure-clause*.
- 9 4) If the *ensure-clause* is omitted, the value of the *body-statement* is V .
- 10 e) If no *rescue-clause* is present or if a *rescue-clause* which can handle E is not found, evaluate
11 the *ensure-clause*. In this case, the value of the *body-statement* is undefined.

12 The *ensure-clause* of a *body-statement*, if any, is always evaluated, even when the evaluation of
13 *body-statement* is terminated by a *jump-expression*.

14 11.5.3 Grouping expression

15 Syntax

16 *grouping-expression* ::
17 (*expression*)
18 | (*compound-statement*)

19 Semantics

20 A *grouping-expression* is evaluated as follows:

- 21 a) Evaluate the *expression* or the *compound-statement*.
- 22 b) The value of the *grouping-expression* is the resulting value.

23 11.5.4 Variable references

24 11.5.4.1 General description

25 Syntax

26 *variable-reference* ::
27 *variable*
28 | *pseudo-variable*

29 *variable* ::
30 *constant-identifier*
31 | *global-variable-identifier*

```

1      | class-variable-identifier
2      | instance-variable-identifier
3      | local-variable-identifier

4  scoped-constant-reference ::
5      primary-expression [no whitespace here] :: constant-identifier
6      | :: constant-identifier

```

7 **11.5.4.2 Constants**

8 A *constant-identifier* is evaluated as follows:

9 a) Let N be the *constant-identifier*.

10 b) Search for a binding of a constant with name N as described below.

11 As soon as the binding is found in any of the following steps, the evaluation of the *constant-identifier* is terminated and the value of the *constant-identifier* is the value of the binding found.

14 c) Let L be the top of `[[class-module-list]]`. Search for a binding of a constant with name N in each element of L from start to end, including the first element, which is the current class or module, but except for the last element, which is the class `Object`.

17 d) If a binding is not found, let C be the current class or module.

18 Let L be the included module list of C . Search each element of L in the reverse order for a binding of a constant with name N .

20 e) If the binding is not found:

21 1) If C is a class:

22 i) If C does not have a direct superclass, create a direct instance of the class `Symbol` with name N , and let R be that instance. Invoke the method `const_missing` on the current class or module with R as the only argument.

25 ii) Let S be the direct superclass of C .

26 iii) Search for a binding of a constant with name N in S .

27 iv) If the binding is not found, let L be the included module list of S and search each element of L in the reverse order for a binding of a constant with name N .

29 v) If the binding is not found, let C be the direct superclass of S . Continue processing from Step e-1-i.

31 2) If C is a module:

32 i) Search for a binding of a constant with name N in the class `Object`.

- 1 ii) If the binding is not found, let L be the included module list of the class `Object`
2 and search each element of L in the reverse order for a binding of a constant with
3 name N .
- 4 iii) If the binding is not found, create a direct instance of the class `Symbol` with name
5 N , and let R be that instance. Invoke the method `const_missing` on the current
6 class or module with R as the only argument.

7 **11.5.4.3 Scoped constants**

8 A *scoped-constant-reference* is evaluated as follows:

- 9 a) If the *primary-expression* is present, evaluate it and let C be the resulting value. Otherwise,
10 let C be the class `Object`.
- 11 b) If C is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.
- 12 c) Otherwise:
- 13 1) Let N be the *constant-identifier*.
- 14 2) If a binding with name N exists in the set of bindings of constants of C , the value of
15 the *scoped-constant-reference* is the value of the binding.
- 16 3) Otherwise:
- 17 i) Let L be the included module list of C . Search each element of L in the reverse
18 order for a binding of a constant with name N .
- 19 ii) If the binding is found, the value of the *scoped-constant-reference* is the value of
20 the binding.
- 21 iii) Otherwise, search for a binding of a constant with name N from Step e of §11.5.4.2.

22 **11.5.4.4 Global variables**

23 A *global-variable-identifier* is evaluated as follows:

- 24 a) Let N be the *global-variable-identifier*.
- 25 b) If a binding with name N exists in `[[global-variable-bindings]]`, the value of *global-variable-*
26 *identifier* is the value of the binding.
- 27 c) Otherwise, the value of *global-variable-identifier* is **nil**.

28 **11.5.4.5 Class variables**

29 A *class-variable-identifier* is evaluated as follows:

- 30 a) Let N be the *class-variable-identifier*. Let C be the first class or module in the list at the
31 top of `[[class-module-list]]` which is not an eigenclass.

- 1 b) Let CS be the set of classes which consists of C and all the superclasses of C . Let MS be
2 the set of modules which consists of all the modules in the included module list of all classes
3 in CS . Let CM be the union of CS and MS .
- 4 c) If a binding with name N exists in the set of bindings of class variables of only one of the
5 classes or modules in CM , let V be the value of the binding.
- 6 d) If more than two classes or modules in CM have a binding with name N in the set of
7 bindings of class variables, let V be the value of one of these bindings. Which binding is
8 selected is unspecified.
- 9 e) If none of the classes or modules in CM has a binding with name N in the set of bindings
10 of class variables, let S be a direct instance of the class `Symbol` with name N and raise a
11 direct instance of the class `NameError` which has S as its name property.
- 12 f) The value of the *class-variable-identifier* is V .

13 11.5.4.6 Instance variables

14 An *instance-variable-identifier* is evaluated as follows:

- 15 a) Let N be the *instance-variable-identifier*.
- 16 b) If a binding with name N exists in the set of bindings of instance variables of the current
17 self, the value of the *instance-variable-identifier* is the value of the binding.
- 18 c) Otherwise, the value of the *instance-variable-identifier* is **nil**.

19 11.5.4.7 Local variables or method invocations

20 11.5.4.7.1 General description

21 An occurrence of a *local-variable-identifier* in a *variable-reference* can be a reference to a local
22 variable or a method invocation.

23 11.5.4.7.2 Determination of the type of local variable identifiers

24 A conforming processor shall determine whether the occurrence of a *local-variable-identifier* I is
25 a reference to a local variable or a method invocation as follows:

- 26 a) Let P be the point where I occurs.
- 27 b) Let S be the innermost local variable scope which encloses P and which does not correspond
28 to a *block* (see §9.2).
- 29 c) Let R be the region of a program between the beginning of S and P .
- 30 d) If the same identifier as I occurs as a reference to a local variable in *variable-reference* in
31 R , then I is a reference to a local variable.
- 32 e) If the same identifier as I occurs in one of the the forms below in R , then I is a reference
33 to a local variable.

- 1 • *mandatory-parameter*
- 2 • *optional-parameter-name*
- 3 • *array-parameter-name*
- 4 • *proc-parameter-name*
- 5 • *variable of left-hand-side*
- 6 • *variable of single-variable-assignment-expression*
- 7 • *variable of single-variable-assignment-statement*
- 8 • *variable of abbreviated-variable-assignment-expression*
- 9 • *variable of abbreviated-variable-assignment-statement*
- 10 f) Otherwise, *I* is a method invocation.

11 11.5.4.7.3 Local variables

12 If a *local-variable-identifier* is a reference to a local variable, it is evaluated as follows:

- 13 a) Let *N* be the *local-variable-identifier*.
- 14 b) Search for a binding of a local variable with name *N* as described in §9.2.
- 15 c) If a binding is found, the value of *local-variable-identifier* is the value of the binding.
- 16 d) Otherwise, the value of *local-variable-identifier* is **nil**.

17 11.5.4.7.4 Method invocations

18 If a *local-variable-identifier* is a method invocation, it is evaluated as follows:

- 19 a) Let *N* be the *local-variable-identifier*.
- 20 b) Create an empty list of arguments *L*, and invoke the method *N* on the current self with *L*
- 21 as the list of arguments.

22 11.5.4.8 Pseudo variables

23 11.5.4.8.1 General description

24 Syntax

25 *pseudo-variable* ::
 26 *nil*
 27 | *true*
 28 | *false*
 29 | *self*

1 11.5.4.8.2 The nil expression

2 Syntax

3 *nil-expression* ::
4 **nil**

5 Semantics

6 A *nil-expression* evaluates to **nil**, which is the only instance of the class NilClass (see §6.5).

7 11.5.4.8.3 The true expression and the false expression

8 Syntax

9 *true-expression* ::
10 **true**

11 *false-expression* ::
12 **false**

13 Semantics

14 A *true-expression* evaluates to **true**, which is the only instance of the class TrueClass. A
15 *false-expression* evaluates to **false**, which is the only instance of the class NilClass (see §6.5).

16 11.5.4.8.4 The self expression

17 Syntax

18 *self-expression* ::
19 **self**

20 Semantics

21 A *self-expression* evaluates to the value of the current self.

22 11.5.5 Object constructors

23 11.5.5.1 Array constructor

24 Syntax

1 *array-constructor* ::
2 [*indexing-argument-list*?]

3 **Semantics**

4 An *array-constructor* is evaluated as follows:

- 5 a) If there is an *indexing-argument-list*, construct a list of arguments from the *indexing-*
6 *argument-list* as described in §11.3.2. Let *L* be the resulting list.
- 7 b) Otherwise, create an empty list of values *L*.
- 8 c) Create a direct instance of the class **Array** which stores the values in *L* in the same order
9 they are stored in *L*. Let *O* be the instance.
- 10 d) The value of the *array-constructor* is *O*.

11 **11.5.5.2 Hash constructor**

12 **Syntax**

13 *hash-constructor* ::
14 { (*association-list* ,?)? }

15 *association-list* ::
16 *association* (, *association*)*

17 *association* ::
18 *association-key* => *association-value*

19 *association-key* ::
20 *operator-expression*

21 *association-value* ::
22 *operator-expression*

23 The *operator-expression* of an *association-key* or *association-value* shall not be a *jump-expression*.

24 **Semantics**

25 Both *hash-constructors* and *association-lists* evaluate to a direct instance of the class **Hash** (see
26 §15.2.13) as follows:

- 27 a) A *hash-constructor* is evaluated as follows:

- 1) If there is an *association-list*, evaluate the *association-list*. The value of the *hash-constructor* is the resulting value.
 - 2) Otherwise, create an empty direct instance of the class `Hash`. The value of the *hash-constructor* is the resulting instance.
- b) An *association-list* is evaluated as follows:
- 1) Create a direct instance of the class `Hash` H .
 - 2) For each *association* A_i , in the order it appears in the program text, take the following steps:
 - i) Evaluate the *operator-expression* of the *association-key* of A_i . Let K_i be the resulting value.
 - ii) Evaluate the *operator-expression* of the *association-value*. Let V_i be the resulting value.
 - iii) Store a pair of K_i and V_i in H , as if by invoking the method `[]=` on H with K_i and V_i as the arguments.
 - 3) The value of the *association-list* is H .

11.5.5.3 Range constructor

Syntax

range-constructor ::
logical-OR-expression
 | *logical-OR-expression*₁ *range-operator* *logical-OR-expression*₂

range-operator ::
 ..
 | ...

Semantics

A *range-constructor* of the form *logical-OR-expression*₁ *range-operator* *logical-OR-expression*₂ is evaluated as follows:

- a) Evaluate the *logical-OR-expression*₁. Let A be the resulting value.
 - b) Evaluate the *logical-OR-expression*₂. Let B be the resulting value.
 - c) If the *range-operator* is the terminal `..`, construct a list L which contains three arguments: A , B , and **false**.
- If the *range-operator* is the terminal `...`, construct a list L which contains three arguments: A , B , and **true**.

1 d) Invoke the method `new` on the class `Range` with L as the list of arguments. The value of
2 the *range-constructor* is the resulting value.

3 **11.5.6 Literals**

4 See §8.7.6.2 for integer literals.

5 See §8.7.6.2 for float literals.

6 See §8.7.6.3 for string literals.

7 See §8.7.6.6 for symbol literals.

8 See §8.7.6.5 for regular expression literals.

9 **12 Statements**

10 **12.1 General description**

11 **Syntax**

```
12 statement ::  
13     expression-statement  
14     | alias-statement  
15     | undef-statement  
16     | if-modifier-statement  
17     | unless-modifier-statement  
18     | while-modifier-statement  
19     | until-modifier-statement  
20     | rescue-modifier-statement  
21     | assignment-statement
```

22 **Semantics**

23 See §13.3.6 for *alias-statement*.

24 See §13.3.7 for *undef-statement*.

25 See §11.4.2 for *assignment-statement*.

26 **12.2 The expression statement**

27 **Syntax**

```
28 expression-statement ::  
29     expression
```

1 Semantics

2 An *expression-statement* is evaluated as follows:

- 3 a) Evaluate the *expression*.
- 4 b) The resulting value is the value of the *expression-statement*.

5 12.3 The if modifier statement

6 Syntax

7 *if-modifier-statement* ::
8 *statement* [no *line-terminator* here] **if** *expression*

9 The *expression* of an *if-modifier-statement* shall not be a *jump-expression*.

10 Semantics

11 An *if-modifier-statement* of the form *S if E*, where *S* is the *statement* and *E* is the *expression*,
12 is evaluated as follows:

- 13 a) Evaluate the *if-expression* of the form **if E then S end**.
- 14 b) The resulting value is the value of the *if-modifier-statement*.

15 12.4 The unless modifier statement

16 Syntax

17 *unless-modifier-statement* ::
18 *statement* [no *line-terminator* here] **unless** *expression*

19 The *expression* of an *unless-modifier-statement* shall not be a *jump-expression*.

20 Semantics

21 An *unless-modifier-statement* of the form *S unless E*, where *S* is the *statement* and *E* is the
22 *expression*, is evaluated as follows:

- 23 a) Evaluate the *unless-expression* of the form **unless E then S end**.
- 24 b) The resulting value is the value of the *unless-modifier-statement*.

25 12.5 The while modifier statement

26 Syntax

1 *while-modifier-statement* ::
2 *statement* [no *line-terminator* here] **while** *expression*

3 The *expression* of a *while-modifier-statement* shall not be a *jump-expression*.

4 **Semantics**

5 A *while-modifier-statement* of the form *S while E*, where *S* is the *statement* and *E* is the
6 *expression*, is evaluated as follows:

- 7 a) If *S* is a *rescue-expression*, the behavior is implementation-defined.
- 8 b) Evaluate the *while-expression* of the form **while** *E do S end*.
- 9 c) The resulting value is the value of the *while-modifier-statement*.

10 **12.6 The until modifier statement**

11 **Syntax**

12 *until-modifier-statement* ::
13 *statement* [no *line-terminator* here] **until** *expression*

14 The *expression* of an *until-modifier-statement* shall not be a *jump-expression*.

15 **Semantics**

16 An *until-modifier-statement* of the form *S until E*, where *S* is the *statement* and *E* is the
17 *expression*, is evaluated as follows:

- 18 a) If *S* is a *rescue-expression*, the behavior is implementation-defined.
- 19 b) Evaluate the *until-expression* of the form **until** *E do S end*.
- 20 c) The resulting value is the value of the *until-modifier-statement*.

21 **12.7 The rescue modifier statement**

22 **Syntax**

23 *rescue-modifier-statement* ::
24 *main-statement-of-rescue-modifier-statement* [no *line-terminator* here]
25 **rescue** *fallback-statement-of-rescue-modifier-statement*

26 *main-statement-of-rescue-modifier-statement* ::
27 *statement*

```

1  fallback-statement-of-rescue-modifier-statement ::
2      statement but not statements-not-allowed-in-fallback-statement

3  statements-not-allowed-in-fallback-statement ::
4      keyword-AND-expression
5      | keyword-OR-expression
6      | if-modifier-statement
7      | unless-modifier-statement
8      | while-modifier-statement
9      | until-modifier-statement
10     | rescue-modifier-statement

```

11 Semantics

12 A *rescue-modifier-statement* is evaluated as follows:

- 13 a) Evaluate the *main-statement-of-rescue-modifier-statement*. Let V be the resulting value.
- 14 b) If a direct instance of the class `StandardError` is raised and not handled in Step a, evalu-
15 ate *fallback-statement-of-rescue-modifier-statement*. The resulting value is the value of the
16 *rescue-modifier-statement*.
- 17 c) If no instances of the class `Exception` are raised in Step a, or all the instances of the class
18 `Exception` raised in Step a are handled in Step a, the value of the *rescue-modifier-statement*
19 is V .

20 13 Classes and modules

21 13.1 Modules

22 13.1.1 General description

23 Every module is an instance of the class `Module` (see §15.2.2). However, not every instance of
24 the class `Module` is a module because the class `Module` is a superclass of the class `Class`, an
25 instance of which is not a module, but a class.

26 Modules have the following attributes:

27 **Included module list:** An ordered list of modules included in the module. Module inclu-
28 sion is described in §13.1.3.

29 **Constants:** A set of bindings of constants.

30 A binding of a constant is created by the following program constructs:

- 31 • Assignments (see §11.4.2)
- 32 • Module definitions (see §13.1.2)

- 1 • Class definitions (see §13.2.2)

2 **Class variables:** A set of bindings of class variables. A binding of a class variable is
3 created by an assignment (see §11.4.2).

4 **Instance methods:** A set of method bindings. A method binding is created by a method
5 definition (see §13.3.1) or a singleton method definition (see §13.4.3). The value of a method
6 binding may be **undef**, which is the flag indicating that a method cannot be invoked (see
7 §13.3.7).

8 13.1.2 Module definition

9 Syntax

```
10 module-definition ::  
11     module module-path module-body end  
  
12 module-path ::  
13     top-module-path  
14     | module-name  
15     | nested-module-path  
  
16 module-name ::  
17     constant-identifier  
  
18 top-module-path ::  
19     :: module-name  
  
20 nested-module-path ::  
21     primary-expression [no line-terminator here] :: module-name  
  
22 module-body ::  
23     body-statement
```

24 Semantics

25 A *module-definition* is evaluated as follows:

- 26 a) Determine the class or module in which a binding with name *module-name* is to be created
27 or modified as follows:
 - 28 1) If the *module-path* is of the form *top-module-path*, let *C* be the class `Object`.
 - 29 2) If the *module-path* is of the form *module-name*, let *C* be the current class or module.
 - 30 3) If the *module-path* is of the form *nested-module-path*, evaluate the *primary-expression*.
31 If the resulting value is an instance of the class `Module`, let *C* be the instance. Otherwise,
32 raise a direct instance of the class `TypeError`.

- 1 b) Let N be the *module-name*.
- 2 1) If a binding with name N exists in the set of bindings of constants of C , let B be this
3 binding. If the value of B is a module, let M be that module. Otherwise, raise a direct
4 instance of the class `TypeError`.
- 5 2) Otherwise, create a direct instance of the class `Module` and let M be that module.
6 Create a variable binding with name N and value M in the set of bindings of constants
7 of C .
- 8 c) Modify the execution context as follows:
- 9 1) Create a new list which has the same members as that of the list at the top of `[[class-`
10 `module-list]]`, and add M to the head of the newly created list. Push the list onto
11 `[[class-module-list]]`.
- 12 2) Push M onto `[[self]]`.
- 13 3) Push the public visibility onto `[[default-visibility]]`.
- 14 4) Push an empty set of bindings onto `[[local-variable-bindings]]`.
- 15 d) Evaluate the *module-body*. The value of the *module-definition* is the value of the *module-*
16 *body*.
- 17 e) Restore the execution context by removing the elements from the tops of `[[class-module-`
18 `list]]`, `[[self]]`, `[[default-visibility]]`, and `[[local-variable-bindings]]`, even when an exception is
19 raised and not handled during Step d.

20 13.1.3 Module inclusion

21 Modules and classes can be extended by including other modules into them. When a module is
22 included, the instance methods, the class variables, and the constants of the included module
23 are available to the including class or module (see §11.5.4.5, §13.3.3, and §11.5.4.2).

24 Modules and classes can include other modules by invoking the method `include` (see §15.2.2.4.27)
25 or the method `extend` (see §15.3.1.3.13).

26 A module M is included in another module N if and only if M is an element of the included
27 module list of N . A module M is included in a class C if and only if M is an element of the
28 included module list of C , or M is included in one of the superclasses of C .

29 13.2 Classes

30 13.2.1 General description

31 Every class is an instance of the class `Class` (see §15.2.3), which is a direct subclass of the class
32 `Module`.

33 Classes have the same set of attributes as modules. In addition, every class has at most one
34 single direct superclass.

1 13.2.2 Class definition

2 Syntax

3 *class-definition* ::
4 **class** *class-path* [no *line-terminator* here] (< *superclass*)? *separator*
5 *class-body* **end**

6 *class-path* ::
7 *top-class-path*
8 | *class-name*
9 | *nested-class-path*

10 *class-name* ::
11 *constant-identifier*

12 *top-class-path* ::
13 :: *class-name*

14 *nested-class-path* ::
15 *primary-expression* [no *line-terminator* here] :: *class-name*

16 *superclass* ::
17 *expression*

18 *class-body* ::
19 *body-statement*

20 Semantics

21 A *class-definition* is evaluated as follows:

22 a) Determine the class or module in which the binding with name *class-name* is to be created
23 or modified as follows:

24 1) If the *class-path* is of the form *top-class-path*, let *M* be the class **Object**.

25 2) If the *class-path* is of the form *class-name*, let *M* be the current class or module.

26 3) If the *class-path* is of the form *nested-class-path*, evaluate the *primary-expression*. If
27 the resulting value is an instance of the class **Module**, let *M* be the instance. Otherwise,
28 raise a direct instance of the class **TypeError**.

29 b) Let *N* be the *class-name*.

30 1) If a binding with name *N* exists in the set of bindings of constants of *M*, let *B* be that
31 binding.

- 1 i) If the value of *B* is an instance of the class `Class`, let *C* be the instance. Otherwise,
2 raise a direct instance of the class `TypeError`.
- 3 ii) If the *superclass* is present, evaluate it. If the resulting value does not correspond
4 to the direct superclass of *C*, raise a direct instance of the class `TypeError`.
- 5 2) Otherwise, create a direct instance of the class `Class`. Let *C* be that class.
 - 6 i) If the *superclass* is present, evaluate it. If the resulting value is not an instance of
7 the class `Class`, raise a direct instance of the class `TypeError`. If the value of *su-*
8 *perclass* is an eigenclass or the class `Class`, the behavior is unspecified. Otherwise,
9 set the direct superclass of *C* to the value of the *superclass*.
 - 10 ii) If the *superclass* of the *class-definition* is omitted, set the direct superclass of *C*
11 to the class `Object`.
 - 12 iii) Create an eigenclass, and associate it with *C*. The eigenclass shall have the eigen-
13 class of the direct superclass of *C* as one of its superclasses.
 - 14 iv) Create a variable binding with name *N* and value *C* in the set of bindings of
15 constants of *M*.
- 16 c) Modify the execution context as follows:
 - 17 1) Create a new list which has the same members as that of the list at the top of `[[class-`
18 `module-list]]`, and add *C* to the head of the newly created list. Push the list onto
19 `[[class-module-list]]`.
 - 20 2) Push *C* onto `[[self]]`.
 - 21 3) Push the public visibility onto `[[default-visibility]]`.
 - 22 4) Push an empty set of bindings onto `[[local-variable-bindings]]`.
- 23 d) Evaluate the *class-body*. The value of the *class-definition* is the value of the *class-body*.
- 24 e) Restore the execution context by removing the elements from the tops of `[[class-module-`
25 `list]]`, `[[self]]`, `[[default-visibility]]`, and `[[local-variable-bindings]]`, even when an exception is
26 raised and not handled during Step d.

27 **13.2.3 Inheritance**

28 A class inherits attributes of its superclasses. Inheritance means that a class implicitly contains
29 all attributes of its superclasses, as described below:

- 30 • Constants and class variables of superclasses can be referenced (see §11.5.4.2 and §11.5.4.5).
- 31 • Singleton methods of superclasses can be invoked (see §13.4).
- 32 • Instance methods defined in superclasses can be invoked on an instance of their subclasses
33 (see §13.3.3).

1 13.2.4 Instance creation

2 A direct instance of a class can be created by invoking the method `new` on the class (see
3 §15.2.3.3.3).

4 13.3 Methods

5 13.3.1 Method definition

6 Syntax

7 *method-definition* ::
8 `def` *method-name* [*no line-terminator here*] *method-parameter-part*
9 *method-body* `end`

10 *method-name* ::
11 *method-identifier*
12 | *operator-method-name*
13 | *keyword*

14 *method-body* ::
15 *body-statement*

16 The following constructs shall not be present in the *method-parameter-part* or the *method-body*:

- 17 • A *class-definition*.
- 18 • A *module-definition*.
- 19 • A *single-variable-assignment*, where its *variable* is a *constant-identifier*.
- 20 • A *scoped-constant-assignment*.
- 21 • A *multiple-assignment-statement*, where the form of any of the *left-hand-sides* which is
22 present in it is any of the following:
 - 23 — *constant-identifier*
 - 24 — *primary-expression* [*no line-terminator here*] (`.` | `::`) (*local-variable-identifier* | *constant-*
25 *identifier*)
 - 26 — `::` *constant-identifier*.

27 However, those constructs may occur within an *eigenclass-definition* in the *method-parameter-*
28 *part* or the *method-body*.

29 Semantics

30 A method is defined by a *method-definition* and has the *method-parameter-part* and the *method-*
31 *body* of the *method-definition*. In addition, a method has the following attributes:

1 **Class module list:** The list of classes and modules which is the top element of `[[class-`
2 `module-list]]` when the method is defined.

3 **Defined name:** The name with which the method is defined.

4 **Visibility:** The visibility of the method (see §13.3.5).

5 A *method-definition* is evaluated as follows:

6 a) Let *N* be the *method-name*.

7 b) Create a method *U* defined by the *method-definition*. Initialize the attributes of *U* as
8 follows:

9 • The class module list is the element at the top of `[[class-module-list]]`.

10 • The defined name is *N*.

11 • The visibility is:

12 — If the current class or module is an eigenclass, then the current visibility.

13 — Otherwise, if *N* is `initialize` or `initialize_copy`, then the private visibility.

14 — Otherwise, the current visibility.

15 c) If a method binding with name *N* exists in the set of bindings of instance methods of the
16 current class or module, let *V* be the value of that binding.

17 1) If *V* is `undef`, the evaluation of the *method-definition* is implementation-defined.

18 2) Replace the value of the binding with *U*.

19 d) Otherwise, create a method binding with name *N* and value *U* in the set of bindings of
20 instance methods of the current class or module.

21 e) The value of the *method-definition* is implementation-defined.

22 13.3.2 Method parameters

23 Syntax

24 *method-parameter-part* ::
25 (*parameter-list?*)
26 | *parameter-list?* *separator*

27 *parameter-list* ::
28 *mandatory-parameter-list* , *optional-parameter-list?* ,
29 *array-parameter?* , *proc-parameter?*
30 | *optional-parameter-list* , *array-parameter?* , *proc-parameter?*

```

1      | array-parameter , proc-parameter?
2      | proc-parameter

3  mandatory-parameter-list ::
4      mandatory-parameter
5      | mandatory-parameter-list , mandatory-parameter

6  mandatory-parameter ::
7      local-variable-identifier

8  optional-parameter-list ::
9      optional-parameter
10     | optional-parameter-list , optional-parameter

11  optional-parameter ::
12     optional-parameter-name = default-parameter-expression

13  optional-parameter-name ::
14     local-variable-identifier

15  default-parameter-expression ::
16     operator-expression

17  array-parameter ::
18     * array-parameter-name
19     | *

20  array-parameter-name ::
21     local-variable-identifier

22  proc-parameter ::
23     & proc-parameter-name

24  proc-parameter-name ::
25     local-variable-identifier

```

26 All the *local-variable-identifiers* of *mandatory-parameters*, *optional-parameter-names*, *array-*
27 *parameter-name*, and *proc-parameter-name* of a *parameter-list* shall be pairwise different.

28 Semantics

29 There are four kinds of parameters as described below. How those parameters are bound to the
30 actual arguments is described in §13.3.3.

31 **Mandatory parameters:** These parameters are represented by *mandatory-parameters*.

1 For each mandatory parameter, a corresponding actual argument shall be given when the
2 method is invoked.

3 **Optional parameters:** These parameters are represented by *optional-parameters*. Each
4 optional parameter consists of a parameter name represented by *optional-parameter-name*
5 and an expression represented by *default-parameter-expression*. For each optional parame-
6 ter, when there is no corresponding argument in the list of arguments given to the method
7 invocation, the value of the *default-parameter-expression* is used as the value of the argu-
8 ment.

9 **An array parameter:** This parameter is represented by *array-parameter-name*. Let N be
10 the number of arguments, excluding a block argument, given to a method invocation. If N
11 is more than the sum of the number of mandatory arguments and optional arguments, this
12 parameter is bound to a direct instance of the class `Array` containing the extra arguments
13 excluding a block argument. Otherwise, the parameter is bound to an empty direct instance
14 of the class `Array`. If an *array-parameter* is of the form “*”, those extra arguments are
15 ignored.

16 **A proc parameter:** This parameter is represented by *proc-parameter-name*. The param-
17 eter is bound to a direct instance of the class `Proc` which represents the block passed to the
18 method invocation.

19 13.3.3 Method invocation

20 The way in which a list of arguments is created are described in §11.3.

21 Given the receiver R , the method name M , and the list of arguments A , take the following steps:

- 22 a) If the method is invoked with a block, let B be the block. Otherwise, let B be block-not-
23 given.
- 24 b) Let C be the eigenclass of R if R has an eigenclass. Otherwise, let C be the class of R .
- 25 c) Search for a method binding with name M , starting from C as described in §13.3.4.
- 26 d) If a binding is found and its value is not `undef`, let V be the value of the binding.
- 27 e) Otherwise, if M is `method_missing`, the behavior is unspecified. If M is not `method_missing`,
28 add a direct instance of the class `Symbol` with name M to the head of A , and invoke the
29 method `method_missing` on R with A as arguments and B as the block. Let O be the
30 resulting value, and go to Step j.
- 31 f) If the method is not invoked internally by a Ruby processor, check the visibility of V to see
32 whether the method can be invoked (see §13.3.5). If the method cannot be invoked, add a
33 direct instance of the class `Symbol` with name M to the head of A , and invoke the method
34 `method_missing` on R with A as arguments and B as the block. Let O be the resulting
35 value, and go to Step j.
- 36 g) Modify the execution context as follows:
 - 37 1) Push the class module list of V onto `[[class-module-list]]`.
 - 38 2) Push R onto `[[self]]`.

- 1 3) Push M onto $\llbracket\text{invoked-method-name}\rrbracket$.
- 2 4) Push the public visibility to $\llbracket\text{default-visibility}\rrbracket$.
- 3 5) Push the defined name of V onto $\llbracket\text{defined-method-name}\rrbracket$.
- 4 6) Push B onto $\llbracket\text{block}\rrbracket$.
- 5 7) Push an empty set of local variable bindings onto $\llbracket\text{local-variable-bindings}\rrbracket$.
- 6 h) Evaluate the *method-parameter-part* of V as follows:
 - 7 1) Let L be the *parameter-list* of the *method-parameter-part*.
 - 8 2) Let P_m , P_o , and P_a be the *mandatory-parameters* of the *mandatory-parameter-list*,
9 the *optional-parameters* of the *optional-parameter-list*, and the *array-parameter* of L ,
10 respectively. Let N_A , N_{P_m} , and N_{P_o} be the number of elements of A , P_m , and P_o
11 respectively. If there are no *mandatory-parameters* or *optional-parameters*, let N_{P_m}
12 and N_{P_o} be 0. Let S_b be the current set of local variable bindings.
 - 13 3) If N_A is smaller than N_{P_m} , raise a direct instance of the class **ArgumentError**.
 - 14 4) If the method does not have P_a and N_A is larger than the sum of N_{P_m} and N_{P_o} , raise
15 a direct instance of the class **ArgumentError**.
 - 16 5) Otherwise, for each argument A_i in A , in the same order in A , take the following steps:
 - 17 i) Let P_i be the *mandatory-parameter* or the *optional-parameter* whose position in
18 the L corresponds to the position of A_i in A .
 - 19 I) If such P_i exists, let n be the *mandatory-parameter* if P_i is a mandatory
20 parameter, or *optional-parameter-name* if P_i is an optional parameter. Create
21 a variable binding with name n and value A_i in S_b .
 - 22 II) If such P_i does not exist, i.e. if N_A is larger than the sum of N_{P_m} and N_{P_o} ,
23 and P_a exists:
 - 24 • Create a direct instance of the class **Array** X whose length is the number
25 of extra arguments.
 - 26 • Store each extra arguments into X , preserving the order in which they
27 occur in the list of arguments.
 - 28 • Let n be the *array-parameter-name* of P_a .
 - 29 • Create a variable binding with name n and value X in S_b .
 - 30 ii) If N_A is smaller than the sum of N_{P_m} and N_{P_o} :
 - 31 I) For each optional parameter P_{O_i} to which no argument corresponds, evaluate
32 the *default-parameter-expression* of P_{O_i} , and let V be the resulting value.

- 1 II) Let n be the *optional-parameter-name* of P_{O_i} .
- 2 III) Create a variable binding with name n and value V in S_b .
- 3 iii) If N_A is smaller than or equal to the sum of N_{P_m} and N_{P_o} , and P_a exists:
- 4 I) Create an empty direct instance of the class **Array** V .
- 5 II) Let n be the *array-parameter-name* of P_a .
- 6 III) Create a variable binding with name n and value V in S_b .
- 7 iv) If the *proc-parameter* of L is present, let D be the top of `[[block]]`.
- 8 I) If D is block-not-given, let V be **nil**.
- 9 II) Otherwise, invoke the method **new** on the class **Proc** with an empty list of
10 arguments and D as the block. Let V be the resulting value of the method
11 invocation.
- 12 III) Let n be the *proc-parameter-name* of *proc-parameter*.
- 13 IV) Create a variable binding with name n and value V in S_b .
- 14 i) Evaluate the *method-body* of V .
- 15 1) If the evaluation of the *method-body* is terminated by a *return-expression*:
- 16 i) If the *jump-argument* of the *return-expression* is present, let O be the value of the
17 *jump-argument*.
- 18 ii) Otherwise, let O be **nil**.
- 19 2) Otherwise, let O be the resulting value of the evaluation.
- 20 j) Restore the execution context by removing the elements from the tops of `[[class-module-list]]`,
21 `[[self]]`, `[[invoked-method-name]]`, `[[default-visibility]]`, `[[defined-method-name]]`, `[[block]]`, and
22 `[[local-variable-bindings]]`, even when an exception is raised and not handled in Step i.
- 23 k) The value of the method invocation is O .

24 The method invocation or the super expression (see Step d of §11.3.4) which corresponds to the
25 set of items on the tops of all the attributes of the execution context modified in Step g, except
26 `[[local-variable-bindings]]`, is called the **current method invocation**.

27 13.3.4 Method lookup

28 Method lookup is the process by which a binding of an instance method is resolved.

29 Given a method name M and a class or a module C which is initially searched for the binding
30 of the method, the method binding is resolved as follows:

- 1 a) If a method binding with name M exists in the set of bindings of instance methods of C ,
2 let B be that binding.
- 3 b) Otherwise, let L_m be the list of included modules of C . Search for a method binding with
4 name M in the set of bindings of instance methods of each module in L_m . Examine modules
5 in L_m in reverse order.
 - 6 1) If a binding is found, let B be this binding.
 - 7 2) Otherwise:
 - 8 i) If C does not have a direct superclass, the binding is considered not resolved.
 - 9 ii) Otherwise, replace C with the direct superclass of C , and continue processing from
10 Step a.
- 11 c) B is the resolved method binding.

12 **13.3.5 Method visibility**

13 **13.3.5.1 General description**

14 Methods are categorized into one of public, private, or protected methods according to the
15 conditions under which the method invocation is allowed. The attribute of a method which
16 determines these conditions is called the **visibility** of the method.

17 **13.3.5.2 Public methods**

18 A public method is a method whose visibility attribute is set to the public visibility.

19 A public method can be invoked on an object anywhere within a program.

20 **13.3.5.3 Private methods**

21 A private method is a method whose visibility attribute is set to the private visibility.

22 A private method can not be invoked with explicit receiver, i.e. method invocations of the
23 forms where *primary-expression* or *chained-method-invocation* occurs at the position which cor-
24 responds to the method receiver are not allowed, except for method invocations of the following
25 forms where the *primary-expression* is a *self-expression*.

- 26 • *single-method-assignment*
- 27 • *abbreviated-method-assignment*
- 28 • *single-indexing-assignment*
- 29 • *abbreviated-indexing-assignment*

30 **13.3.5.4 Protected methods**

31 A protected method is a method whose visibility attribute is set to the protected visibility.

1 A protected method can be invoked if and only if the following condition holds:

2 • Let M be an instance of the class `Module` in which the binding of the method exists.

3 M is included in the current self, or M is the class of the current self or one of its superclasses.

4 If M is an eigenclass, whether the method can be invoked or not may be determined in a
5 implementation-defined way.

6 **13.3.5.5 Visibility change**

7 The visibility of methods can be changed with built-in methods `public` (§15.2.2.4.38), `private`
8 (§15.2.2.4.36), and `protected` (§15.2.2.4.37), which are defined in the class `Module`.

9 **13.3.6 The alias statement**

10 **Syntax**

11 *alias-statement* ::
12 **alias** *new-name* *aliased-name*

13 *new-name* ::
14 *method-name*
15 | *symbol*

16 *aliased-name* ::
17 *method-name*
18 | *symbol*

19 **Semantics**

20 An *alias-statement* is evaluated as follows:

21 a) Evaluate the *new-name* as follows:

22 1) If the *new-name* is of the form *method-name*, let N be the *method-name*.

23 2) If the *new-name* is of the form *symbol*, evaluate it. Let N be the name of the resulting
24 instance of the class `Symbol`.

25 b) Evaluate the *aliased-name* as follows:

26 1) If *aliased-name* is of the form *method-name*, let A be the *method-name*.

27 2) If *aliased-name* is of the form *symbol*, evaluate it. Let A be the name of the resulting
28 instance of the class `Symbol`.

29 c) Let C be the current class or module.

- 1 d) Search for a method binding with name A , starting from C as described in §13.3.4.
- 2 e) If a binding is found and its value is not `undef`, let V be the value of the binding.
- 3 f) Otherwise, let S be a direct instance of the class `Symbol` with name A and raise a direct
4 instance of the class `NameError` which has S as its name property.
- 5 g) If a method binding with name N exists in the set of bindings of instance methods of the
6 current class or module, replace the value of the binding with V .
- 7 h) Otherwise, create a method binding with name N and value V in the set of bindings of
8 instance methods of the current class or module.
- 9 i) The value of *alias-statement* is `nil`.

10 13.3.7 The undef statement

11 Syntax

12 *undef-statement* ::
13 `undef undef-list`

14 *undef-list* ::
15 *method-name-or-symbol* (, *method-name-or-symbol*)*

16 *method-name-or-symbol* ::
17 *method-name*
18 | *symbol*

19 Semantics

20 An *undef-statement* is evaluated as follows:

- 21 a) For each *method-name-or-symbol* of the *undef-list*, take the following steps:
 - 22 1) Let C be the current class or module.
 - 23 2) If the *method-name-or-symbol* is of the form *method-name*, let N be the *method-name*.
24 Otherwise, evaluate the *symbol*. Let N be the name of the resulting instance of the
25 class `Symbol`.
 - 26 3) Search for a method binding with name N , starting from C as described in §13.3.4.
 - 27 4) If a binding is found and its value is not `undef`:
 - 28 i) If the binding is found in C , replace the value of the binding with `undef`.
 - 29 ii) Otherwise, create a method binding with name N and value `undef` in the set of
30 bindings of instance methods of C .

1 5) Otherwise, let *S* be a direct instance of the class `Symbol` with name *N* and raise a
2 direct instance of the class `NameError` which has *S* as its name property.

3 b) The value of *undef-statement* is `nil`.

4 **13.4 Eigenclasses**

5 **13.4.1 General description**

6 An eigenclass is a special class which is associated with a particular object. An eigenclass
7 modifies the behavior of an object when associated with it. When such an association is made,
8 the eigenclass is called the eigenclass of the object, and the object is called the primary associated
9 object of the eigenclass.

10 An object has at most one eigenclass. When an object is created, it shall not be associated with
11 any eigenclasses unless the object is an instance of the class `Class`. Eigenclasses are associated
12 with an object by an evaluation of a program construct such as a *singleton-method-definition*
13 or an *eigenclass-definition*. However, when an instance of the class `Class` is created, it shall
14 already have been associated with its eigenclass.

15 Normally, an eigenclass shall be associated with only its primary associated object; however, the
16 eigenclass of an instance of the class `Class` may be associated with some additional instances
17 of the class `Class` which are not the primary associated objects of any other eigenclasses, in an
18 implementation-defined way. Once associated, the primary associated object of an eigenclass
19 shall not be dissociated from its eigenclass; however the aforementioned additional associated
20 instances of the class `Class` are dissociated from their eigenclass when they become the primary
21 associated object of another eigenclass (see Step e of §13.4.2 and Step e of §13.4.3).

22 The direct superclass of an eigenclass is implementation-defined. However, an eigenclass shall
23 be a subclass of the class of the object with which it is associated.

24 NOTE 1 For example, the eigenclass of the class `Object` is a subclass of the class `Class` because the
25 class `Object` is a direct instance of the class `Class`. Therefore, the instance methods of the class `Class`
26 can be invoked on the class `Object`.

27 The eigenclass of a class which has a direct superclass shall satisfy the following condition:

- 28 • Let E_c be the eigenclass of a class *C*, and let *S* be the direct superclass of *C*, and let E_s be
29 the eigenclass of *S*. Then, E_c have E_s as one of its superclasses.

30 NOTE 2 This requirement enables classes to inherit singleton methods from its superclasses. For ex-
31 ample, the eigenclass of the class `File` has the eigenclass of the class `I0` as its superclass. Thereby, the
32 class `File` inherits the singleton method `open` of the class `I0`.

33 Although eigenclasses are instances of the class `Class`, they cannot create an instance of them-
34 selves. When the method `new` is invoked on an eigenclass, a direct instance of the class `TypeError`
35 shall be raised (see Step a of §15.2.3.3.3).

36 Whether an eigenclass can be a superclass of other classes is unspecified (see Step b-2-i of §13.2.2
37 and Step c of §15.2.3.3.1).

38 Whether an eigenclass can have class variables or not is implementation-defined.

1 13.4.2 Eigenclass definition

2 Syntax

3 *eigenclass-definition* ::
4 **class** << *expression* *separator* *eigenclass-body* **end**

5 *eigenclass-body* ::
6 *body-statement*

7 Semantics

8 An *eigenclass-definition* is evaluated as follows:

- 9 a) Evaluate the *expression*. Let O be the resulting value. A conforming processor may specify
10 the set of classes such that if O is an instance of one of the classes in the set, a direct
11 instance of the class **TypeError** is raised.
- 12 b) If O is one of **nil**, **true**, or **false**, let E be the class of O and go to Step f.
- 13 c) If O is not associated with an eigenclass, create a new eigenclass. Let E be the newly
14 created eigenclass, and associate O with E as its primary associated object.
- 15 d) If O is associated with an eigenclass as its primary associated object, let E be that eigenclass.
- 16 e) If O is associated with an eigenclass not as its primary associated object, dissociate O from
17 the eigenclass, and create a new eigenclass. Let E be the newly created eigenclass, and
18 associate O with E as its primary associated object.
- 19 f) Modify the execution context as follows:
- 20 1) Create a new list which consists of the same elements as the list at the top of \llbracket class-
21 module-list \rrbracket and add E to the head of the newly created list. Push the list onto
22 \llbracket class-module-list \rrbracket .
- 23 2) Push E onto \llbracket self \rrbracket .
- 24 3) Push the public visibility onto \llbracket default-visibility \rrbracket .
- 25 4) Push an empty set of bindings onto \llbracket local-variable-bindings \rrbracket .
- 26 g) Evaluate the *eigenclass-body*. The value of the *eigenclass-definition* is the value of the
27 *eigenclass-body*.
- 28 h) Restore the execution context by removing the elements from the tops of \llbracket class-module-
29 list \rrbracket , \llbracket self \rrbracket , \llbracket default-visibility \rrbracket , and \llbracket local-variable-bindings \rrbracket , even when an exception is
30 raised and not handled during Step g.

1 13.4.3 Singleton method definition

2 Syntax

```
3 singleton-method-definition ::  
4     def singleton ( . | :: ) method-name [no line-terminator here]  
5         method-parameter-part method-body end  
  
6 singleton ::  
7     variable-reference  
8     | ( expression )
```

9 Semantics

10 A *singleton-method-definition* is evaluated as follows:

- 11 a) Evaluate the *singleton*. Let S be the resulting value.
- 12 b) If S is one of **nil**, **true**, or **false**, let E be the class of O and go to Step f.
- 13 c) If S is not associated with an eigenclass, create a new eigenclass. Let E be the newly
14 created eigenclass, and associate S with E as its primary associated object.
- 15 d) If S is associated with an eigenclass as its primary associated object, let E be that eigenclass.
- 16 e) If S is associated with an eigenclass not as its primary associated object, dissociate S from
17 the eigenclass, and create a new eigenclass. Let E be the newly created eigenclass, and
18 associate S with E as its primary associated object.
- 19 f) Let N be the *method-name*.
- 20 g) Create a method U defined by the *method-definition*. Initialize the attributes of U as
21 follows:
 - 22 • The class module list is the element at the top of \llbracket class-module-list \rrbracket .
 - 23 • The defined name is N .
 - 24 • The visibility is the public visibility.
- 25 h) If a method binding with name N exists in the set of bindings of instance methods of E ,
26 let V be the value of that binding.
 - 27 1) If V is undef, the evaluation of the *singleton-method-definition* is implementation-
28 defined.
 - 29 2) Replace the value of the binding with U .
- 30 i) Otherwise, create a method binding with name N and value U in the set of bindings of
31 instance methods of E .

1 j) The value of the *singleton-method-definition* is implementation-defined.

2 14 Exceptions

3 14.1 General description

4 If an instance of the class `Exception` is raised, the current evaluation process stops, and the
5 evaluation process is transferred to a program construct that can handle this exception.

6 14.2 Cause of exceptions

7 An exception is raised when:

- 8 • the method `raise` (see §15.3.1.2.13) is invoked.
- 9 • an exceptional condition occurs as described in various parts of this document.

10 Only instances of the class `Exception` shall be raised.

11 14.3 Exception handling

12 Exceptions are handled by a *body-statement*, an *assignment-with-rescue-modifier*, or a *rescue-*
13 *modifier-statement*. These program constructs are called **exception handlers**. When an ex-
14 ception handler is handling an exception, the exception being handled is called the **current**
15 **exception**.

16 When an exception is raised, it is handled by an exception handler. This exception handler is
17 determined as follows:

- 18 a) Let S be the innermost local variable scope which lexically encloses the location where
19 the exception is raised, and which corresponds to one of a *program*, a *method-definition*, a
20 *singleton-method-definition*, or a *block*.
- 21 b) Test each exception handler in S which lexically encloses the location where the exception
22 is raised from the innermost to the outermost.
 - 23 • An *assignment-with-rescue-modifier* is considered to handle the exception if the ex-
24 ception is an instance of the class `StandardError` (see §11.4.2.5), except when the
25 exception is raised in its *operator-expression*₂. In this case, *assignment-with-rescue-*
26 *modifier* does not handle the exception.
 - 27 • A *rescue-modifier-statement* is considered to handle the exception if the exception is an
28 instance of the class `StandardError` (see §12.7), except when the exception is raised
29 in its *fallback-statement-of-rescue-modifier-statement*. In this case, *rescue-modifier-*
30 *statement* does not handle the exception.
 - 31 • A *body-statement* is considered to handle the exception if one of its *rescue-clauses* is
32 considered to handle the exception (see §11.5.2.4.1), except when the exception is raised
33 in one of its *rescue-clauses*, *else-clauses*, or *ensure-clauses*. In this case, *body-statement*
34 does not handle the exception. If an *ensure-clause* of a *body-statement* is present, it is
35 evaluated even if the handler does not handle the exception (see §11.5.2.4.1).

- 1 c) If an exception handler which can handle the exception is found in *S*, terminate the search
2 for the exception handler. Continue evaluating the program as defined for the relevant
3 construct (see §11.5.2.4.1, §11.4.2.5, and §12.7).
- 4 d) If none of the exception handlers in *S* can handle the exception:
- 5 1) If *S* corresponds to a *method-definition* or a *singleton-method-definition*, terminate Step
6 h or Step i of §13.3.3, and take Step j of the current method invocation (see §13.3.3).
7 Continue the search from Step a, under the assumption that the exception is raised at
8 the location where the method is invoked.
- 9 2) If *S* corresponds to a *block*, terminate the evaluation of the current *block*. Continue the
10 search from Step a, under the assumption that the exception is raised at the location
11 where the block is called.
- 12 3) Otherwise, terminate the evaluation of the *program*.

13 15 Built-in classes and modules

14 15.1 General description

15 Built-in classes and modules are specified in this clause. Those classes and modules are defined
16 in the class `Object` as constants (see §15.2.1.4).

17 Each built-in class or module is specified by describing its attribute values, as described in §13.1
18 and §13.2.

19 When a subclass specifying a built-in class or module contains a subclass titled “Included
20 modules”, the built-in class or module shall include the modules listed in that subclass in the
21 order of that listing.

22 Each subclass in the subclass titled “Singleton methods” with the title of the form *C.m*
23 specifies the singleton method *m* of the class *C*.

24 Each subclass in the subclass titled “Instance methods” with the title of the form *C#m*
25 specifies the instance method *m* of the class *C*.

26 The parameter specification of a method is described in the form of *method-parameter-part* (see
27 §13.3.1).

28 EXAMPLE 1 The following example defines the parameter specification of a method `sample`.

```
29 sample( arg1, arg2, opt=expr, *ary, &blk )
```

30 For a singleton method, the method name is prefixed by the name of the class or the module,
31 and a dot (`.`).

32 EXAMPLE 2 The following example defines the parameter specification of a singleton method `sample`
33 of a class `SampleClass`:

```
34 SampleClass.sample( arg1, arg2, opt=expr, *ary, &blk )
```

1 Next to the parameter specification, the visibility and the behavior of the method are specified.

2 The visibility, which is any one of public, protected or private, is specified after the label named
3 “Visibility:”.

4 The behavior, which is the steps which shall be taken while evaluating the *method-body* of the
5 method (see Step i of §13.3.3), is specified after the label named “Behavior:”.

6 In these steps, a reference to the name of an argument in the parameter specification is considered
7 to be the object bound to the local variables of the same name. The phrase “call the *block* with
8 *X* as the argument” indicates that the block corresponding to the block parameter *block* shall
9 be called as described in §11.3.3 with *X* as the argument to the block call. The phrase “return
10 *X*” indicates that the evaluation of the *method-body* shall be terminated at that point, and *X*
11 shall be the value of the *method-body*. The phrase “the name designated by *N*” means the result
12 of the following steps:

- 13 a) If *N* is an instance of the class `Symbol`, the name of *N*.
- 14 b) If *N* is an instance of the class `String`, the content of *N*.
- 15 c) Otherwise, the behavior of the method is unspecified.

16 The class module list of an instance method of a built-in class or module shall be a list which
17 consists of two elements: the first is the built-in class or module; the second is the class `Object`.
18 The class module list of a singleton method of a built-in class or module shall be a list which
19 consists of two elements: the first is the eigenclass of the built-in class or module; the second is
20 the class `Object`.

21 A conforming processor may provide additional attributes and/or values: a specific initial value
22 for a predefined attribute whose initial value is not specified in this document, constants, single-
23 ton methods, instance methods, additional optional parameters or array parameters for methods
24 specified in this document, and additional inclusion of modules into built-in classes/modules.

25 **15.2 Built-in classes**

26 **15.2.1 Object**

27 **15.2.1.1 General description**

28 The class `Object` is an implicit direct superclass for other classes; that is, if the direct superclass
29 of a class is not specified explicitly in the class definition, the direct superclass of the class is
30 the class `Object` (see §13.2.2).

31 All built-in classes and modules can be referenced through constants of the class `Object`.

32 **15.2.1.2 Direct superclass**

33 The class `Object` does not have a direct superclass, or a conforming processor may define a
34 finite sequence of superclasses of the class `Object`.

1 15.2.1.3 Included modules

2 The following module is included in the class `Object`.

- 3 • `Kernel`

4 15.2.1.4 Constants

5 The following constants are defined in the class `Object`.

6 **STDIN:** An implementation-defined instance of the class `IO`, which shall be readable.

7 **STDOUT:** An implementation-defined instance of the class `IO`, which shall be writable.

8 **STDERR:** An implementation-defined instance of the class `IO`, which shall be writable.

9 Besides, every built-in class or module, including the class `Object` itself, shall be defined in the
10 class `Object` as a constant, whose name is the name of the class or module, and whose value is
11 the class or module.

12 15.2.1.5 Instance methods

13 15.2.1.5.1 `Object#initialize`

14 `initialize(*args)`

15 **Visibility:** private

16 **Behavior:** The method `initialize` is the default object initialization method, which is
17 invoked when an instance is created (see §13.2.4). It returns an implementation-defined
18 value.

19 15.2.2 Module

20 15.2.2.1 General description

21 All modules are instances of the class `Module`. Therefore, behaviors defined in the class `Module`
22 are shared by all modules.

23 The binary relation on the instances of the class `Module` denoted $A \sqsubset B$ is defined as follows:

- 24 • B is a module and B is included in A (see §13.1.3) or
- 25 • both A and B are classes and B is a superclass of A .

26 15.2.2.2 Direct superclass

27 The class `Object`

1 **15.2.2.3 Singleton methods**

2 **15.2.2.3.1 Module.constants**

3 `Module.constants`

4 **Visibility:** public

5 **Behavior:**

- 6 a) Create an empty direct instance of the class `Array`. Let A be the instance.
- 7 b) Let C be the current class or module. Let L be the list which consists of the same
8 elements as the list at the second element from the top of `[[class-module-list]]`, except
9 the last element, which is the class `Object`.
- 10 Let CS be the set of classes which consists of C and all the superclasses of C except
11 the class `Object`, but when C is the class `Object`, it shall be included in CS . Let MS
12 be the set of modules which consists of all the modules in the included module list of
13 all classes in CS . Let CM be the union of L , CS and MS .
- 14 c) For each class or module c in CM , and for each name N of a constant defined in c ,
15 take the following steps:
- 16 1) Let S be either a new direct instance of the class `String` whose content is N or a
17 direct instance of the class `Symbol` whose name is N . Which of these classes of
18 instance is chosen as the value of S is implementation-defined.
- 19 2) Unless A contains the element of the same name as S , when S is an instance of the
20 class `Symbol`, or the same content as S , when S is an instance of the class `String`,
21 append S to A .
- 22 d) Return A .

23 **15.2.2.3.2 Module.nesting**

24 `Module.nesting`

25 **Visibility:** public

26 **Behavior:** The method returns a new direct instance of the class `Array` which contains all
27 but the last element of the list at the second element from the top of the `[[class-module-list]]`
28 in the same order.

29 **15.2.2.4 Instance methods**

30 **15.2.2.4.1 Module#<**

1 <(*other*)

2 **Visibility:** public

3 **Behavior:** Let A be the *other*. Let R be the receiver of the method.

4 a) If A is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.

5 b) If A and R is the same object, return **false**.

6 c) If $R \sqsubset A$, return **true**.

7 d) If $A \sqsubset R$, return **false**.

8 e) Otherwise, return **nil**.

9 **15.2.2.4.2 Module#<=**

10 <=(*other*)

11 **Visibility:** public

12 **Behavior:**

13 a) If the *other* and the receiver are the same object, return **true**.

14 b) Otherwise, the behavior is the same as the method < (see §15.2.2.4.1).

15 **15.2.2.4.3 Module#<=>**

16 <=>(*other*)

17 **Visibility:** public

18 **Behavior:** Let A be the *other*. Let R be the receiver of the method.

19 a) If A is not an instance of the class `Module`, return **nil**.

20 b) If A and R is the same object, return an instance of the class `Integer` whose value is
21 0.

22 c) If $R \sqsubset A$, return an instance of the class `Integer` whose value is -1 .

23 d) If $A \sqsubset R$, return an instance of the class `Integer` whose value is 1.

24 e) Otherwise, return **nil**.

1 **15.2.2.4.4** `Module#==`

2 `==(other)`

3 **Visibility:** public

4 **Behavior:** Same as the method `==` of the module `Kernel` (see §15.3.1.3.1).

5 **15.2.2.4.5** `Module#===`

6 `===(object)`

7 **Visibility:** public

8 **Behavior:** The method behaves as if the method `kind_of?` were invoked on the *object*
9 with the receiver as the only argument (see §15.3.1.3.26).

10 **15.2.2.4.6** `Module#>`

11 `>(other)`

12 **Visibility:** public

13 **Behavior:** Let *A* be the *other*. Let *R* be the receiver of the method.

- 14 a) If *A* is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.
15 b) If *A* and *R* is the same object, return **false**.
16 c) If $R \sqsubset A$, return **false**.
17 d) If $A \sqsubset R$, return **true**.
18 e) Otherwise, return **nil**.

19 **15.2.2.4.7** `Module#>=`

20 `>=(other)`

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the *other* and the receiver are the same object, return **true**.
24 b) Otherwise, the behavior is the same as the method `>` (see §15.2.2.4.6).

1 15.2.2.4.8 Module#alias_method

2 alias_method(*new_name*, *aliased_name*)

3 **Visibility:** private

4 **Behavior:** Let *C* be the receiver of the method.

- 5 a) Let *N* be the name designated by the *new_name*. Let *A* be the name designated by
6 the *aliased_name*.
- 7 b) Take Step d through h of §13.3.6, assuming that *A*, *C*, and *N* in §13.3.6 to be *A*, *C*,
8 and *N* in the above steps.
- 9 c) Return *C*.

10 15.2.2.4.9 Module#ancestors

11 ancestors

12 **Visibility:** public

13 **Behavior:**

- 14 a) Create an empty direct instance of the class `Array` *A*.
- 15 b) Let *C* be the receiver of the method.
- 16 c) If *C* is not an eigenclass, append *C* to *A*.
- 17 d) Append each element of the included module list of *C*, in the order in the list, to *A*.
- 18 e) If *C* is a class, replace *C* with the direct superclass of the current *C*.
- 19 f) If *C* is not `nil`, repeat from Step c.
- 20 g) Return *A*.

21 15.2.2.4.10 Module#append_features

22 append_features(*module*)

23 **Visibility:** private

24 **Behavior:** Let *L1* and *L2* be the included modules list of the receiver and the *module*
25 respectively.

- 26 a) If the *module* and the receiver is the same object, the behavior is unspecified.

- 1 b) If the receiver is an element of *L2*, the behavior is implementation-defined.
- 2 c) Otherwise, for each module *M* in *L1*, in the same order in *L1*, take the following steps:
 - 3 1) If *M* and the *module* are the same object, the behavior is unspecified.
 - 4 2) If *M* is not in *L2*, append *M* to the end of *L2*.
- 5 d) Append the receiver to *L2*.
- 6 e) Return the receiver.

7 15.2.2.4.11 Module#attr

```
8     attr( symbol, writable=false )
```

9 **Visibility:** private

10 **Behavior:** Let *C* be the method receiver.

- 11 a) If the *symbol* is not an instance of the class `Symbol`, the behavior is unspecified.
- 12 b) Let *N* be the name of the *symbol*.
- 13 c) If *N* is not of the form *local-variable-identifier* or *constant-identifier*, raise a direct

14 instance of the class `NameError` which has the *symbol* as its name property.
- 15 d) Define an instance method in *C* as if by evaluating the following method definition at

16 the location of the invocation. In the following method definition, *N* is *N*, and *@N* is the

17 name which is *N* prefixed by “@”.

```
18                 def N
19                     @N
20                 end
```

- 22 e) If the *writable* is **true**, define an instance method in *C* as if by evaluating the following

23 method definition at the location of the invocation. In the following method definition,

24 *N=* is the name *N* postfixed by =, and *@N* is the name which is *N* prefixed by “@”. The

25 choice of the parameter name is arbitrary, and *val* is chosen only for the expository

26 purpose.

```
27                 def N=(val)
28                     @N = val
29                 end
```

- 31 f) Return **nil**.

32 15.2.2.4.12 Module#attr_accessor

1 `attr_accessor(*symbol_list)`

2 **Visibility:** private

3 **Behavior:**

4 a) For each element *S* of the *symbol_list*, invoke the method `attr` with *S* as the first
5 argument and **true** as the second argument (see §15.2.2.4.11).

6 b) Return **nil**.

7 **15.2.2.4.13 Module#attr_reader**

8 `attr_reader(*symbol_list)`

9 **Visibility:** private

10 **Behavior:**

11 a) For each element *S* of the *symbol_list*, invoke the method `attr` with *S* as the first
12 argument and **false** as the second argument (see §15.2.2.4.11).

13 b) Return **nil**.

14 **15.2.2.4.14 Module#attr_writer**

15 `attr_writer(*symbol_list)`

16 **Visibility:** private

17 **Behavior:**

18 a) For each element *S* of the *symbol_list*, invoke the method `attr` with *S* as the first
19 argument and **true** as the second argument, but skip Step d (see §15.2.2.4.11).

20 b) Return **nil**.

21 **15.2.2.4.15 Module#class_eval**

22 `class_eval(string = nil, &block)`

23 **Visibility:** public

24 **Behavior:**

- 1 a) Let M be the receiver.
- 2 b) If the *block* is given:
- 3 1) If the *string* is given, raise a direct instance of the class **ArgumentError**.
- 4 2) Call the *block* with implementation-defined arguments as described in §11.3.3, and
5 let V be the resulting value. A conforming processor shall modify the execution
6 context just before Step d of §11.3.3 as follows:
- 7 • Create a new list which has the same members as those of the list at the top
8 of `[[class-module-list]]`, and add M to the head of the newly created list. Push
9 the list onto `[[class-module-list]]`.
- 10 • Push the receiver onto `[[self]]`.
- 11 • Push the public visibility onto `[[default-visibility]]`.
- 12 In Step d and e of §11.3.3, a conforming processor shall ignore M which is added
13 to the head of the top of `[[class-module-list]]` as described above, except when
14 referring to the current class or module in a *method-definition* (see §13.3.1), an
15 *alias-statement* (see §13.3.6), or an *undef-statement* (see §13.3.7).
- 16 3) Return V .
- 17 c) If the *block* is not given:
- 18 1) If the *string* is not an instance of the class **String**, the behavior is unspecified.
- 19 2) Modify the execution context as follows:
- 20 • Create a new list which has the same members as those of the list at the top
21 of `[[class-module-list]]`, and add M to the head of the newly created list. Push
22 the list onto `[[class-module-list]]`.
- 23 • Push the receiver onto `[[self]]`.
- 24 • Push the public visibility onto `[[default-visibility]]`.
- 25 3) Parse the content of the *string* as a *program* (see §10.1). If it fails, raise a direct
26 instance of the class **SyntaxError**.
- 27 4) Evaluate the *program*. Let V be the resulting value of the evaluation.
- 28 5) Restore the execution context by removing the elements from the tops of `[[class-`
29 `module-list]]`, `[[self]]`, and `[[default-visibility]]`, even when an exception is raised and
30 not handled in Step c-3 or c-4.
- 31 6) Return V .

32 In Step c-4, the *string* is evaluated under the new local variable scope in which references
33 to *local-variable-identifiers* are resolved in the same way as in scopes created by *blocks* (see
34 §9.2).

1 **15.2.2.4.16** `Module#class_variable_defined?`

2 `class_variable_defined?(symbol)`

3 **Visibility:** public

4 **Behavior:** Let *C* be the receiver of the method.

- 5 a) Let *N* be the name designated by the *symbol*.
- 6 b) If *N* is not of the form *class-variable-identifier*, raise a direct instance of the class
7 `NameError` which has the *symbol* as its name property.
- 8 c) Search for a binding of the class variable with name *N* by taking Step b through d of
9 §11.5.4.5, assuming that *C* and *N* in §11.5.4.5 to be *C* and *N* in the above steps.
- 10 d) If a binding is found, return **true**.
- 11 e) Otherwise, return **false**.

12 **15.2.2.4.17** `Module#class_variable_get`

13 `class_variable_get(symbol)`

14 **Visibility:** private

15 **Behavior:** Let *C* be the receiver of the method.

- 16 a) Let *N* be the name designated by the *symbol*.
- 17 b) If *N* is not of the form *class-variable-identifier*, raise a direct instance of the class
18 `NameError` which has the *symbol* as its name property.
- 19 c) Search for a binding of the class variable with name *N* by taking Step b through d of
20 §11.5.4.5, assuming that *C* and *N* in §11.5.4.5 to be *C* and *N* in the above steps.
- 21 d) If a binding is found, return the value of the binding.
- 22 e) Otherwise, raise a direct instance of the class `NameError` which has the *symbol* as its
23 name property.

24 **15.2.2.4.18** `Module#class_variable_set`

25 `class_variable_set(symbol, obj)`

26 **Visibility:** private

- 1 **Behavior:** Let *C* be the receiver of the method.
- 2 a) Let *N* be the name designated by the *symbol*.
- 3 b) If *N* is not of the form *class-variable-identifier*, raise a direct instance of the class
4 **NameError** which has the *symbol* as its name property.
- 5 c) Search for a binding of the class variable with name *N* by taking Step b through d of
6 §11.5.4.5, assuming that *C* and *N* in §11.5.4.5 to be *C* and *N* in the above steps.
- 7 d) If a binding is found, replace the value of the binding with the *obj*.
- 8 e) Otherwise, create a variable binding with name *N* and value *obj* in the set of bindings
9 of class variables of *C*.
- 10 f) Return *obj*.

11 **15.2.2.4.19 Module#class_variables**

12 **class_variables**

13 **Visibility:** public

14 **Behavior:** The method returns a new direct instance of the class **Array** which consists
15 of names of all class variables of the receiver. These names are represented by direct
16 instances of either the class **String** or the class **Symbol**. Which of those classes is chosen is
17 implementation-defined.

18 **15.2.2.4.20 Module#const_defined?**

19 **const_defined?(symbol)**

20 **Visibility:** public

21 **Behavior:**

- 22 a) Let *C* be the receiver of the method.
- 23 b) Let *N* be the name designated by the *symbol*.
- 24 c) If *N* is not of the form *constant-identifier*, raise a direct instance of the class **NameError**
25 which has the *symbol* as its name property.
- 26 d) If a binding with name *N* exists in the set of bindings of constants of *C*, return **true**.
- 27 e) Otherwise, return **false**.

28 **15.2.2.4.21 Module#const_get**

1 `const_get(symbol)`

2 **Visibility:** public

3 **Behavior:**

- 4 a) Let N be the name designated by the *symbol*.
- 5 b) If N is not of the form *constant-identifier*, raise a direct instance of the class `NameError`
6 which has the *symbol* as its name property.
- 7 c) Search for a binding of a constant with name N from Step e of §11.5.4.2, assuming
8 that C in §11.5.4.2 to be the receiver of the method.
- 9 d) If a binding is found, return the value of the binding.
- 10 e) Otherwise, return the value of the invocation of the method `const_missing` (See Step
11 e-1-i of §11.5.4.2).

12 **15.2.2.4.22 Module#const_missing**

13 `const_missing(symbol)`

14 **Visibility:** public

15 **Behavior:** The method `const_missing` is invoked when a binding of a constant does not
16 exist on a constant reference (see §11.5.4.2).

17 When the method is invoked, take the following steps:

- 18 a) Take Step a through c of §15.2.2.4.20.
- 19 b) Raise a direct instance of the class `NameError` which has the *symbol* as its name prop-
20 erty.

21 **15.2.2.4.23 Module#const_set**

22 `const_set(symbol, obj)`

23 **Visibility:** public

24 **Behavior:** Let C be the receiver of the method.

- 25 a) Let N be the name designated by the *symbol*.
- 26 b) If N is not of the form *constant-identifier*, raise a direct instance of the class `NameError`
27 which has the *symbol* as its name property.

- 1 c) If a binding with name *N* exists in the set of bindings of constants of *C*, replace the
2 value of the binding with the *obj*.
- 3 d) Otherwise, create a variable binding with *N* and value *obj* in the set of bindings of
4 constants of *C*.
- 5 e) Return the *obj*.

6 15.2.2.4.24 `Module#constants`

7 `constants`

8 **Visibility:** public

9 **Behavior:** The method returns a new direct instance of the class `Array` which consists
10 of names of all constants defined in the receiver. These names are represented by direct
11 instances of either the class `String` or the class `Symbol`. Which of those classes is chosen is
12 implementation-defined.

13 15.2.2.4.25 `Module#extend_object`

14 `extend_object(object)`

15 **Visibility:** private

16 **Behavior:** Let *S* be the eigenclass of the *object*. The method behaves as if by invoking
17 the method `append_features` on the receiver with *S* as the only argument.

18 15.2.2.4.26 `Module#extended`

19 `extended(object)`

20 **Visibility:** private

21 **Behavior:** The method returns `nil`.

22 15.2.2.4.27 `Module#include`

23 `include(*module_list)`

24 **Visibility:** private

25 **Behavior:** Let *C* be the receiver of the method.

- 26 a) For each element *A* of the *module_list*, in the reverse order in the *module_list*, take the
27 following steps:

- 1 1) If *A* is not an instance of the class `Module`, raise a direct instance of the class
2 `TypeError`.
- 3 2) If *A* is an instance of the class `Class`, raise a direct instance of the class `TypeError`.
- 4 3) Invoke the method `append_features` on *A* with *C* as the only argument.
- 5 4) Invoke the method `included` on *A* with *C* as the only argument.
- 6 b) Return *C*.

7 **15.2.2.4.28** `Module#include?`

8 `include?(module)`

9 **Visibility:** public

10 **Behavior:** Let *C* be the receiver of the method.

- 11 a) If the *module* is not an instance of the class `Module`, raise a direct instance of the class
12 `TypeError`.
- 13 b) If the *module* is an element of the included module list of *C*, return **true**.
- 14 c) Otherwise, if *C* is an instance of the class `Class`, and if the *module* is an element of
15 the included module list of one of the superclasses of *C*, then return **true**.
- 16 d) Otherwise, return **false**.

17 **15.2.2.4.29** `Module#included`

18 `included(module)`

19 **Visibility:** private

20 **Behavior:** The method returns **nil**.

21 **15.2.2.4.30** `Module#included_modules`

22 `included_modules`

23 **Visibility:** public

24 **Behavior:** Let *C* be the receiver of the method.

- 25 a) Create an empty direct instance of the class `Array` *A*.

- 1 b) Append each element of the included module list of C , in the reverse order, to A .
- 2 c) If C is an instance of the class `Class`, and if C has a direct superclass, then replace C
- 3 with the direct superclass of the current C , and repeat from Step b.
- 4 d) Otherwise, return A .

5 **15.2.2.4.31 Module#initialize**

6 `initialize(&block)`

7 **Visibility:** private

8 **Behavior:**

- 9 a) If the *block* is given, take Step b of the method `class_eval` of the class `Module` (see
- 10 §15.2.2.4.15), assuming that *block* in §15.2.2.4.15 to be the *block* given to this method.
- 11 b) Return an implementation-defined value.

12 **15.2.2.4.32 Module#initialize_copy**

13 `initialize_copy(original)`

14 **Visibility:** private

15 **Behavior:**

- 16 a) Invoke the instance method `initialize_copy` defined in the module `Kernel` on the
- 17 receiver with the *original* as the argument.
- 18 b) If the receiver is associated with an eigenclass, let E_o be the eigenclass, and take the
- 19 following steps:
 - 20 1) Create an eigenclass whose direct superclass is the direct superclass of E_o . Let E_n
 - 21 be the eigenclass.
 - 22 2) For each binding B_{v1} of the constants of E_o , create a variable binding with the
 - 23 same name and value as B_{v1} in the set of bindings of constants of E_n .
 - 24 3) For each binding B_{v2} of the class variables of E_o , create a variable binding with
 - 25 the same name and value as B_{v2} in the set of bindings of class variables of E_n .
 - 26 4) For each binding B_m of the instance methods of E_o , create a method binding with
 - 27 the same name and value as B_m in the set of bindings of instance methods of E_n .
 - 28 5) Associate the receiver with E_n .
- 29 c) If the receiver is an instance of the class `Class`:

- 1) If the *original* has a direct superclass, set the direct superclass of the receiver to the direct superclass of the *original*.
- 2) Otherwise, the behavior is unspecified.
- d) Append each element of the included module list of the *original*, in the same order, to the included module list of the receiver.
- e) For each binding B_{v3} of the constants of the *original*, create a variable binding with the same name and value as B_{v3} in the set of bindings of constants of the receiver.
- f) For each binding B_{v4} of the class variables of the *original*, create a variable binding with the same name and value as B_{v4} in the set of bindings of class variables of the receiver.
- g) For each binding B_{m2} of the instance methods of the *original*, create a method binding with the same name and value as B_{m2} in the set of bindings of instance methods of the receiver.
- h) Return an implementation-defined value.

15.2.2.4.33 Module#instance_methods

```
instance_methods( include_super=true )
```

Visibility: public

Behavior: Let C be the receiver of the method.

- a) Create an empty direct instance of the class `Array` A .
- b) Let I be the set of bindings of instance methods of C . For each binding B of I , let N be the name of B , and let V be the value of B , and take the following steps:
 - 1) If V is undef, or the visibility of V is private, skip the next two steps.
 - 2) Let S be either a new direct instance of the class `String` whose content is N or a direct instance of the class `Symbol` whose name is N . Which of these classes of instance is chosen as the value of S is implementation-defined.
 - 3) Unless A contains the element of the same name (if S is an instance of the class `Symbol`) or the same content (if S is an instance of the class `String`) as S , append S to A .
- c) If the *include_super* is a trueish value:
 - 1) For each module M in included module list of C , take Step b, assuming that C in that step to be M .
 - 2) If C does not have a direct superclass, return A .

- 1 3) Replace *C* with the direct superclass of *C*.
2 4) Repeat from Step b.
3 d) Return *A*.

4 **15.2.2.4.34** `Module#method_defined?`

5 `method_defined?(symbol)`

6 **Visibility:** public

7 **Behavior:** Let *C* be the receiver of the method.

- 8 a) Let *N* be the name designated by the *symbol*.
9 b) Search for a binding of an instance method named *N* starting from *C* as described in
10 §13.3.4.
11 c) If a binding is found and its value is not `undef`, return **true**.
12 d) Otherwise, return **false**.

13 **15.2.2.4.35** `Module#module_eval`

14 `module_eval(string = nil, &block)`

15 **Visibility:** public

16 **Behavior:** Same as the method `class_eval` (see §15.2.2.4.15).

17 **15.2.2.4.36** `Module#private`

18 `private(*symbol-list)`

19 **Visibility:** private

20 **Behavior:** Same as the method `public` (see §15.2.2.4.38), except that the method changes
21 current visibility or visibilities of methods corresponding to each element of the *symbol-list*
22 to private.

23 **15.2.2.4.37** `Module#protected`

1 protected(*symbol-list)

2 **Visibility:** private

3 **Behavior:** Same as the method `public` (see §15.2.2.4.38), except that the method changes
4 current visibility or visibilities of methods corresponding to each element of the *symbol-list*
5 to `protected`.

6 **15.2.2.4.38 Module#public**

7 public(*symbol-list)

8 **Visibility:** private

9 **Behavior:** Let *C* be the receiver of the method.

10 a) If the length of *symbol-list* is 0, change the current visibility to `public` and return *C*.

11 b) Otherwise, for each element *S* of the *symbol-list*, take the following steps:

12 1) Let *N* be the name designated by *S*.

13 2) Search for a method binding with name *N* starting from *C* as described in §13.3.4.

14 3) If a binding is found and its value is not `undef`, let *V* the value of the binding.

15 4) Otherwise, raise a direct instance of the class `NameError` which has *S* as its name
16 property.

17 5) If *C* is the class or module in which the binding is found, change the visibility of
18 *V* to the `public` visibility.

19 6) Otherwise, define an instance method in *C* as if by evaluating the following method
20 definition. In the definition, *N* is *N*. The choice of the parameter name is arbitrary,
21 and `args` is chosen only for the expository purpose.

```
22                   def N(*args)
23                    super
24                   end
```

25

26 The attributes of the method created by the above definition are initialized as
27 follows:

28 i) The class module list is the element at the top of `[[class-module-list]]`.

29 ii) The defined name is the defined name of *V*.

30 iii) The visibility is the `public` visibility.

1 c) Return C .

2 15.2.2.4.39 Module#remove_class_variable

3 `remove_class_variable(symbol)`

4 **Visibility:** private

5 **Behavior:** Let C be the receiver of the method.

6 a) Let N be the name designated by the *symbol*.

7 b) If N is not of the form *class-variable-identifier*, raise a direct instance of the class
8 `NameError` which has the *symbol* as its name property.

9 c) If a binding with name N exists in the set of bindings of class variables of C , let V be
10 the value of the binding.

11 1) Remove the binding from the set of bindings of class variables of C .

12 2) Return V .

13 d) Otherwise, raise a direct instance of the class `NameError` which has the *symbol* as its
14 name property.

15 15.2.2.4.40 Module#remove_const

16 `remove_const(symbol)`

17 **Visibility:** private

18 **Behavior:** Let C be the receiver of the method.

19 a) Let N be the name designated by the *symbol*.

20 b) If N is not of the form *constant-identifier*, raise a direct instance of the class `NameError`
21 which has the *symbol* as its name property.

22 c) If a binding with name N exists in the set of bindings of constants of C , let V be the
23 value of the binding.

24 1) Remove the binding from the set of bindings of constants of C .

25 2) Return V .

26 d) Otherwise, raise a direct instance of the class `NameError` which has the *symbol* as its
27 name property.

1 **15.2.2.4.41 Module#remove_method**

2 `remove_method(*symbol_list)`

3 **Visibility:** private

4 **Behavior:** Let *C* be the receiver of the method.

5 a) For each element *S* of the *symbol_list*, in the order in the list, take following steps:

6 1) Let *N* be the name designated by *S*.

7 2) If a binding with name *N* exists in the set of bindings of instance methods of *C*,
8 remove the binding from the set.

9 3) Otherwise, raise a direct instance of the class `NameError` which has *S* as its name
10 property. In this case, the remaining elements of the *symbol_list* are not processed.

11 b) Return *C*.

12 **15.2.2.4.42 Module#undef_method**

13 `undef_method(*symbol_list)`

14 **Visibility:** private

15 **Behavior:** Let *C* be the receiver of the method.

16 a) For each element *S* of the *symbol_list*, in the order in the list, take following steps:

17 1) Let *N* be the name designated by *S*.

18 2) Take Step a-3 and a-4 of §13.3.7, assuming that *C* in §13.3.7 to be *C* in the above
19 steps.

20 b) Return *C*.

21 **15.2.3 Class**

22 **15.2.3.1 General description**

23 All classes are instances of the class `Class`. Therefore, behaviors defined in the class `Class` are
24 shared by all classes.

25 A conforming processor shall undefine the instance methods `append_features` and `extend_object`
26 of the class `Class`, as if by invoking the method `undef_method` on the class `Class` with instances
27 of the class `Symbol` whoses names are “append_features” and “extend_object” as the arguments
28 (see §15.2.2.4.42).

1 15.2.3.2 Direct superclass

2 The class `Module`

3 15.2.3.3 Instance methods

4 15.2.3.3.1 `Class#initialize`

5 `initialize(superclass=Object, &block)`

6 **Visibility:** private

7 **Behavior:**

- 8 a) If the receiver has its direct superclass, or is the root of the class inheritance tree, then
9 raise a direct instance of the class `TypeError`.
- 10 b) If the *superclass* is not an instance of the class `Class`, raise a direct instance of the
11 class `TypeError`.
- 12 c) If the *superclass* is an eigenclass or the class `Class`, the behavior is unspecified.
- 13 d) Set the direct superclass of the receiver to the *superclass*.
- 14 e) Create an eigenclass, and associate it with the receiver. The eigenclass shall have the
15 eigenclass of the *superclass* as one of its superclasses.
- 16 f) If the *block* is given, take Step b of the method `class_eval` of the class `Module` (see
17 §15.2.2.4.15), assuming that *block* in §15.2.2.4.15 to be the *block* given to this method.
- 18 g) Return an implementation-defined value.

19 15.2.3.3.2 `Class#initialize_copy`

20 `initialize_copy(original)`

21 **Visibility:** private

22 **Behavior:**

- 23 a) If the direct superclass of the receiver has already been set, or if the receiver is the root
24 of the class inheritance tree, then raise a direct instance of the class `TypeError`.
- 25 b) If the receiver is an eigenclass, raise a direct instance of the class `TypeError`.
- 26 c) Invoke the instance method `initialize_copy` defined in the class `Module` on the re-
27 ceiver with the *original* as the argument.
- 28 d) Return an implementation-defined value.

1 **15.2.3.3.3 Class#new**

2 `new(*args, &block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the receiver is an eigenclass, raise a direct instance of the class `TypeError`.
- 6 b) Create a direct instance of the receiver which has no bindings of instance variables.
7 Let *O* be the newly created instance.
- 8 c) Invoke the method `initialize` on *O* with all the elements of the *args* as arguments
9 and the *block* as the block.
- 10 d) Return *O*.

11 **15.2.3.3.4 Class#superclass**

12 `superclass`

13 **Visibility:** public

14 **Behavior:** Let *C* be the receiver of the methods.

- 15 a) If *C* is an eigenclass, return an implementation-defined value.
- 16 b) If *C* does not have a direct superclass, return `nil`.
- 17 c) Otherwise, return the direct superclass of *C*.

18 **15.2.4 NilClass**

19 **15.2.4.1 General description**

20 The class `NilClass` has only one instance, which is represented by the pseudo variable `nil`.

21 Instances of the class `NilClass` shall not be created by the method `new` of the class `NilClass`.
22 Therefore, a conforming processor shall undefine the singleton method `new` of the class `NilClass`,
23 as if by invoking the method `undef_method` on the eigenclass of the class `NilClass` with a direct
24 instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.4.42).

25 **15.2.4.2 Direct superclass**

26 The class `Object`

27 **15.2.4.3 Instance methods**

28 **15.2.4.3.1 NilClass#&**

1 &(*other*)

2 **Visibility:** public

3 **Behavior:** The method returns **false**.

4 **15.2.4.3.2 NilClass#^**

5 ^(*other*)

6 **Visibility:** public

7 **Behavior:**

8 a) If the *other* is a falseish value, return **false**.

9 b) Otherwise, return **true**.

10 **15.2.4.3.3 NilClass#nil?**

11 nil?

12 **Visibility:** public

13 **Behavior:** The method returns **true**.

14 **15.2.4.3.4 NilClass#|**

15 |(*other*)

16 **Visibility:** public

17 **Behavior:**

18 a) If the *other* is a falseish value, return **false**.

19 b) Otherwise, return **true**.

20 **15.2.5 TrueClass**

21 **15.2.5.1 General description**

22 The class **TrueClass** has only one instance, which is represented by the pseudo variable **true**.
23 **true** represents a logical trueish value.

1 Instances of the class `TrueClass` shall not be created by the method `new` of the class `TrueClass`.
2 Therefore, a conforming processor shall undefine the singleton method `new` of the class `TrueClass`,
3 as if by invoking the method `undef_method` on the eigenclass of the class `TrueClass` with a direct
4 instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.4.42).

5 **15.2.5.2 Direct superclass**

6 The class `Object`

7 **15.2.5.3 Instance methods**

8 **15.2.5.3.1 `TrueClass#&`**

9 `&(other)`

10 **Visibility:** public

11 **Behavior:**

12 a) If the *other* is a falseish value, return **false**.

13 b) Otherwise, return **true**.

14 **15.2.5.3.2 `TrueClass#^`**

15 `^(other)`

16 **Visibility:** public

17 **Behavior:**

18 a) If the *other* is a falseish value, return **true**.

19 b) Otherwise, return **false**.

20 **15.2.5.3.3 `TrueClass#to_s`**

21 `to_s`

22 **Visibility:** public

23 **Behavior:** The method creates a direct instance of the class `String`, the content of which
24 is “true”, and returns this instance.

25 **15.2.5.3.4 `TrueClass#|`**

1 | (*other*)

2 | **Visibility:** public

3 | **Behavior:** The method returns **true**.

4 | 15.2.6 FalseClass

5 | 15.2.6.1 General description

6 | The class `FalseClass` has only one instance, which is represented by the pseudo variable **false**.
7 | **false** represents a logical false value.

8 | Instances of the class `FalseClass` shall not be created by the method `new` of the class `FalseClass`.
9 | Therefore, a conforming processor shall undefine the singleton method `new` of the class `FalseClass`,
10 | as if by invoking the method `undef_method` on the eigenclass of the class `FalseClass` with a
11 | direct instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.4.42).

12 | 15.2.6.2 Direct superclass

13 | The class `Object`

14 | 15.2.6.3 Instance methods

15 | 15.2.6.3.1 `FalseClass#&`

16 | `&(other)`

17 | **Visibility:** public

18 | **Behavior:** The method returns **false**.

19 | 15.2.6.3.2 `FalseClass#^`

20 | `^(other)`

21 | **Visibility:** public

22 | **Behavior:**

23 | a) If the *other* is a falseish value, return **false**.

24 | b) Otherwise, return **true**.

25 | 15.2.6.3.3 `FalseClass#to_s`

1 to_s

2 **Visibility:** public

3 **Behavior:** The method creates a direct instance of the class `String`, the content of which
4 is “false”, and returns this instance.

5 **15.2.6.3.4 FalseClass#|**

6 | (*other*)

7 **Visibility:** public

8 **Behavior:**

9 a) If the *other* is a falseish value, return **false**.

10 b) Otherwise, return **true**.

11 **15.2.7 Numeric**

12 **15.2.7.1 General description**

13 Instances of the class `Numeric` represent numbers. The class `Numeric` is a superclass of all the
14 other built-in classes which represent numbers.

15 The notation “the value of the instance *N* of the class `Numeric`” means the number which *N*
16 represent.

17 **15.2.7.2 Direct superclass**

18 The class `Object`

19 **15.2.7.3 Included modules**

20 The following module is included in the class `Numeric`.

- 21
 - `Comparable`

22 **15.2.7.4 Instance methods**

23 **15.2.7.4.1 Numeric#+@**

24 +@

25 **Visibility:** public

26 **Behavior:** The method returns the receiver.

1 **15.2.7.4.2 Numeric#-@**

2 -@

3 **Visibility:** public

4 **Behavior:**

5 a) Invoke the method `coerce` on the receiver with an instance of the class `Integer` whose
6 value is 0 as the only argument. Let V be the resulting value.

7 1) If V is an instance of the class `Array` which contains two elements, let F and S
8 be the first and the second element of V respectively.

9 i) Invoke the method `-` on F with S as the only argument.

10 ii) Return the resulting value.

11 2) Otherwise, raise a direct instance of the class `TypeError`.

12 **15.2.7.4.3 Numeric#abs**

13 `abs`

14 **Visibility:** public

15 **Behavior:**

16 a) Invoke the method `<` on the receiver with an instance of the class `Integer` whose value
17 is 0.

18 b) If this invocation results in a trueish value, invoke the method `-@` on the receiver and
19 return the resulting value.

20 Otherwise, return the receiver.

21 **15.2.7.4.4 Numeric#coerce**

22 `coerce(other)`

23 **Visibility:** public

24 **Behavior:**

25 a) If the class of the receiver and the class of the *other* are the same class, let X and Y
26 be the *other* and the receiver, respectively.

- 1 b) Otherwise, let *X* and *Y* be instances of the class `Float` which are converted from the
2 *other* and the receiver, respectively. the *other* and the receiver are converted as follows:
- 3 1) Let *O* be the *other* or the receiver.
- 4 2) If *O* is an instance of the class `Float`, let *F* be *O*.
- 5 3) Otherwise:
- 6 i) If an invocation of the method `respond_to?` on *O* with a direct instance of
7 the class `Symbol` whose name is `to_f` as the argument results in a falseish
8 value, raise a direct instance of the class `TypeError`.
- 9 ii) Invoke the method `to_f` on *O* with no arguments, and let *F* be the resulting
10 value.
- 11 iii) If *F* is not an instance of the class `Float`, raise a direct instance of the class
12 `TypeError`.
- 13 4) If the value of *F* is NaN, the behavior is unspecified.
- 14 5) The converted value of *O* is *F*.
- 15 c) Create a direct instance of the class `Array` which consists of two elements: the first is
16 *X*; the second is *Y*.
- 17 d) Return the instance of the class `Array`.

18 **15.2.8 Integer**

19 **15.2.8.1 General description**

20 Instances of the class `Integer` represent integers. The ranges of these integers are unbounded.

21 Instances of the class `Integer` shall not be created by the method `new` of the class `Integer`.
22 Therefore, a conforming processor shall undefine the singleton method `new` of the class `Integer`,
23 as if by invoking the method `undef_method` on the eigenclass of the class `Integer` with a direct
24 instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.4.42).

25 A conforming processor may define subclasses of the class `Integer` which differ only in the
26 ranges of the representing integer values. In this case, a conforming processor:

- 27 • shall define instance methods `+`, `-`, `*`, `/`, and `%` in all of these classes.
- 28 • shall not create a direct instance of the class `Integer`, but shall create a direct instance of
29 one of these subclasses, instead of the class `Integer`.

30 If a conforming processor does not define any subclass of the class `Integer`, the processor shall
31 define instance methods `+`, `-`, `*`, `/` and `%` in the class `Integer`.

32 **15.2.8.2 Direct superclass**

33 The class `Numeric`

1 **15.2.8.3 Instance methods**

2 **15.2.8.3.1 Integer#+**

3 `+(other)`

4 **Visibility:** public

5 **Behavior:**

6 a) If the *other* is an instance of the class `Integer`, return an instance of the class `Integer`
7 whose value is the sum of the values of the receiver and the *other*.

8 b) If the *other* is an instance of the class `Float`, let *R* be the value of the receiver as a
9 floating point number.

10 Return a direct instance of the class `Float` whose value is the sum of *R* and the value
11 of the *other*.

12 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
13 ment. Let *V* be the resulting value.

14 1) If *V* is an instance of the class `Array` which contains two elements, let *F* and *S*
15 be the first and the second element of *V* respectively.

16 i) Invoke the method `+` on *F* with *S* as the only argument.

17 ii) Return the resulting value.

18 2) Otherwise, raise a direct instance of the class `TypeError`.

19 **15.2.8.3.2 Integer#-**

20 `-(other)`

21 **Visibility:** public

22 **Behavior:**

23 a) If the *other* is an instance of the class `Integer`, return an instance of the class `Integer`
24 whose value is the result of subtracting the value of the *other* from the value of the
25 receiver.

26 b) If the *other* is an instance of the class `Float`, let *R* be the value of the receiver as a
27 floating point number.

28 Return a direct instance of the class `Float` whose value is the result of subtracting the
29 value of the *other* from *R*.

- 1 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
2 ment. Let V be the resulting value.
- 3 1) If V is an instance of the class `Array` which contains two elements, let F and S
4 be the first and the second element of V respectively.
- 5 i) Invoke the method `-` on F with S as the only argument.
- 6 ii) Return the resulting value.
- 7 2) Otherwise, raise a direct instance of the class `TypeError`.

8 15.2.8.3.3 `Integer#*`

9 `*(other)`

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the *other* is an instance of the class `Integer`, return an instance of the class `Integer`
13 whose value is the result of multiplication of the values of the receiver and the *other*.
- 14 b) If the *other* is an instance of the class `Float`, let R be the value of the receiver as a
15 floating point number.
- 16 Return a direct instance of the class `Float` whose value is the result of multiplication
17 of R and the value of the *other*.
- 18 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
19 ment. Let V be the resulting value.
- 20 1) If V is an instance of the class `Array` which contains two elements, let F and S
21 be the first and the second element of V respectively.
- 22 i) Invoke the method `*` on F with S as the only argument.
- 23 ii) Return the resulting value.
- 24 2) Otherwise, raise a direct instance of the class `TypeError`.

25 15.2.8.3.4 `Integer# /`

26 `/(other)`

27 **Visibility:** public

28 **Behavior:**

- 1 a) If the *other* is an instance of the class `Integer`:
- 2 1) If the value of the *other* is 0, raise a direct instance of the class `ZeroDivisionError`.
- 3 2) Otherwise, let n be the value of the receiver divided by the value of the *other*.
4 Return an instance of the class `Integer` whose value is the largest integer smaller
5 than or equal to n .
- 6 b) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
7 ment. Let V be the resulting value.
- 8 1) If V is an instance of the class `Array` which contains two elements, let F and S
9 be the first and the second element of V respectively.
- 10 i) Invoke the method `/` on F with S as the only argument.
- 11 ii) Return the resulting value.
- 12 2) Otherwise, raise a direct instance of the class `TypeError`.

13 **15.2.8.3.5 Integer#%**

14 `%(other)`

15 **Visibility:** public

16 **Behavior:**

- 17 a) If the *other* is an instance of the class `Integer`:
- 18 1) If the value of the *other* is 0, raise a direct instance of the class `ZeroDivisionError`.
- 19 2) Otherwise, let x and y be the values of the receiver and the *other*.
- 20 i) Let t be the largest integer smaller than or equal to x divided by y .
- 21 ii) Let m be $x - t \times y$.
- 22 iii) If $m \times y < 0$, return an instance of the class `Integer` whose value is $m + y$.
- 23 iv) Otherwise, return an instance of the class `Integer` whose value is m .
- 24 b) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
25 ment. Let V be the resulting value.
- 26 1) If V is an instance of the class `Array` which contains two elements, let F and S
27 be the first and the second element of V respectively.
- 28 i) Invoke the method `%` on F with S as the only argument.

- 1 ii) Return the resulting value.
- 2 2) Otherwise, raise a direct instance of the class `TypeError`.

3 **15.2.8.3.6** `Integer#<=>`

4 `<=>(other)`

5 **Visibility:** public

6 **Behavior:**

7 a) If the *other* is an instance of the class `Integer`:

- 8 1) If the value of the receiver is larger than the value of the *other*, return an instance
9 of the class `Integer` whose value is 1.
- 10 2) If the values of the receiver and the *other* are the same integer, return an instance
11 of the class `Integer` whose value is 0.
- 12 3) If the value of the receiver is smaller than the value of the *other*, return an instance
13 of the class `Integer` whose value is -1 .

14 b) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
15 ment. Let *V* be the resulting value.

16 1) If *V* is an instance of the class `Array` which contains two elements, let *F* and *S*
17 be the first and the second element of *V* respectively.

18 i) Invoke the method `<=>` on *F* with *S* as the only argument.

19 ii) If this invocation does not result in an instance of the class `Integer`, the
20 behavior is unspecified.

21 iii) Otherwise, return the value of this invocation.

22 2) Otherwise, return `nil`.

23 **15.2.8.3.7** `Integer#==`

24 `==(other)`

25 **Visibility:** public

26 **Behavior:**

27 a) If the *other* is an instance of the class `Integer`:

- 1 1) If the values of the receiver and the *other* are the same integer, return **true**.
- 2 2) Otherwise, return **false**.
- 3 b) Otherwise, invoke the method `==` on the *other* with the receiver as the argument.
- 4 Return the resulting value of this invocation.

5 **15.2.8.3.8 Integer#~**

6 ~

7 **Visibility:** public

8 **Behavior:** The method returns an instance of the class `Integer` whose two's complement
9 representation is the one's complement of the two's complement representation of the re-
10 ceiver.

11 **15.2.8.3.9 Integer#&**

12 &(*other*)

13 **Visibility:** public

14 **Behavior:**

- 15 a) If the *other* is not an instance of the class `Integer`, the behavior is unspecified.
- 16 b) Otherwise, return an instance of the class `Integer` whose two's complement represen-
17 tation is the bitwise AND of the two's complement representations of the receiver and
18 the *other*.

19 **15.2.8.3.10 Integer#|**

20 |(*other*)

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the *other* is not an instance of the class `Integer`, the behavior is unspecified.
- 24 b) Otherwise, return an instance of the class `Integer` whose two's complement repre-
25 sentation is the bitwise inclusive OR of the two's complement representations of the
26 receiver and the *other*.

27 **15.2.8.3.11 Integer#^**

1 $\sim(\textit{other})$

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is not an instance of the class `Integer`, the behavior is unspecified.

5 b) Otherwise, return an instance of the class `Integer` whose two's complement representation is the bitwise exclusive OR of the two's complement representations of the receiver and the *other*.

8 **15.2.8.3.12 Integer#<<**

9 $\ll(\textit{other})$

10 **Visibility:** public

11 **Behavior:**

12 a) If the *other* is not an instance of the class `Integer`, the behavior is unspecified.

13 b) Otherwise, let x and y be the values of the receiver and the *other*.

14 c) Return an instance of the class `Integer` whose value is the largest integer smaller than or equal to $x \times 2^y$.

16 **15.2.8.3.13 Integer#>>**

17 $\gg(\textit{other})$

18 **Visibility:** public

19 **Behavior:**

20 a) If the *other* is not an instance of the class `Integer`, the behavior is unspecified.

21 b) Otherwise, let x and y be the values of the receiver and the *other*.

22 c) Return an instance of the class `Integer` whose value is the largest integer smaller than or equal to $x \times 2^{-y}$.

24 **15.2.8.3.14 Integer#ceil**

1 **ceil**

2 **Visibility:** public

3 **Behavior:** The method returns the receiver.

4 **15.2.8.3.15 Integer#downto**

5 **downto**(*num*, &*block*)

6 **Visibility:** public

7 **Behavior:**

8 a) If the *num* is not an instance of the class **Integer**, or the *block* is not given, the
9 behavior is unspecified.

10 b) Let *i* be the value of the receiver.

11 c) If *i* is smaller than the value of the *num*, return the receiver.

12 d) Call the *block* with an instance of the class **Integer** whose value is *i*.

13 e) Decrement *i* by 1 and continue processing from Step c.

14 **15.2.8.3.16 Integer#eql?**

15 **eql?**(*other*)

16 **Visibility:** public

17 **Behavior:**

18 a) If the *other* is not an instance of the class **Integer**, return **false**.

19 b) Otherwise, invoke the method **==** on the *other* with the receiver as the argument.

20 c) If this invocation results in a trueish value, return **true**. Otherwise, return **false**.

21 **15.2.8.3.17 Integer#floor**

22 **floor**

23 **Visibility:** public

24 **Behavior:** The method returns the receiver.

1 **15.2.8.3.18 Integer#hash**

2 **hash**

3 **Visibility:** public

4 **Behavior:** The method returns an implementation-defined instance of the class `Integer`,
5 which satisfies the following condition:

6 a) Let I_1 and I_2 be instances of the class `Integer`.

7 b) Let H_1 and H_2 be the resulting values of invocations of the method `hash` on I_1 and I_2 ,
8 respectively.

9 c) The values of the H_1 and H_2 shall be the same integer, if and only if the values of I_1
10 and I_2 are the same integer.

11 **15.2.8.3.19 Integer#next**

12 **next**

13 **Visibility:** public

14 **Behavior:** The method returns an instance of the class `Integer`, whose value is the value
15 of the receiver plus 1.

16 **15.2.8.3.20 Integer#round**

17 **round**

18 **Visibility:** public

19 **Behavior:** The method returns the receiver.

20 **15.2.8.3.21 Integer#succ**

21 **succ**

22 **Visibility:** public

23 **Behavior:** Same as the method `next` (see §15.2.8.3.19).

24 **15.2.8.3.22 Integer#times**

1 `times(&block)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the *block* is not given, the behavior is unspecified.

5 b) Let *i* be 0.

6 c) If *i* is larger than or equal to the value of the receiver, return the receiver.

7 d) Call the *block* with an instance of the class `Integer` whose value is *i*.

8 e) Increment *i* by 1 and continue processing from Step c.

9 **15.2.8.3.23 Integer#to_f**

10 `to_f`

11 **Visibility:** public

12 **Behavior:** The method returns a direct instance of the class `Float` whose value is the
13 value of the receiver as a floating point number.

14 **15.2.8.3.24 Integer#to_i**

15 `to_i`

16 **Visibility:** public

17 **Behavior:** The method returns the receiver.

18 **15.2.8.3.25 Integer#truncate**

19 `truncate`

20 **Visibility:** public

21 **Behavior:** The method returns the receiver.

22 **15.2.8.3.26 Integer#upto**

1 `upto(num, &block)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the *num* is not an instance of the class `Integer`, or the *block* is not given, the
5 behavior is unspecified.

6 b) Let *i* be the value of the receiver.

7 c) If *i* is larger than the value of the *num*, return the receiver.

8 d) Call the *block* with an instance of the class `Integer` whose value is *i*.

9 e) Increment *i* by 1 and continue processing from Step c.

10 **15.2.9 Float**

11 **15.2.9.1 General description**

12 Instances of the class `Float` represent floating point numbers.

13 When an arithmetic operation involving floating point numbers results in a value which cannot
14 be represented exactly as an instance of the class `Float`, how the result is rounded to fit in the
15 representation of an instance of the class `Float` is implementation-defined.

16 If the underlying platform of a conforming processor supports IEC 60559:1989:

- 17 • The representation of an instance of the class `Float` shall be the 64-bit double format as
18 specified in §3.2.2 of IEC 60559:1989.
- 19 • If an arithmetic operation involving floating point numbers results in NaN while invoking
20 a method of the class `Float`, the behavior of the method is unspecified.

21 Instances of the class `Float` shall not be created by the method `new` of the class `Float`. Therefore,
22 a conforming processor shall undefine the singleton method `new` of the class `Float`, as if by
23 invoking the method `undef_method` on the eigenclass of the class `Float` with a direct instance
24 of the class `Symbol` whose name is “new” as the argument (see §15.2.2.4.42).

25 **15.2.9.2 Direct superclass**

26 The class `Numeric`

27 **15.2.9.3 Instance methods**

28 **15.2.9.3.1 Float#+**

1 + (*other*)

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is an instance of the class `Float`, return a direct instance of the class `Float`
5 whose value is the sum of the values of the receiver and the *other*.

6 b) If the *other* is an instance of the class `Integer`, let *R* be the value of the *other* as a
7 floating point number.

8 Return a direct instance of the class `Float` whose value is the sum of *R* and the value
9 of the receiver.

10 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
11 ment. Let *V* be the resulting value.

12 1) If *V* is an instance of the class `Array` which contains two elements, let *F* and *S*
13 be the first and the second element of *V* respectively.

14 i) Invoke the method `+` on *F* with *S* as the only argument.

15 ii) Return the resulting value.

16 2) Otherwise, raise a direct instance of the class `TypeError`.

17 **15.2.9.3.2 Float#-**

18 - (*other*)

19 **Visibility:** public

20 **Behavior:**

21 a) If the *other* is an instance of the class `Float`, return a direct instance of the class `Float`
22 whose value is the result of subtracting the value of the *other* from the value of the
23 receiver.

24 b) If the *other* is an instance of the class `Integer`, let *R* be the value of the *other* as a
25 floating point number.

26 Return a direct instance of the class `Float` whose value is the result of subtracting *R*
27 from the value of the receiver.

28 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
29 ment. Let *V* be the resulting value.

- 1 1) If V is an instance of the class `Array` which contains two elements, let F and S
2 be the first and the second element of V respectively.
- 3 i) Invoke the method `-` on F with S as the only argument.
- 4 ii) Return the resulting value.
- 5 2) Otherwise, raise a direct instance of the class `TypeError`.

6 **15.2.9.3.3 Float#***

7 `*(other)`

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the *other* is an instance of the class `Float`, return a direct instance of the class `Float`
11 whose value is the result of multiplication of the values of the receiver and the *other*.
- 12 b) If the *other* is an instance of the class `Integer`, let R be the value of the *other* as a
13 floating point number.
- 14 Return a direct instance of the class `Float` whose value is the result of multiplication
15 of R and the value of the receiver.
- 16 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
17 ment. Let V be the resulting value.
- 18 1) If V is an instance of the class `Array` which contains two elements, let F and S
19 be the first and the second element of V respectively.
- 20 i) Invoke the method `*` on F with S as the only argument.
- 21 ii) Return the resulting value.
- 22 2) Otherwise, raise a direct instance of the class `TypeError`.

23 **15.2.9.3.4 Float#/**

24 `/(other)`

25 **Visibility:** public

26 **Behavior:**

- 27 a) If the *other* is an instance of the class `Float`, return a direct instance of the class `Float`
28 whose value is the value of the receiver divided by the value of the *other*.

1 b) If the *other* is an instance of the class `Integer`, let R be the value of the *other* as a
2 floating point number.

3 Return a direct instance of the class `Float` whose value is the value of the receiver
4 divided by R .

5 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
6 ment. Let V be the resulting value.

7 1) If V is an instance of the class `Array` which contains two elements, let F and S
8 be the first and the second element of V respectively.

9 i) Invoke the method `/` on F with S as the only argument.

10 ii) Return the resulting value.

11 2) Otherwise, raise a direct instance of the class `TypeError`.

12 **15.2.9.3.5** `Float#%`

13 `%(other)`

14 **Visibility:** public

15 **Behavior:** In the following steps, binary operators `+`, `-`, and `*` represent floating point
16 arithmetic operations addition, subtraction, and multiplication which are used in the in-
17 stance methods `+`, `-`, and `*` of the class `Float`, respectively. The operator `*` has a higher
18 precedence than the operators `+` and `-`.

19 a) If the *other* is an instance of the class `Integer` or the class `Float`:

20 Let x be the value of the receiver.

21 1) If the *other* is an instance of the class `Float`, let y be the value of the *other*. If
22 the *other* is an instance of the class `Integer`, let y be the value of the *other* as a
23 floating point number.

24 i) Let t be the largest integer smaller than or equal to x divided by y .

25 ii) Let m be $x - t * y$.

26 iii) If $m * y < 0$, return a direct instance of the class `Float` whose value is $m +$
27 y .

28 iv) Otherwise, return a direct instance of the class `Float` whose value is m .

29 b) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
30 ment. Let V be the resulting value.

31 1) If V is an instance of the class `Array` which contains two elements, let F and S
32 be the first and the second element of V respectively.

- 1 i) Invoke the method % on F with S as the only argument.
- 2 ii) Return the resulting value.
- 3 2) Otherwise, raise a direct instance of the class `TypeError`.

4 **15.2.9.3.6 Float#<=>**

5 <=>(*other*)

6 **Visibility:** public

7 **Behavior:**

- 8 a) If the *other* is an instance of the class `Integer` or the class `Float`:
 - 9 1) Let a be the value of the receiver. If the *other* is an instance of the class `Float`,
10 let b be the value of the *other*. Otherwise, let b be the value of the *other* as a
11 floating point number.
 - 12 2) If a conforming processor supports IEC 60559:1989, and if a or b is NaN, then
13 return an implementation-defined value.
 - 14 3) If $a > b$, return an instance of the class `Integer` whose value is 1.
 - 15 4) If $a = b$, return an instance of the class `Integer` whose value is 0.
 - 16 5) If $a < b$, return an instance of the class `Integer` whose value is -1 .
- 17 b) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
18 ment. Let V be the resulting value.
 - 19 1) If V is an instance of the class `Array` which contains two elements, let F and S
20 be the first and the second element of V respectively.
 - 21 i) Invoke the method <=> on F with S as the only argument.
 - 22 ii) If this invocation does not result in an instance of the class `Integer`, the
23 behavior is unspecified.
 - 24 iii) Otherwise, return the value of this invocation.
 - 25 2) Otherwise, return `nil`.

26 **15.2.9.3.7 Float#==**

27 ==(*other*)

1 **Visibility:** public

2 **Behavior:**

3 a) If the *other* is an instance of the class `Float`:

4 1) If a conforming processor supports IEC 60559:1989, and if the value of the receiver
5 is NaN, then return **false**.

6 2) If the values of the receiver and the *other* are the same number, return **true**.

7 3) Otherwise, return **false**.

8 b) If the *other* is an instance of the class `Integer`:

9 1) If the values of the receiver and the *other* are the mathematically the same, return
10 **true**.

11 2) Otherwise, return **false**.

12 c) Otherwise, invoke the method `==` on the *other* with the receiver as the argument and
13 return the resulting value of this invocation.

14 **15.2.9.3.8 Float#ceil**

15 **ceil**

16 **Visibility:** public

17 **Behavior:** The method returns an instance of the class `Integer` whose value is the smallest
18 integer larger than or equal to the value of the receiver.

19 **15.2.9.3.9 Float#finite?**

20 **finite?**

21 **Visibility:** public

22 **Behavior:**

23 a) If the value of the receiver is a finite number, return **true**.

24 b) Otherwise, return **false**.

25 **15.2.9.3.10 Float#floor**

1 floor

2 **Visibility:** public

3 **Behavior:** The method returns an instance of the class `Integer` whose value is the largest
4 integer smaller than or equal to the value of the receiver.

5 **15.2.9.3.11 Float#infinite?**

6 infinite?

7 **Visibility:** public

8 **Behavior:**

9 a) If the value of the receiver is the positive infinite, return an instance of the class `Integer`
10 whose value is 1.

11 b) If the value of the receiver is the negative infinite, return an instance of the class
12 `Integer` whose value is -1 .

13 c) Otherwise, return `nil`.

14 **15.2.9.3.12 Float#round**

15 round

16 **Visibility:** public

17 **Behavior:** The method returns an instance of the class `Integer` whose value is the nearest
18 integer to the value of the receiver. If there are two integers equally distant from the value
19 of the receiver, the one which has the larger absolute value is chosen.

20 **15.2.9.3.13 Float#to_f**

21 to_f

22 **Visibility:** public

23 **Behavior:** The method returns the receiver.

24 **15.2.9.3.14 Float#to_i**

1 to_i

2 **Visibility:** public

3 **Behavior:** The method returns an instance of the class `Integer` whose value is the integer
4 part of the receiver.

5 **15.2.9.3.15 Float#truncate**

6 truncate

7 **Visibility:** public

8 **Behavior:** Same as the method `to_i` (see §15.2.9.3.14).

9 **15.2.10 String**

10 **15.2.10.1 General description**

11 Instances of the class `String` represent sequences of characters.

12 An instance of the class `String` which does not contain any character is said to be **empty**. An
13 instance of the class `String` shall be empty when it is created by Step b of the method `new` of
14 the class `Class`.

15 The notation “an instance of the class `Object` which represents the character *C*” means either
16 of the following:

- 17 • An instance of the class `Integer` whose value is the character code of *C*.
18 • An instance of the class `String` whose content is the single character *C*.

19 A conforming processor shall choose one of the above representations and use the same repre-
20 sentation wherever this notation is used.

21 The notation “the *n*th character of an instance of the class `String`” means the character of the
22 instance whose index is *n* counted up from 0.

23 **15.2.10.2 Direct superclass**

24 The class `Object`

25 **15.2.10.3 Included modules**

26 The following modules are included in the class `String`.

- 27 • `Comparable`

1 **15.2.10.4 Upper-case and lower-case characters**

2 Some methods of the class `String` handle upper-case and lower-case characters. The correspon-
3 dence between upper-case and lower-case characters is given in Table 3.

Table 3 – The correspondence between upper-case and lower-case characters

upper-case characters	lower-case characters
A	a
B	b
C	c
D	d
E	e
F	f
G	g
H	h
I	i
J	j
K	k
L	l
M	m
N	n
O	o
P	p
Q	q
R	r
S	s
T	t
U	u
V	v
W	w
X	x
Y	y
Z	z

4 **15.2.10.5 Instance methods**

5 **15.2.10.5.1 `String#*`**

6 `*(num)`

7 **Visibility:** public

1 **Behavior:**

- 2 a) If the *num* is not an instance of the class `Integer`, the behavior is unspecified.
- 3 b) Let *n* be the value of the *num*.
- 4 c) If *n* is smaller than 0, raise a direct instance of the class `ArgumentError`.
- 5 d) Otherwise, let *C* be the content of the receiver.
- 6 e) Create a direct instance of the class `String` *S* the content of which is *C* repeated *n*
7 times.
- 8 f) Return *S*.

9 **15.2.10.5.2 String#+**

10 +(*other*)

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the *other* is not an instance of the class `String`, the behavior is unspecified.
- 14 b) Let *S* and *O* be the contents of the receiver and the *other* respectively.
- 15 c) Return a new direct instance of the class `String` the content of which is the concate-
16 nation of *S* and *O*.

17 **15.2.10.5.3 String#<=>**

18 <=>(*other*)

19 **Visibility:** public

20 **Behavior:**

- 21 a) If the *other* is not an instance of the class `String`, the behavior is unspecified.
- 22 b) Let *S1* and *S2* be the contents of the receiver and the *other* respectively.
- 23 c) If both *S1* and *S2* are empty, return an instance of the class `Integer` whose value is 0.
- 24 d) If *S1* is empty, return an instance of the class `Integer` whose value is -1.
- 25 e) If *S2* is empty, return an instance of the class `Integer` whose value is 1.
- 26 f) Let *a*, *b* be the character codes of the first characters of *S1* and *S2* respectively.

- 1) If $a > b$, return an instance of the class `Integer` whose value is 1.
- 2) If $a < b$, return an instance of the class `Integer` whose value is -1 .
- 3) Otherwise, replace $S1$ and $S2$ with $S1$ and $S2$ excluding their first characters, respectively. Continue processing from Step c.

15.2.10.5.4 `String#==`

`==(other)`

Visibility: public

Behavior:

- a) If the *other* is not an instance of the class `String`, the behavior is unspecified.
- b) If the *other* is an instance of the class `String`:
 - 1) If the content of the receiver and the *other* is the same, return **true**.
 - 2) Otherwise, return **false**.

15.2.10.5.5 `String#=~`

`=~(regexp)`

Visibility: public

Behavior:

- a) If the *regexp* is not an instance of the class `Regexp`, the behavior is unspecified.
- b) Otherwise, behave as if the method `match` is invoked on the *regexp* with the receiver as the argument (see §15.2.15.7.7).

15.2.10.5.6 `String#[]`

`[](*args)`

Visibility: public

Behavior:

- a) If the length of the *args* is 0 or larger than 2, raise a direct instance of the class `ArgumentError`.

- 1 b) Let P be the first element of the *args*. Let n be the length of the receiver.
- 2 c) If P is an instance of the class `Integer`, let b be the value of P .
- 3 1) If the length of the *args* is 1:
- 4 i) If b is smaller than 0, increment b by n . If b is still smaller than 0, return **nil**.
- 5 ii) If $b \geq n$, return **nil**.
- 6 iii) Create an instance of the class `Object` which represents the b th character of
- 7 the receiver and return this instance.
- 8 2) If the length of the *args* is 2:
- 9 i) If the last element of the *args* is an instance of the class `Integer`, let l be the
- 10 value of the instance. Otherwise, the behavior is unspecified.
- 11 ii) If l is smaller than 0, or b is larger than n , return **nil**.
- 12 iii) If b is smaller than 0, increment b by n . If b is still smaller than 0, return **nil**.
- 13 iv) If $b + l$ is larger than n , let l be $n - b$.
- 14 v) If l is smaller than or equal to 0, create an empty direct instance of the class
- 15 `String` and return the instance.
- 16 vi) Otherwise, create a direct instance of the class `String` whose content is the
- 17 $(n-l)$ characters of the receiver, from the b th index, preserving their order.
- 18 Return the instance.
- 19 d) If P is an instance of the class `Regex`:
- 20 1) If the length of the *args* is 1, let i be 0.
- 21 2) If the length of the *args* is 2, and the last element of *args* is an instance of the class
- 22 `Integer`, let i be the value of the instance. Otherwise, the behavior is unspecified.
- 23 3) Match the pattern of P against the content of the receiver. (see §15.2.15.4 and
- 24 Step 15.2.15.5). Let M be the result of the matching process.
- 25 4) If M is **nil**, return **nil**.
- 26 5) If i is larger than the length of the match result of M , return **nil**.
- 27 6) If i is smaller than 0, increment i by the length of the match result of M . If i is
- 28 still smaller than or equal to 0, return **nil**.
- 29 7) Let m be the i th element of the match result of M . Create a direct instance of the
- 30 class `String` whose content is the first element of m and return the instance.
- 31 e) If P is an instance of the class `String`:

- 1) If the length of the *args* is 2, the behavior is unspecified.
 - 2) If the receiver includes the content of *P* as a substring, create a direct instance of the class **String** whose content is equal to the content *P* and return the instance.
 - 3) Otherwise, return **nil**.
- f) Otherwise, the behavior is unspecified.

15.2.10.5.7 **String#capitalize**

`capitalize`

Visibility: public

Behavior: The method returns a new direct instance of the class **String** which contains all the characters of the receiver, except:

- If the first character of the receiver is a lower-case character, the first character of the resulting instance is the corresponding upper-case character.
- If the *i*th character of the receiver (where *i* > 0) is an upper case character, the *i*th character of the resulting instance is the corresponding lower-case character.

15.2.10.5.8 **String#capitalize!**

`capitalize!`

Visibility: public

Behavior:

- a) Let *s* be the content of the instance of the class **String** returned when the method `capitalize` is invoked on the receiver.
- b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the content of the receiver to *s*, and return the receiver.

15.2.10.5.9 **String#chomp**

`chomp(rs = "\n")`

Visibility: public

Behavior:

- a) If the *rs* is **nil**, return a new direct instance of the class **String** whose content is the same as the receiver.

- 1 b) If the receiver is empty, return a new direct instance of the class **String**.
- 2 c) If *rs* is not an instance of the class **String**, the behavior is unspecified.
- 3 d) Otherwise, return a new direct instance of the class **String** whose content is the same
- 4 as the receiver, except the following characters:
 - 5 1) If the *rs* consists of only one character 0x0a, the *line-terminator* on the end, if
 - 6 any, is excluded.
 - 7 2) If the *rs* is empty, the sequence of *line-terminators* on the end, if any, is excluded.
 - 8 3) Otherwise, if the receiver ends with the content of *rs*, this sequence of the charac-
 - 9 ters at the end of the receiver is excluded.

10 **15.2.10.5.10 String#chomp!**

11 chomp! (*rs* = "\n")

12 **Visibility:** public

13 **Behavior:**

- 14 a) Let *s* be the content of the instance of the class **String** returned when the method
- 15 **chomp** is invoked on the receiver with the *rs* as the argument.
- 16 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
- 17 content of the receiver to *s*, and return the receiver.

18 **15.2.10.5.11 String#chop**

19 chop

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the receiver is empty, return a new direct instance of the class **String**.
- 23 b) Otherwise, create a new direct instance of the class **String** whose content is the receiver
- 24 without the last character and return this instance. If the last character is 0x0a, and
- 25 the character just before the 0x0a is 0x0d, the 0x0d is also dropped.

26 **15.2.10.5.12 String#chop!**

27 chop!

1 **Visibility:** public

2 **Behavior:**

3 a) Let s be the content of the instance of the class `String` returned when the method
4 `chop` is invoked on the receiver.

5 b) If the content of the receiver and s are the same, return **nil**. Otherwise, change the
6 content of the receiver to s , and return the receiver.

7 **15.2.10.5.13 String#downcase**

8 `downcase`

9 **Visibility:** public

10 **Behavior:** The method returns a new direct instance of the class `String` which contains
11 all the characters of the receiver, with the upper-case characters replaced with the corre-
12 sponding lower-case characters.

13 **15.2.10.5.14 String#downcase!**

14 `downcase!`

15 **Visibility:** public

16 **Behavior:**

17 a) Let s be the content of the instance of the class `String` returned when the method
18 `downcase` is invoked on the receiver.

19 b) If the content of the receiver and s are the same, return **nil**. Otherwise, change the
20 content of the receiver to s , and return the receiver.

21 **15.2.10.5.15 String#each_line**

22 `each_line(&block)`

23 **Visibility:** public

24 **Behavior:** Let s be the content of the receiver. Let c be the first character of s .

25 a) If the *block* is not given, the behavior is unspecified.

26 b) Find the first 0x0a in s from c . If there is such a 0x0a:

27 1) Let d be that 0x0a.

- 1 2) Create a direct instance of the class **String** *S* whose content is the sequence of
2 the characters from *c* to *d*.
- 3 3) Call the *block* with *S* as the argument.
- 4 4) If *d* is the last character of *s*, return the receiver. Otherwise, let new *c* be the
5 character just after *d* and continue processing from Step b.
- 6 c) If there is not such a 0x0a, create a direct instance of the class **String** whose content
7 is the sequence of the characters from *c* to the last character of *s*. Call the *block* with
8 this instance as the argument.
- 9 d) Return the receiver.

10 **15.2.10.5.16 String#empty?**

11 empty?

12 **Visibility:** public

13 **Behavior:**

14 a) If the receiver is empty, return **true**.

15 b) Otherwise, return **false**.

16 **15.2.10.5.17 String#eql?**

17 eql?(*other*)

18 **Visibility:** public

19 **Behavior:**

20 a) If the *other* is an instance of the class **String**:

21 1) If the contents of the receiver and the *other* are the same, return **true**.

22 2) Otherwise, return **false**.

23 b) If the *other* is not an instance of the class **String**, return **false**.

24 **15.2.10.5.18 String#gsub**

25 gsub(**args*, &*block*)

26 **Visibility:** public

1 **Behavior:**

- 2 a) If the length of the *args* is 0 or larger than 2, or the length of the *args* is 1 and the
3 *block* is not given, raise a direct instance of the class **ArgumentError**.
- 4 b) Let *P* be the first element of the *args*. If *P* is not an instance of the class **Regexp**, or
5 the length of the *args* is 2 and the last element of the *args* is not an instance of the
6 class **String**, the behavior is unspecified.
- 7 c) Let *S* be the content of the receiver, and let *l* be the length of *S*.
- 8 d) Let *L* be an empty list and let *n* be an integer 0.
- 9 e) Match the pattern of *P* against *S* at the offset *n* (see §15.2.15.4 and §15.2.15.5). Let
10 *M* be the result of the matching process.
- 11 f) If *M* is **nil**, append to *L* the substring of *S* beginning at the *n*th character up to the
12 last character of *S*.
- 13 g) Otherwise:
- 14 1) If the length of the *args* is 1:
- 15 i) Call the *block* with a new direct instance of the class **String** whose content
16 is the matched substring of *M* as the argument.
- 17 ii) Let *V* be the resulting value of this call. If *V* is not an instance of the class
18 **String**, the behavior is unspecified.
- 19 2) Let *pre* be the pre-match of *M*. Append to *L* the substring of *pre* beginning at the
20 *n*th character up to the last character of *pre*, unless *n* is larger than the offset of
21 the last character of *pre*.
- 22 3) If the length of the *args* is 1, append the content of *V* to *L*. If the length of the
23 *args* is 2, append to *L* the content of the last element of the *args*.
- 24 4) Let *post* be the post-match of *M*. Let *i* be the offset of the first character of *post*
25 within *S*.
- 26 i) If *i* is equal to *n*, i.e. if *P* matched an empty string:
- 27 I) Append to *L* a new direct instance of the class **String** whose content is
28 the *i*th character of *S*.
- 29 II) Increment *n* by 1.
- 30 ii) Otherwise, replace *n* with *i*.
- 31 5) If *n* < *l*, continue processing from Step e.
- 32 h) Create a direct instance of the class **String** whose content is the concatenation of all
33 the elements of *L*, and return the instance.

1 **15.2.10.5.19 String#gsub!**

2 `gsub!(*args, &block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Let s be the content of the instance of the class **String** returned when the method
6 `gsub` is invoked on the receiver with the same arguments.
- 7 b) If the content of the receiver and s are the same, return **nil**. Otherwise, change the
8 content of the receiver to s , and return the receiver.

9 **15.2.10.5.20 String#hash**

10 `hash`

11 **Visibility:** public

12 **Behavior:** The method returns an implementation-defined instance of the class **Integer**
13 which satisfies the following condition:

- 14 a) Let S_1 and S_2 be two distinct instances of the class **String**.
- 15 b) Let H_1 and H_2 be the resulting values of the invocations of the method `hash` on S_1 and
16 S_2 respectively.
- 17 c) If and only if S_1 and S_2 has the same content, the values of H_1 and H_2 shall be the
18 same integer.

19 **15.2.10.5.21 String#include?**

20 `include?(obj)`

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the *obj* is an instance of the class **Integer**:
- 24 If the receiver includes the character whose character code is the value of the *obj*,
25 return **true**. Otherwise, return **false**.
- 26 b) If the *obj* is an instance of the class **String**:
- 27 If there exists a substring of the receiver whose sequence of characters is the same as
28 the content of the *obj*, return **true**. Otherwise, return **false**.

1 c) Otherwise, the behavior is unspecified.

2 15.2.10.5.22 String#index

3 `index(substring, offset=nil)`

4 **Visibility:** public

5 **Behavior:**

6 a) If the *substring* is not an instance of the class **String**, the behavior is unspecified.

7 b) Let *R* and *S* be the contents of the receiver and the *substring*, respectively.

8 c) If the *offset* is given:

9 1) If the *offset* is not an instance of the class **Integer**, the behavior is unspecified.

10 2) Let *n* be the value of the *offset*.

11 3) If *n* is larger than or equal to 0, let *O* be *n*.

12 4) Otherwise, let *O* be *l* + *n*, where *l* is the length of *S*.

13 5) If *O* is smaller than 0, return **nil**.

14 d) Otherwise, let *O* be 0.

15 e) If *S* appears as a substring of *R* at one or more positions whose index is larger than
16 or equal to *O*, return an instance of the class **Integer** whose value is the index of the
17 first such position.

18 f) Otherwise, return **nil**.

19 15.2.10.5.23 String#initialize

20 `initialize(str="")`

21 **Visibility:** private

22 **Behavior:**

23 a) If the *str* is not an instance of the class **String**, the behavior is unspecified.

24 b) Otherwise, initialize the content of the receiver to the same sequence of characters as
25 the content of the *str*.

26 c) Return an implementation-defined value.

1 **15.2.10.5.24 String#initialize_copy**

2 `initialize_copy(original)`

3 **Visibility:** private

4 **Behavior:**

- 5 a) If the *original* is not an instance of the class **String**, the behavior is unspecified.
- 6 b) Change the content of the receiver to the content of the *original*.
- 7 c) Return an implementation-defined value.

8 **15.2.10.5.25 String#intern**

9 `intern`

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the length of the receiver is 0, or if the receiver contains 0x00, then the behavior is
13 unspecified.
- 14 b) Otherwise, return a direct instance of the class **Symbol** whose name is the content of
15 the receiver.

16 **15.2.10.5.26 String#length**

17 `length`

18 **Visibility:** public

19 **Behavior:** The method returns an instance of the class **Integer** whose value is the number
20 of characters of the content of the receiver.

21 **15.2.10.5.27 String#match**

22 `match(regexp)`

23 **Visibility:** public

24 **Behavior:**

- 25 a) If the *regexp* is an instance of the class **Regexp**, let *R* be the *regexp*.

- 1 b) If the *regexp* is an instance of the class **String**, create a direct instance of the class
2 **Regexp** as if the method **new** is invoked on the class **Regexp** with the *regexp* as the
3 argument. Let *R* be the instance of the class **Regexp**.
- 4 c) Otherwise, the behavior is unspecified.
- 5 d) Invoke the method **match** on *R* with the receiver as the argument.
- 6 e) Return the resulting value of the invocation.

7 **15.2.10.5.28 String#replace**

8 `replace(other)`

9 **Visibility:** public

10 **Behavior:** Same as the method `initialize_copy` (see §15.2.10.5.24).

11 **15.2.10.5.29 String#reverse**

12 `reverse`

13 **Visibility:** public

14 **Behavior:** The method returns a new direct instance of the class **String** which contains
15 all the characters of the content of the receiver in the reverse order.

16 **15.2.10.5.30 String#reverse!**

17 `reverse!`

18 **Visibility:** public

19 **Behavior:**

- 20 a) Change the content of the receiver to the content of the resulting instance of the class
21 **String** when the method `reverse` is invoked on the receiver.
- 22 b) Return the receiver.

23 **15.2.10.5.31 String#rindex**

24 `rindex(substring, offset=nil)`

25 **Visibility:** public

1 **Behavior:**

- 2 a) If the *substring* is not an instance of the class **String**, the behavior is unspecified.
- 3 b) Let *R* and *S* be the contents of the receiver and the *substring*, respectively.
- 4 c) If the *offset* is given:
- 5 1) If the *offset* is not an instance of the class **Integer**, the behavior is unspecified.
- 6 2) Let *n* be the value of the *offset*.
- 7 3) If *n* is larger than or equal to 0, let *O* be *n*.
- 8 4) Otherwise, let *O* be $l + n$, where *l* is the length of *S*.
- 9 5) If *O* is smaller than 0, return **nil**.
- 10 d) Otherwise, let *O* be 0.
- 11 e) If *S* appears as a substring of *R* at one or more positions whose index is smaller than
12 or equal to *O*, return an instance of the class **Integer** whose value is the index of the
13 last such position.
- 14 f) Otherwise, return **nil**.

15 **15.2.10.5.32 String#scan**

16 `scan(reg, &block)`

17 **Visibility:** public

18 **Behavior:**

- 19 a) If the *reg* is not an instance of the class **Regexp**, the behavior is unspecified.
- 20 b) If the *block* is not given, create an empty direct instance of the class **Array** *A*.
- 21 c) Let *S* be the content of the receiver, and let *l* be the length of *S*.
- 22 d) Let *n* be an integer 0.
- 23 e) Match the pattern of the *reg* against *S* at the offset *n* (see §15.2.15.4 and Step 15.2.15.5).
24 Let *M* be the result of the matching process.
- 25 f) If *M* is not **nil**:
- 26 1) Let *L* be the match result of *M*.
- 27 2) If the length of *L* is 1, create a direct instance of the class **String** *V* whose content
28 is the matched substring of *M*.

- 1 3) If the length of L is larger than 1:
- 2 i) Create an empty direct instance of the class `Array` V .
- 3 ii) Except for the first element, for each element e of L , in the same order in the
4 list, append to V a new direct instance of the class `String` whose content is
5 the first element of e .
- 6 4) If the *block* is given, call the *block* with V as the argument. Otherwise, append V
7 to A .
- 8 5) Let *post* be the post-match of M . Let i be the offset of the first character of *post*
9 within S .
- 10 i) If i and n are the same, i.e. if the *reg* matched the empty string, increment
11 n by 1.
- 12 ii) Otherwise, replace n with i .
- 13 6) If $n < l$, continue processing from Step e.
- 14 g) If the *block* is given, return the receiver. Otherwise, return A .

15 **15.2.10.5.33** `String#size`

16 `size`

17 **Visibility:** public

18 **Behavior:** Same as the method `length` (see §15.2.10.5.26).

19 **15.2.10.5.34** `String#slice`

20 `slice(*args)`

21 **Visibility:** public

22 **Behavior:** Same as the method `[]` (see §15.2.10.5.6).

23 **15.2.10.5.35** `String#split`

24 `split(sep)`

25 **Visibility:** public

26 **Behavior:**

- 1 a) If the *sep* is not an instance of the class **Regexp**, the behavior is unspecified.
- 2 b) Create an empty direct instance of the class **Array** *A*.
- 3 c) Let *S* be the content of the receiver, and let *l* be the length of *S*.
- 4 d) Let both *sp* and *bp* be 0, and let *was-empty* be false.
- 5 e) Match the pattern of the *sep* against *S* at the offset *sp* (see §15.2.15.4 and Step
6 15.2.15.5). Let *M* be the result of the matching process.
- 7 f) If *M* is **nil**, append to *A* a new direct instance of the class **String** whose content is the
8 substring of *S* beginning at the *sp*th character up to the last character of *S*.
- 9 g) Otherwise:
 - 10 1) If the matched substring of *M* is an empty string:
 - 11 i) If *was-empty* is true, append to *A* a new direct instance of the class **String**
12 whose content is the *bp*th character of *S*.
 - 13 ii) Otherwise, increment *sp* by 1. If *sp* < *l*, replace *was-empty* with true and
14 continue processing from Step e.
 - 15 2) Otherwise, replace *was-empty* with false. Let *pre* be the pre-match of *M*. Append
16 to *A* a new direct instance of the class **String** whose content is the substring of
17 *pre* beginning at the *bp*th character up to the last character of *pre*, unless *bp* is
18 larger than the offset of the last character of *pre*.
 - 19 3) Let *L* be the match result of *M*.
 - 20 4) If the length of *L* is larger than 1, except for the first element, for each element *e*
21 of *L*, in the same order in the list, take the following steps:
 - 22 i) Let *c* be the first element of *e*.
 - 23 ii) If *c* is not **nil**, append to *A* a new direct instance of the class **String** whose
24 content is *c*.
 - 25 5) Let *post* be the post-match of *M*, and replace both *sp* and *bp* with the offset of
26 the first character of *post*.
 - 27 6) If *sp* > *l*, continue processing from Step e.
- 28 h) If the last element of *A* is an instance of the class **String** whose content is empty,
29 remove the element. Repeat this step until the this condition does not hold.
- 30 i) Return *A*.

31 15.2.10.5.36 **String#sub**

1 `sub(*args, &block)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the length of the *args* is 1 and the *block* is given, or the length of the *args* is 2:

5 1) If the first element of the *args* is not an instance of the class `Regexp`, the behavior
6 is unspecified.

7 2) Match the pattern of the first element of the *args* against the content of the receiver
8 (see §15.2.15.4 and Step 15.2.15.5). Let *M* be the result of the matching process.

9 3) If *M* is `nil`, create a direct instance of the class `String` whose content is the same
10 as the receiver and return the instance.

11 4) Otherwise:

12 i) If the length of the *args* is 1, call the *block* with a new direct instance of the
13 class `String` whose content is the matched substring of *M* as the argument.
14 Let *S* be the resulting value of this call. If *S* is not an instance of the class
15 `String`, the behavior is unspecified.

16 ii) If the length of the *args* is 2, let *S* be the last element of the *args*. If *S* is not
17 an instance of the class `String`, the behavior is unspecified.

18 iii) Create a direct instance of the class `String` whose content is the concatenation
19 of pre-match of *M*, the content of *S*, and post-match of *M*, and return the
20 instance.

21 b) Otherwise, raise a direct instance of the class `ArgumentError`.

22 15.2.10.5.37 `String#sub!`

23 `sub!(*args, &block)`

24 **Visibility:** public

25 **Behavior:**

26 a) Let *s* be the content of the instance of the class `String` returned when the method `sub`
27 is invoked on the receiver with the same arguments.

28 b) If the content of the receiver and *s* are the same, return `nil`. Otherwise, change the
29 content of the receiver to *s*, and return the receiver.

30 15.2.10.5.38 `String#upcase`

1 **uppercase**

2 **Visibility:** public

3 **Behavior:** The method returns a new direct instance of the class **String** which contains
4 all the characters of the receiver, with all the lower-case characters replaced with the cor-
5 responding upper-case characters.

6 **15.2.10.5.39 String#uppercase!**

7 **uppercase!**

8 **Visibility:** public

9 **Behavior:**

- 10 a) Let *s* be the content of the instance of the class **String** returned when the method
11 **uppercase** is invoked on the receiver.
- 12 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
13 content of the receiver to *s*, and return the receiver.

14 **15.2.10.5.40 String#to_i**

15 **to_i(*base=10*)**

16 **Visibility:** public

17 **Behavior:**

- 18 a) If the *base* is not an instance of the class **Integer** whose value is 2, 8, 10, nor 16, the
19 behavior is unspecified. Otherwise, let *b* be the value of the *base*.
- 20 b) If the receiver is empty, return an instance of the class **Integer** whose value is 0.
- 21 c) Let *i* be 0. Increment *i* by 1 while the *i*th character of the receiver is a *whitespace*.
- 22 d) If the *i*th character of the receiver is “+” or “-”, increment *i* by 1.
- 23 e) If the *i*th character of the receiver is “0”, and any of the following conditions holds,
24 increment *i* by 2:

25 Let *c* be the character of the receiver whose index is *i* plus 1.

- 26 • *b* is 2, and *c* is “b” or “B”.
- 27 • *b* is 8, and *c* is “o” or “O”.

- 1 • b is 10, and c is “d” or “D”.
 - 2 • b is 16, and c is “x” or “X”.
 - 3 f) Let s be a sequence of the following characters of the receiver from the i th index:
 - 4 • If b is 2, *binary-digit* and “_”.
 - 5 • If b is 8, *octal-digit* and “_”.
 - 6 • If b is 10, *decimal-digit* and “_”.
 - 7 • If b is 16, *hexadecimal-digit* and “_”.
 - 8 g) If the length of s is 0, return an instance of the class `Integer` whose value is 0.
 - 9 h) If s starts with “_”, or s contains successive “_”s, the behavior is unspecified.
 - 10 i) Let n be the value of s , computed in base b .
- 11 If the “-” occurs in Step d, return an instance of the class `Integer` whose value is
 12 $-n$. Otherwise, return an instance of the class `Integer` whose value is n .

13 15.2.10.5.41 `String#to_f`

14 `to_f`

15 **Visibility:** public

16 **Behavior:**

- 17 a) If the receiver is empty, return a direct instance of the class `Float` whose value is 0.0.
- 18 b) If the receiver starts with the sequence of the characters which is a *float-literal*, return
 19 a direct instance of the class `Float` whose value is the value of the *float-literal* (see
 20 §8.7.6.2).
- 21 c) If the receiver starts with the sequence of the characters which is a *unprefixed-decimal-*
 22 *integer-literal*, return a direct instance of the class `Float` whose value is the value of
 23 the *unprefixed-decimal-integer-literal* as a floating point number (see §8.7.6.2).
- 24 d) Otherwise, return a direct instance of the class `Float` whose value is implementation-
 25 defined.

26 15.2.10.5.42 `String#to_s`

27 `to_s`

28 **Visibility:** public

29 **Behavior:** The method returns the receiver.

1 15.2.10.5.43 String#to_sym

2 to_sym

3 **Visibility:** public

4 **Behavior:** Same as the method `intern` (see §15.2.10.5.25).

5 15.2.11 Symbol

6 15.2.11.1 General description

7 Instances of the class `Symbol` represent names (see §8.7.6.6). No two instances of the class
8 `Symbol` have the same name.

9 Instances of the class `Symbol` shall not be created by the method `new` of the class `Symbol`.
10 Therefore, a conforming processor shall undefine the singleton method `new` of the class `Symbol`,
11 as if by invoking the method `undef_method` on the eigenclass of the class `Symbol` with a direct
12 instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.4.42).

13 15.2.11.2 Direct superclass

14 The class `Object`

15 15.2.11.3 Instance methods

16 15.2.11.3.1 Symbol#===

17 `===(other)`

18 **Visibility:** public

19 **Behavior:** Same as the method `==` of the module `Kernel` (see §15.3.1.3.1).

20 15.2.11.3.2 Symbol#id2name

21 `id2name`

22 **Visibility:** public

23 **Behavior:** The method creates a direct instance of the class `String`, the content of which
24 represents the name of the receiver, and returns this instance.

25 15.2.11.3.3 Symbol#to_s

1 `to_s`

2 **Visibility:** public

3 **Behavior:** Same as the method `id2name` (see §15.2.11.3.2).

4 15.2.11.3.4 `Symbol#to_sym`

5 `to_sym`

6 **Visibility:** public

7 **Behavior:** The method returns the receiver.

8 15.2.12 `Array`

9 15.2.12.1 `General description`

10 Instances of the class `Array` represent arrays, which are unbounded. An instance of the class
11 `Array` which has no element is said to be *empty*. The number of elements in an instance of the
12 class `Array` is called its *length*.

13 Instances of the class `Array` shall be empty when they are created by Step b of the method `new`
14 of the class `Class`.

15 Elements of an instance of the class `Array` has their indexes counted up from 0.

16 Given an instance of the class `Array` *A*, operations *append*, *prepend*, *remove* are defined as
17 follows:

18 **append:** To append an object *O* to *A* is defined as follows:

19 Insert *O* after the last element of *A*.

20 Appending an object to *A* increases its length by 1.

21 **prepend:** To prepend an object *O* to *A* is defined as follows:

22 Insert *O* to the first index of *A*. Original elements of *A* are moved toward the end of *A* by
23 one position.

24 Prepending an object to *A* increases its length by 1.

25 **remove:** To remove an element *X* from *A* is defined as follows:

26 a) Remove *X* from *A*.

27 b) If *X* is not the last element of *A*, move the elements after *X* toward the head of *A* by
28 one position.

1 Removing an object to A decreases its length by 1.

2 15.2.12.2 Direct superclass

3 The class `Object`

4 15.2.12.3 Included modules

5 The following module is included in the class `Array`.

- 6 • `Enumerable`

7 15.2.12.4 Singleton methods

8 15.2.12.4.1 `Array.[]`

9 `Array.[] (*items)`

10 **Visibility:** public

11 **Behavior:** The method returns a newly created instance of the class `Array` which contains
12 the elements of the *items*, preserving their order.

13 15.2.12.5 Instance methods

14 15.2.12.5.1 `Array#*`

15 `*(num)`

16 **Visibility:** public

17 **Behavior:**

- 18 a) If the *num* is not an instance of the class `Integer`, the behavior is unspecified.
- 19 b) If the value of the *num* is smaller than 0, raise a direct instance of the class `ArgumentError`.
- 20 c) If the value of the *num* is 0, return an empty direct instance of the class `Array`.
- 21 d) Otherwise, create a direct instance of the class `Array` A and repeat the following for
22 the *num* times:
23 Append all the elements of the receiver to A , preserving their order.
- 24 e) Return A .

25 15.2.12.5.2 `Array#+`

1 +(*other*)

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is an instance of the class **Array**, let *A* be the *other*. Otherwise, the
5 behavior is unspecified.

6 b) Create an empty direct instance of the class **Array** *R*.

7 c) For each element of the receiver, in the indexing order, append the element to *R*. Then,
8 for each element of *A*, in the indexing order, append the element to *R*.

9 d) Return *R*.

10 **15.2.12.5.3** **Array#<<**

11 <<(*obj*)

12 **Visibility:** public

13 **Behavior:** The method appends the *obj* to the receiver and return the receiver.

14 **15.2.12.5.4** **Array#[[]]**

15 [[]](**args*)

16 **Visibility:** public

17 **Behavior:**

18 a) Let *n* be the length of the receiver.

19 b) If the length of the *args* is 0, raise a direct instance of the class **ArgumentError**.

20 c) If the length of the *args* is 1:

21 1) If the only argument is an instance of the class **Integer**, let *k* be the value of the
22 only argument. Otherwise, the behavior is unspecified.

23 2) If $k < 0$, increment *k* by *n*. If *k* is still smaller than 0, return **nil**.

24 3) If $k \geq n$, return **nil**.

25 4) Otherwise, return the *k*th element of the receiver.

- 1 d) If the length of the *args* is 2:
- 2 1) If the elements of the *args* are instances of the class **Integer**, let *b* and *l* be the
- 3 values of the first and the last element of the *args*, respectively. Otherwise, the
- 4 behavior is unspecified.
- 5 2) If $b < 0$, increment *b* by *n*. If *b* is still smaller than 0, return **nil**.
- 6 3) If $b = n$, create an empty direct instance of the class **Array** and return this instance.
- 7 4) If $b > n$ or $l < 0$, return **nil**.
- 8 5) If $l > n - b$, let new *l* be $n - b$.
- 9 6) Create an empty direct instance of the class **Array** *A*. Append the *l* elements of
- 10 the receiver to *A*, from the *b*th index, preserving their order. Return *A*.
- 11 e) If the length of the *args* is larger than 2, raise a direct instance of the class **ArgumentError**.

12 **15.2.12.5.5 Array#[]=**

13 [] = (*args)

14 **Visibility:** public

15 **Behavior:**

- 16 a) Let *n* be the length of the receiver.
- 17 b) If the length of the *args* is smaller than 2, raise a direct instance of the class **ArgumentError**.
- 18 c) If the length of the *args* is 2:
- 19 1) If the first element of the *args* is an instance of the class **Integer**, let *k* be the
- 20 value of the element and let *V* be the last element of the *args*. Otherwise, the
- 21 behavior is unspecified.
- 22 2) If $k < 0$, increment *k* by *n*. If *k* is still smaller than 0, raise a direct instance of
- 23 the class **IndexError**.
- 24 3) If $k < n$, replace the *k*th element of the receiver with *V*.
- 25 4) Otherwise, expand the length of the receiver to $k + 1$. The last element of the
- 26 receiver is *V*. If $k > n$, the elements whose index is from *n* to $k - 1$ is **nil**.
- 27 5) Return *V*.
- 28 d) If the length of the *args* is 3, the behavior is unspecified.
- 29 e) If the length of the *args* is larger than 3, raise a direct instance of the class **ArgumentError**.

1 **15.2.12.5.6** `Array#clear`

2 `clear`

3 **Visibility:** public

4 **Behavior:** The method removes all the elements from the receiver and return the receiver.

5 **15.2.12.5.7** `Array#collect!`

6 `collect!(&block)`

7 **Visibility:** public

8 **Behavior:**

9 a) If the *block* is given:

10 1) For each element of the receiver in the indexing order, call the *block* with the
11 element as the only argument and replace the element with the resulting value.

12 2) Return the receiver.

13 b) If the *block* is not given, the behavior is unspecified.

14 **15.2.12.5.8** `Array#concat`

15 `concat(other)`

16 **Visibility:** public

17 **Behavior:**

18 a) If the *other* is not an instance of the class `Array`, the behavior is unspecified.

19 b) Otherwise, append all the elements of the *other* to the receiver, preserving their order.

20 c) Return the receiver.

21 **15.2.12.5.9** `Array#delete_at`

22 `delete_at(index)`

23 **Visibility:** public

24 **Behavior:**

- 1 a) If the *index* is not an instance of the class `Integer`, the behavior is unspecified.
- 2 b) Otherwise, let *i* be the value of the *index*.
- 3 c) Let *n* be the length of the receiver.
- 4 d) If *i* is smaller than 0, increment *i* by *n*. If *i* is still smaller than 0, return **nil**.
- 5 e) If *i* is larger than or equal to *n*, return **nil**.
- 6 f) Otherwise, remove the *i*th element of the receiver, and return the removed element.

7 **15.2.12.5.10 Array#each**

8 `each(&block)`

9 **Visibility:** public

10 **Behavior:**

- 11 a) If the *block* is given:
 - 12 1) For each element of the receiver in the indexing order, call the *block* with the
 - 13 element as the only argument.
 - 14 2) Return the receiver.
- 15 b) If the *block* is not given, the behavior is unspecified.

16 **15.2.12.5.11 Array#each_index**

17 `each_index(&block)`

18 **Visibility:** public

19 **Behavior:**

- 20 a) If the *block* is given:
 - 21 1) For each element of the receiver in the indexing order, call the *block* with an
 - 22 argument, which is an instance of the class `Integer` whose value is the index of
 - 23 the element.
 - 24 2) Return the receiver.
- 25 b) If the *block* is not given, the behavior is unspecified.

26 **15.2.12.5.12 Array#empty?**

1 empty?

2 **Visibility:** public

3 **Behavior:**

4 a) If the receiver is empty, return **true**.

5 b) Otherwise, return **false**.

6 **15.2.12.5.13 Array#first**

7 first(*args)

8 **Visibility:** public

9 **Behavior:**

10 a) If the length of the *args* is 0:

11 1) If the receiver is empty, return **nil**.

12 2) Otherwise, return the first element of the receiver.

13 b) If the length of the *args* is 1:

14 1) If the only argument is not an instance of the class **Integer**, the behavior is
15 unspecified. Otherwise, let *n* be the value of the only argument.

16 2) If *n* is smaller than 0, raise a direct instance of the class **ArgumentError**.

17 3) Otherwise, let *N* be the smaller of *n* and the length of the receiver.

18 4) Return a newly created instance of the class **Array** which contains the first *N*
19 elements of the receiver, preserving their order.

20 c) If the length of *args* is larger than 1, raise a direct instance of the class **ArgumentError**.

21 **15.2.12.5.14 Array#index**

22 index(object=nil)

23 **Visibility:** public

24 **Behavior:**

25 a) If the *object* is given:

- 1 1) For each element E of the receiver in the indexing order, take the following steps:
 - 2 i) Invoke the method `==` on E with the *object* as the argument.
 - 3 ii) If the resulting value is a trueish value, return the index of E .
- 4 2) Return **nil**.
- 5 b) Otherwise, the behavior is unspecified.

6 **15.2.12.5.15** `Array#initialize`

```
7     initialize(size=0, obj=nil, &block)
```

8 **Visibility:** private

9 **Behavior:**

- 10 a) If the *size* is not an instance of the class `Integer`, the behavior is unspecified. Other-
11 wise, let n be the value of the *size*.
- 12 b) If n is smaller than 0, raise a direct instance of the class `ArgumentError`.
- 13 c) If n is 0, return an implementation-defined value.
- 14 d) If n is larger than 0:
 - 15 1) If the *block* is given:
 - 16 i) Let k be 0.
 - 17 ii) Call the *block* with an argument, which is an instance of the class `Integer`
18 whose value is k . Append the resulting value of this call to the receiver.
 - 19 iii) Increase k by 1. If k is equal to n , terminate this process. Otherwise, repeat
20 from Step d-1-ii.
 - 21 2) Otherwise, append the *obj* to the receiver n times.
 - 22 3) Return an implementation-defined value.

23 **15.2.12.5.16** `Array#initialize_copy`

```
24     initialize_copy(original)
```

25 **Visibility:** private

26 **Behavior:**

- 1 a) If the *original* is not an instance of the class **Array**, the behavior is unspecified.
- 2 b) Remove all the elements from the receiver.
- 3 c) Append all the elements of the *original* to the receiver, preserving their order.
- 4 d) Return an implementation-defined value.

5 **15.2.12.5.17 Array#join**

6 `join(sep=nil)`

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the *sep* is neither **nil** nor an instance of the class **String**, the behavior is unspecified.
- 10 b) Create an empty direct instance of the class **String** *S*.
- 11 c) For each element *X* of the receiver, in the indexing order:
 - 12 1) If the *sep* is not **nil**, and *X* is not the first element of the receiver, append the
13 content of the *sep* to *S*.
 - 14 2) If *X* is an instance of the class **String**, append the content of *X* to *S*.
 - 15 3) If *X* is an instance of the class **Array**:
 - 16 i) If *X* is the receiver, i.e. if the receiver contains itself, append an implementation-
17 defined sequence of characters to *S*.
 - 18 ii) Otherwise, append to *S* the content of the instance of the class **String** re-
19 turned as if by the invocation of the method `join` on *X* with the *sep* as the
20 argument.
 - 21 4) Otherwise, the behavior is unspecified.
- 22 d) Return *S*.

23 **15.2.12.5.18 Array#last**

24 `last(*args)`

25 **Visibility:** public

26 **Behavior:**

- 27 a) If the length of the *args* is 0:

- 1 1) If the receiver is empty, return **nil**.
- 2 2) Otherwise, return the last element of the receiver.
- 3 b) If the length of the *args* is 1:
- 4 1) If the only argument is not an instance of the class **Integer**, the behavior is
5 unspecified. Otherwise, let *n* be the value of the only argument.
- 6 2) If *n* is smaller than 0, raise a direct instance of the class **ArgumentError**.
- 7 3) Otherwise, let *N* be the smaller of *n* and the length of the receiver.
- 8 Return a newly created instance of the class **Array** which contains the last *N*
9 elements of the receiver, preserving their order.
- 10 c) If the length of *args* is larger than 1, raise a direct instance of the class **ArgumentError**.

11 **15.2.12.5.19 Array#length**

12 length

13 **Visibility:** public

14 **Behavior:** The method returns an instance of the class **Integer** whose value is the number
15 of elements of the receiver.

16 **15.2.12.5.20 Array#map!**

17 map!(&block)

18 **Visibility:** public

19 **Behavior:** Same as the method **collect!** (see §15.2.12.5.7).

20 **15.2.12.5.21 Array#pop**

21 pop

22 **Visibility:** public

23 **Behavior:**

- 24 a) If the receiver is empty, return **nil**.
- 25 b) Otherwise, remove the last element from the receiver and return that element.

1 **15.2.12.5.22 Array#push**

2 `push(*items)`

3 **Visibility:** public

4 **Behavior:**

5 a) For each element of the *items*, in the indexing order, append it to the receiver.

6 b) Return the receiver.

7 **15.2.12.5.23 Array#replace**

8 `replace(other)`

9 **Visibility:** public

10 **Behavior:** Same as the method `initialize_copy` (see §15.2.12.5.16).

11 **15.2.12.5.24 Array#reverse**

12 `reverse`

13 **Visibility:** public

14 **Behavior:** The method returns a newly created instance of the class `Array` which contains
15 all the elements of the receiver in the reverse order.

16 **15.2.12.5.25 Array#reverse!**

17 `reverse!`

18 **Visibility:** public

19 **Behavior:** The method reverses the order of the elements of the receiver and return the
20 receiver.

21 **15.2.12.5.26 Array#rindex**

22 `rindex(object=nil)`

23 **Visibility:** public

1 **Behavior:**

2 a) If the *object* is given:

3 1) For each element *E* of the receiver in the reverse indexing order, take the following
4 steps:

5 i) Invoke the method `==` on *E* with the *object* as the argument.

6 ii) If the resulting value is a trueish value, return the index of *E*.

7 2) Return **nil**.

8 b) Otherwise, the behavior is unspecified.

9 **15.2.12.5.27 Array#shift**

10 **shift**

11 **Visibility:** public

12 **Behavior:**

13 a) If the receiver is empty, return **nil**.

14 b) Otherwise, remove the first element from the receiver and return that element.

15 **15.2.12.5.28 Array#size**

16 **size**

17 **Visibility:** public

18 **Behavior:** Same as the method `length` (see §15.2.12.5.19).

19 **15.2.12.5.29 Array#slice**

20 **slice(*args)**

21 **Visibility:** public

22 **Behavior:** Same as the method `[]` (see §15.2.12.5.4).

23 **15.2.12.5.30 Array#unshift**

1 unshift(*items)

2 **Visibility:** public

3 **Behavior:**

- 4 a) For each element of the *items*, in the reverse indexing order, prepend it to the receiver.
- 5 b) Return the receiver.

6 15.2.13 Hash

7 15.2.13.1 General description

8 Instances of the class `Hash` represent hashes, which are sets of key/value pairs.

9 An instance of the class `Hash` which has no key/value pair is said to be **empty**. Instances of
10 the class `Hash` shall be empty when they are created by Step b of the method `new` of the class
11 `Class`.

12 An instance of the class `Hash` cannot contain more than one key/value pair for each key.

13 An instance of the class `Hash` has the following property:

14 **default value or proc:** Either of the followings:

- 15 • A default value, which is returned by the method `[]` when the specified key is not
16 found in the an instance of the class `Hash`.
- 17 • A default proc, which is called to generate the return value of the method `[]` when the
18 specified key is not found in the an instance of the class `Hash`.

19 An instance of the class `Hash` shall not have both a default value and a default proc simul-
20 taneously.

21 Given two keys K_1 and K_2 , the notation “ $K_1 \equiv K_2$ ” means that the keys are equivalent, i.e. all
22 of the following conditions hold:

- 23 • An invocation of the method `eq1?` on K_1 with K_2 as the only argument evaluates to a
24 trueish value.
- 25 • Let H_1 and H_2 be the results of invocations of the method `hash` on K_1 and K_2 , respectively.

26 H_1 and H_2 are the instances of the class `Integer` which represents the same integer.

27 A conforming processor may define a certain range of integers, and when the values of H_1 or
28 H_2 lies outside of this range, an implementation shall convert H_1 or H_2 to another instance
29 of the class `Integer` whose value is within the range. Let I_1 and I_2 be each of the resulting
30 instances respectively.

31 The values of I_1 and I_2 are the same integer.

1 If H_1 or H_2 is not an instance of the class `Integer`, whether $K_1 \equiv K_2$ is unspecified.

2 Note that $K_1 \equiv K_2$ is not equivalent to $K_2 \equiv K_1$.

3 **15.2.13.2 Direct superclass**

4 The class `Object`

5 **15.2.13.3 Included modules**

6 The following module is included in the class `Hash`.

- 7 • `Enumerable`

8 **15.2.13.4 Instance methods**

9 **15.2.13.4.1 Hash#==**

10 `==(other)`

11 **Visibility:** public

12 **Behavior:**

13 a) If the *other* is not an instance of the class `Hash`, the behavior is unspecified.

14 b) If all of the following conditions hold, return **true**:

- 15 • The receiver and the *other* have the same number of key/value pairs.
- 16 • For each key/value pair P in the receiver, the *other* has a corresponding key/value
17 pair Q which satisfies the following conditions:

18 — The key of $P \equiv$ the key of Q .

19 — An invocation of the method `==` on the value of P with the value of Q results
20 in a trueish value.

21 c) Otherwise, return **false**.

22 **15.2.13.4.2 Hash#[]**

23 `[](key)`

24 **Visibility:** public

25 **Behavior:**

- 1 a) If the receiver has a key/value pair P where the $key \equiv$ the key of P , return the value
2 of P .
- 3 b) Otherwise, invoke the method `default` on the receiver with the key as the argument
4 and return the resulting value.

5 **15.2.13.4.3 Hash#[]=**

6 `[]=(key, value)`

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the receiver has a key/value pair P where the $key \equiv$ the key of P , replace the value
10 of P with the $value$.
- 11 b) Otherwise:
- 12 1) If the key is a direct instance of the class `String`, create a copy of the key , i.e.
13 create a direct instance of the class `String` K whose content is the same as the
14 key .
- 15 2) If the key is not an instance of the class `String`, let K be the key .
- 16 3) If the key is an instance of a subclass of the class `String`, whether to create a copy
17 or not is implementation-defined.
- 18 4) Store a pair of K and the $value$ into the receiver.
- 19 c) Return the $value$.

20 **15.2.13.4.4 Hash#clear**

21 `clear`

22 **Visibility:** public

23 **Behavior:**

- 24 a) Remove all the key/value pairs from the receiver.
- 25 b) Return the receiver.

26 **15.2.13.4.5 Hash#default**

1 `default(*args)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the length of the *args* is larger than 1, raise a direct instance of the class `ArgumentError`.

5 b) If the receiver has the default value, return the value.

6 c) If the receiver has the default proc:

7 1) If the length of the *args* is 0, return **nil**.

8 2) If the length of the *args* is 1, invoke the method `call` on the default proc of the
9 receiver with two arguments, the receiver and the only element of the *args*. Return
10 the resulting value of this invocation.

11 d) Otherwise, return **nil**.

12 **15.2.13.4.6 Hash#default=**

13 `default=(value)`

14 **Visibility:** public

15 **Behavior:**

16 a) If the receiver has the default proc, remove the default proc.

17 b) Set the default value of the receiver to the *value*.

18 c) Return the *value*.

19 **15.2.13.4.7 Hash#default_proc**

20 `default_proc`

21 **Visibility:** public

22 **Behavior:**

23 a) If the receiver has the default proc, return the default proc.

24 b) Otherwise, return **nil**.

1 15.2.13.4.8 Hash#delete

2 delete(*key*, &*block*)

3 **Visibility:** public

4 **Behavior:**

5 a) If the receiver has a key/value pair *P* where the *key* \equiv the key of *P*, remove *P* from
6 the receiver and return the value of *P*.

7 b) Otherwise:

8 1) If the *block* is given, call the *block* with the *key* as the argument. Return the
9 resulting value of this call.

10 2) Otherwise, return **nil**.

11 15.2.13.4.9 Hash#each

12 each(&*block*)

13 **Visibility:** public

14 **Behavior:**

15 a) If the *block* is given, for each key/value pair of the receiver in an implementation defined
16 order:

17 1) Create a direct instance of the class **Array** which contains two elements, the key
18 and the value of the pair.

19 2) Call the *block* with the instance as an argument.

20 Return the receiver.

21 b) If the *block* is not given, the behavior is unspecified.

22 15.2.13.4.10 Hash#each_key

23 each_key(&*block*)

24 **Visibility:** public

25 **Behavior:**

- 1 a) If the *block* is given, for each key/value pair of the receiver, in an implementation-
2 defined order, call the *block* with the key of the pair as the argument. Return the
3 receiver.
- 4 b) If the *block* is not given, the behavior is unspecified.

5 **15.2.13.4.11 Hash#each_value**

6 `each_value(&block)`

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the *block* is given, call the *block* for each key/value pair of the receiver, with the
10 value as the argument, in an implementation-defined order. Return the receiver.
- 11 b) If the *block* is not given, the behavior is unspecified.

12 **15.2.13.4.12 Hash#empty?**

13 `empty?`

14 **Visibility:** public

15 **Behavior:**

- 16 a) If the receiver is empty, return **true**.
- 17 b) Otherwise, return **false**.

18 **15.2.13.4.13 Hash#has_key?**

19 `has_key?(key)`

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the receiver has a key/value pair *P* where the *key* \equiv the key of *P*, return **true**.
- 23 b) Otherwise, return **false**.

24 **15.2.13.4.14 Hash#has_value?**

1 `has_value?(value)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the receiver has a key/value pair whose value holds the following condition, return
5 **true**.

6 • An invocation of the method `==` on the value with the *value* as the argument result
7 in a trueish value.

8 b) Otherwise, return **false**.

9 **15.2.13.4.15 Hash#include?**

10 `include?(key)`

11 **Visibility:** public

12 **Behavior:** Same as the method `has_key?` (see §15.2.13.4.13).

13 **15.2.13.4.16 Hash#initialize**

14 `initialize(*args, &block)`

15 **Visibility:** private

16 **Behavior:**

17 a) If the *block* is given, and the length of the *args* is not 0, raise a direct instance of the
18 class `ArgumentError`.

19 b) If the *block* is given and the length of the *args* is 0, create a direct instance of the
20 class `Proc` which represents the *block* and set the default proc of the receiver to this
21 instance.

22 c) If the *block* is not given:

23 1) If the length of the *args* is 0, let *D* be **nil**.

24 2) If the length of the *args* is 1, let *D* be the only argument.

25 3) If the length of the *args* is larger than 1, raise a direct instance of the class
26 `ArgumentError`.

27 4) Set the default value of the receiver to *D*.

28 d) Return an implementation-defined value.

1 **15.2.13.4.17 Hash#initialize_copy**

2 `initialize_copy(original)`

3 **Visibility:** private

4 **Behavior:**

- 5 a) If the *original* is not an instance of the class `Hash`, the behavior is unspecified.
- 6 b) Remove all the key/value pairs from the receiver.
- 7 c) For each key/value pair *P* of the *original*, in an implementation-defined order, store *P*
8 in the receiver.
- 9 d) Remove the default value and the default proc from the receiver.
- 10 e) If the *original* has a default value, set the default value of the receiver to that value.
- 11 f) If the *original* has a default proc, set the default proc of the receiver to that proc.
- 12 g) Return an implementation-defined value.

13 **15.2.13.4.18 Hash#key?**

14 `key?(key)`

15 **Visibility:** public

16 **Behavior:** Same as the method `has_key?` (see §15.2.13.4.13).

17 **15.2.13.4.19 Hash#keys**

18 `keys`

19 **Visibility:** public

20 **Behavior:** The method returns a newly created instance of the class `Array` whose content
21 is the keys of the receiver. The order of the keys stored in somewhere are implementation-
22 defined.

23 **15.2.13.4.20 Hash#length**

24 `length`

1 **Visibility:** public

2 **Behavior:** The method returns an instance of the class `Integer` whose value is the number
3 of key/value pairs stored in the receiver.

4 **15.2.13.4.21 Hash#member?**

5 `member?(key)`

6 **Visibility:** public

7 **Behavior:** Same as the method `has_key?` (see §15.2.13.4.13).

8 **15.2.13.4.22 Hash#merge**

9 `merge(other, &block)`

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the *other* is not an instance of the class `Hash`, the behavior is unspecified.
- 13 b) Otherwise, create a direct instance of the class `Hash` *H* which has the same key/value
14 pairs as the receiver.
- 15 c) For each key/value pair *P* of the *other*, in an implementation-defined order:
- 16 1) If the *block* is given:
- 17 i) If *H* has the key/value pair *Q* where the key of *P* \equiv the key of *Q*, call the
18 *block* with three arguments, the key of *P*, the value of *Q*, and the value of *P*.
19 Store into *H* the key/value pair whose key is the key of *P* and whose value is
20 the resulting value of this call.
- 21 ii) Otherwise, store *P* in *H*.
- 22 2) If the *block* is not given, store *P* in *H*.
- 23 d) Return *H*.

24 **15.2.13.4.23 Hash#replace**

25 `replace(other)`

26 **Visibility:** public

27 **Behavior:** Same as the method `initialize_copy` (see §15.2.13.4.17).

1 **15.2.13.4.24 Hash#shift**

2 `shift`

3 **Visibility:** public

4 **Behavior:**

5 a) If the receiver is empty:

6 1) If the receiver has the default proc, invoke the method `call` on the default proc
7 with two arguments, the receiver and `nil`. Return the resulting value of this call.

8 2) If the receiver has the default value, return the value.

9 3) Otherwise, return `nil`.

10 b) Otherwise, choose a key/value pair P and remove P from the receiver. Return a newly
11 created instance of the class `Array` which contains two elements, the key and the value
12 of P .

13 Which pair is chosen is implementation-defined.

14 **15.2.13.4.25 Hash#size**

15 `size`

16 **Visibility:** public

17 **Behavior:** Same as the method `length` (see §15.2.13.4.20).

18 **15.2.13.4.26 Hash#store**

19 `store(key, value)`

20 **Visibility:** public

21 **Behavior:** Same as the method `[]=` (see §15.2.13.4.3).

22 **15.2.13.4.27 Hash#value?**

23 `value?(value)`

24 **Visibility:** public

25 **Behavior:** Same as the method `has_value?` (see §15.2.13.4.14).

1 15.2.13.4.28 Hash#values

2 values

3 **Visibility:** public

4 **Behavior:** The method returns a newly created instance of the class `Array` which contains
5 all the values of the receiver. The order of the values stored are implementation-defined.

6 15.2.14 Range

7 15.2.14.1 General description

8 Instances of the class `Range` represent ranges between two values, the start and end point.

9 An instance of the class `Range` has the following properties:

10 **start point:** The value at the start of the range.

11 **end point:** The value at the end of the range.

12 **exclusive flag:** If this is true, the end point is excluded from the range. Otherwise, the
13 end point is included in the range.

14 When the method `clone` (see §15.3.1.3.8) or the method `dup` (see §15.3.1.3.9) of the class `Kernel`
15 is invoked on an instance of the class `Range`, those properties shall be copied from the receiver
16 to the resulting value.

17 15.2.14.2 Direct superclass

18 The class `Object`

19 15.2.14.3 Included modules

20 The following module is included in the class `Range`.

- 21 • `Enumerable`

22 15.2.14.4 Instance methods

23 15.2.14.4.1 Range#==

24 `==(other)`

25 **Visibility:** public

26 **Behavior:**

- 27 a) If all of the following conditions hold, return **true**:

- 1 • the *other* is an instance of the class **Range**.
- 2 • Let *S* be the start point of the *other*. Invocation of the method **==** on the start
- 3 point of the receiver with *S* as the argument results in a trueish value.
- 4 • Let *E* be the end point of the *other*. Invocation of the method **==** on the end point
- 5 of the receiver with *E* as the argument results in a trueish value.
- 6 • The exclusive flag of the receiver and the one of the *other* are the same boolean
- 7 value.
- 8 b) Otherwise, return **false**.

9 **15.2.14.4.2 Range#===**

10 **===(obj)**

11 **Visibility:** public

12 **Behavior:**

- 13 a) If neither the start point of the receiver nor the end point of the receiver is an instance
- 14 of the class **Numeric**, the behavior is unspecified.
- 15 b) Invoke the method **<=>** on the start point of the receiver with the *obj* as the argument.
- 16 Let *S* be the result of this invocation.
 - 17 1) If *S* is not an instance of the class **Integer**, the behavior is unspecified.
 - 18 2) If the value of *S* is larger than 0, return **false**.
- 19 c) Invoke the method **<=>** on the *obj* with the end point of the receiver as the argument.
- 20 Let *E* be the result of this invocation.
 - 21 • If *E* is not an instance of the class **Integer**, the behavior is unspecified.
 - 22 • If the exclusive flag of the receiver is true, and the value of *E* is smaller than 0,
 - 23 return **true**.
 - 24 • If the exclusive flag of the receiver is false, and the value of *E* is smaller than or
 - 25 equal to 0, return **true**.
 - 26 • Otherwise, return **false**.

27 **15.2.14.4.3 Range#begin**

28 **begin**

29 **Visibility:** public

1 **Behavior:** The method returns the start point of the receiver.

2 15.2.14.4.4 Range#each

3 each(&block)

4 **Visibility:** public

5 **Behavior:**

- 6 a) If the *block* is not given, the behavior is unspecified.
- 7 b) If an invocation of the method `respond_to?` on the start point of the receiver with a
8 direct instance of the class `Symbol` whose name is `succ` as the argument results in a
9 falseish value, raise a direct instance of the class `TypeError`.
- 10 c) Let *V* be the start point of the receiver.
- 11 d) Invoke the method `<=>` on *V* with the end point of the receiver as the argument. Let
12 *C* be the resulting value.
- 13 1) If *C* is not an instance of the class `Integer`, the behavior is unspecified.
- 14 2) If the value of *C* is larger than 0, return the receiver.
- 15 3) If the value of *C* is 0:
- 16 i) If the exclusive flag of the receiver is true, return the receiver.
- 17 ii) If the exclusive flag of the receiver is false, call the *block* with *V* as the
18 argument, then, return the receiver.
- 19 e) Call the *block* with *V* as the argument.
- 20 f) Invoke the method `succ` on *V* with no argument, and let new *V* be the resulting value.
- 21 Continue processing from Step d.

22 15.2.14.4.5 Range#end

23 end

24 **Visibility:** public

25 **Behavior:** The method returns the end point of the receiver.

26 15.2.14.4.6 Range#exclude_end?

1 `exclude_end?`

2 **Visibility:** public

3 **Behavior:** If the exclusive flag of the receiver is true, return **true**. Otherwise, return **false**.

4 **15.2.14.4.7 Range#first**

5 `first`

6 **Visibility:** public

7 **Behavior:** Same as the method `begin` (see §15.2.14.4.3).

8 **15.2.14.4.8 Range#include?**

9 `include?(obj)`

10 **Visibility:** public

11 **Behavior:** Same as the method `===` (see §15.2.14.4.2).

12 **15.2.14.4.9 Range#initialize**

13 `initialize(left, right, exclusive=false)`

14 **Visibility:** public

15 **Behavior:**

16 a) Invoke the method `<=>` on the *left* with the *right* as the argument. If an exception
17 is raised and not handled during this invocation, raise a direct instance of the class
18 **ArgumentError**. If the result of this invocation is not an instance of the class **Integer**,
19 the behavior is unspecified.

20 b) If the *exclusive* is a trueish value, let *f* be true. Otherwise, let *f* be false.

21 c) Set the start point, end point, and exclusive flag of the receiver to the *left*, the *right*,
22 and *f*, respectively.

23 d) Return an implementation-defined value.

24 **15.2.14.4.10 Range#last**

1 last

2 **Visibility:** public

3 **Behavior:** Same as the method `end` (see §15.2.14.4.5).

4 15.2.14.4.11 Range#member?

5 member?(*obj*)

6 **Visibility:** public

7 **Behavior:** Same as the method `===` (see §15.2.14.4.2).

8 15.2.15 Regexp

9 15.2.15.1 General description

10 Instances of the class `Regexp` represent regular expressions, and have the following properties.

11 **pattern:** A *pattern* of the regular expression (see §15.2.15.4). The default value of this
12 property is empty.

13 If the value of this property is empty when a method is invoked on an instance of the class
14 `Regexp`, except for the invocation of the method `initialize`, the behavior of the invoked
15 method is unspecified.

16 **ignorecase:** A boolean value which denotes whether a match is performed in the case
17 insensitive manner. The default value of this property is false.

18 **multiline:** A boolean value which denotes whether the pattern “.” matches a *line-*
19 *terminator* (see §15.2.15.4). The default value of this property is false.

20 15.2.15.2 Direct superclass

21 The class `Object`

22 15.2.15.3 Constants

23 The following constants are defined in the class `Regexp`.

24 **IGNORECASE:** An instance of the class `Integer` whose value is 2^n , where the integer n
25 is an implementation-defined value. The value of this constant shall be different from that
26 of `MULTILINE` described below.

27 **MULTILINE:** An instance of the class `Integer` whose value is 2^m , where the integer m
28 is an implementation-defined value.

29 The above constants are used to set the `ignorecase` and `multiline` properties of an instance of
30 the class `Regexp` (see §15.2.15.7.1).

1 15.2.15.4 Patterns

2 Syntax

3 *pattern* ::

4 *alternative*₁

5 | *pattern*₁ | *alternative*₂

6 *alternative* ::

7 [empty]

8 | *alternative*₃ *term*

9 *term* ::

10 *anchor*

11 | *atom*₁

12 | *atom*₂ *quantifier*

13 *anchor* ::

14 *left-anchor* | *right-anchor*

15 *left-anchor* ::

16 *\A* | *^*

17 *right-anchor* ::

18 *\z* | *\$*

19 *quantifier* ::

20 * | + | ?

21 *atom* ::

22 *pattern-character*

23 | *grouping*

24 | *.*

25 | *atom-escape-sequence*

26 *pattern-character* ::

27 *source-character* **but not** *regexp-meta-character*

28 *regexp-meta-character* ::

29 | | *.* | * | + | *^* | ? | (|) | # | **

30 | *future-reserved-meta-character*

31 *future-reserved-meta-character* ::

32 [|] | { | }

```

1  grouping ::
2      ( pattern )

3  atom-escape-sequence ::
4      decimal-escape-sequence
5      | regexp-character-escape-sequence

6  decimal-escape-sequence ::
7      \ decimal-digit-without-zero

8  regexp-character-escape-sequence ::
9      regexp-escape-sequence
10     | regexp-non-escaped-sequence
11     | hex-escape-sequence
12     | regexp-octal-escape-sequence
13     | regexp-control-escape-sequence

14 regexp-escape-sequence ::
15     \ regexp-escaped-character

16 regexp-escaped-character ::
17     n | t | r | f | v | a | e

18 regexp-non-escaped-sequence ::
19     \ regexp-meta-character

20 regexp-octal-escape-sequence ::
21     octal-escape-sequence but not decimal-escape-sequence

22 regexp-control-escape-sequence ::
23     \ ( C - | c ) regexp-control-escaped-character

24 regexp-control-escaped-character ::
25     regexp-character-escape-sequence
26     | ?
27     | source-character but not ( \ | ? )

```

28 *future-reserved-meta-characters* are reserved for the extension of the pattern of regular expres-
29 sions.

30 Semantics

31 A regular expression selects specific substrings from a string called a target string according
32 to the pattern of the regular expression. If the pattern matches more than one substring, the
33 substring which begins earliest in the target string is selected. If there is more than one such
34 substring beginning at that point, the substring that has the highest priority, which is described

1 below, is selected. Each component of the pattern matches a substring of the target string as
2 follows:

3 A *pattern* matches the following substring:

- 4 a) If the *pattern* is an *alternative*₁, it matches the string which the *alternative*₁ matches.
- 5 b) If the *pattern* is a *pattern*₁ | *alternative*₂, it matches the string which either the *pattern*₁ or
6 the *alternative*₂ matches. The one which the *pattern*₁ matches has a higher priority.

7 EXAMPLE 1 `"ab".slice(/(a|ab)/)` returns "a", not "ab".

8 An *alternative* matches the following substring:

- 9 a) If the *alternative* is [empty], it matches an empty string.
- 10 b) If the *alternative* is an *alternative*₃ *term*, the *alternative* matches the substring whose first
11 part is matched with the *alternative*₃ and whose rest part is matched with the *term*.

12 If there is more than one such substring, the priority of the substrings is determined as
13 follows:

- 14 1) If there is more than one candidate which is matched with the *alternative*₃, a substring
15 whose first part is a candidate with a higher priority has a higher priority.

16 EXAMPLE 2 `"abc".slice(/(a|ab)(c|b)/)` returns "ab", not "abc". In this case, (a|ab)
17 is prior to (c|b).

- 18 2) If the first parts of substrings are the same, and if there is more than one candidate
19 which is matched with the *term*, a substring whose rest part is a candidate with a
20 higher priority has a higher priority.

21 EXAMPLE 3 `"abc".slice(/a(b|bc)/)` returns "ab", not "abc".

22 A *term* matches the following substring:

- 23 a) If the *term* is an *atom*₁, it matches the string which the *atom*₁ matches.
- 24 b) If the *term* is an *atom*₂ *quantifier*, it matches a string as follows:
 - 25 1) If the *quantifier* is *, it matches a sequence of zero or more strings which the *atom*₂
26 matches.
 - 27 2) If the *quantifier* is +, it matches a sequence of one or more strings which the *atom*₂
28 matches.
 - 29 3) If the *quantifier* is ?, it matches a sequence of zero or one strings which the *atom*₂
30 matches.

31 A longer sequence has a higher priority.

32 EXAMPLE 4 `"aaa".slice(/a*/)` returns "aaa", none of "", "a", and "aa".

1 c) If the *term* is an *anchor*, it matches the empty string at a specific position within the target
2 string S , as follows:

3 1) If the *anchor* is $\backslash\mathbf{A}$, it matches an empty string at the beginning of S .

4 2) If the *anchor* is \wedge , it matches an empty string at the beginning of S or just after a
5 *line-terminator* which is followed by at least one character.

6 3) If the *anchor* is $\backslash\mathbf{z}$, it matches an empty string at the end of S .

7 4) If the *anchor* is $\mathbf{\$}$, it matches an empty string at the end of S or just before a *line-*
8 *terminator*.

9 An *atom* matches the following substring:

10 a) If the *atom* is a *pattern-character*, it matches a single character C represented by the
11 *pattern-character*. If the *atom* is present in the pattern of an instance of the class **Regexp**
12 whose `ignorecase` property is true, it also matches a corresponding uppercase character of
13 C , if C is a lowercase character, or a corresponding lowercase character of C , if C is an
14 uppercase character.

15 b) If the *atom* is a *grouping*, it matches the string which the *grouping* matches.

16 c) If the *atom* is “.”, it matches any character except for a *line-terminator*. If the *atom* is
17 present in the pattern of an instance of the class **Regexp** whose `multiline` property is true,
18 it also matches a *line-terminator*.

19 d) If the *atom* is an *atom-escape-sequence*, it matches the string which the *atom-escape-*
20 *sequence* matches.

21 A *grouping* matches the substring which the *pattern* matches.

22 An *atom-escape-sequence* matches the following substring:

23 a) If the *atom-escape-sequence* is a *decimal-escape-sequence*, it matches the string which the
24 *decimal-escape-sequence* matches.

25 b) If the *atom-escape-sequence* is a *regexp-character-escape-sequence*, it matches a string of
26 length one, the content of which is the character which the *regexp-character-escape-sequence*
27 represents.

28 A *decimal-escape-sequence* matches the following substring:

29 a) Let i be an integer represented by *decimal-digit-without-zero*.

30 b) Let G be the i th *grouping* in the *pattern*, counted from 1, in the order of the occurrence of
31 “(” of *groupings* from the left of the *pattern*.

32 c) If the *decimal-escape-sequence* is present before G within the *pattern*, it does not match any
33 string.

34 d) If G matches any string, the *decimal-escape-sequence* matches the same string.

1 e) Otherwise, the *decimal-escape-sequence* does not match any string.

2 A *regex-character-escape-sequence* represents a character as follows:

3 • A *regex-escape-sequence* represents a character as shown in Table 1 in §8.7.6.3.3.

4 • A *regex-non-escaped-sequence* represents a *regex-meta-character*.

5 • A *hex-escape-sequence* represents a character as described in §8.7.6.3.3.

6 • A *regex-octal-escape-sequence* is interpreted in the same way as an *octal-escape-sequence*
7 (see §8.7.6.3.3).

8 • A *regex-control-escape-sequence* represents a character, the code of which is computed by
9 taking bitwise AND of 0x9f and the code of the character represented by the *regex-control-*
10 *escaped-character*, except when the *regex-control-escaped-character* is ?, in which case, the
11 *regex-control-escape-sequence* represents a character whose code is 127.

12 15.2.15.5 Matching process

13 A *pattern* P is considered to successfully match the given string S , if there exists a substring of
14 S (including S itself) which P matches.

15 When a numerical offset is specified, P is matched against the part of S which begins at the
16 offset and ends at the end of S . Note, however, that if the match succeeds, the string property of
17 the resulting instance of the class `MatchData` is S , not the part of S which begins at the offset,
18 as described below.

19 A matching process returns either an instance of the class `MatchData` (see §15.2.16) if the match
20 succeeds or `nil` if the match fails.

21 An instance of the class `MatchData` is created as follows:

22 a) Let B be the substring of S which P matched.

23 b) Create a direct instance of the class `MatchData`, and let M be the instance.

24 c) Set the string of M to S .

25 d) Create a new empty list L .

26 e) Let O be an ordered pair whose first element is B and whose second element is the offset
27 of the first character of B within S , counted from 0. Append O to L .

28 f) For each *grouping* G in P , in the order of the occurrence of its “(” within P , take the
29 following steps:

30 1) If G contributed to the match of P , let B be the substring which G matched. Let O
31 be an ordered pair whose first element is B and whose second element is the offset of
32 the first character of B within S , counted from 0. Append O to L .

33 2) Otherwise, append to L an ordered pair whose elements are both `nil`.

1 g) Set the match result of M to L .

2 h) M is the instance of the class `MatchData` returned by the matching process.

3 A matching process creates or updates a local variable binding with name “~”, which is specifically used by the method `Regexp.last_match` (see §15.2.15.6.3), as follows:

5 a) Let M be the value which the matching process returns.

6 b) If the binding for the name “~” can be resolved by the process described in §9.2 as if “~” were a *local-variable-identifier*, replace the value of the binding with M .

8 c) Otherwise, create a local variable binding with name “~” and value M in the uppermost non-block element of `[[local-variable-bindings]]` where the non-block element means the element which does not correspond to a *block*.

11 A conforming processor may name the binding other than “~”; however, it shall not be of the form *local-variable-identifier*.

13 15.2.15.6 Singleton methods

14 15.2.15.6.1 `Regexp.compile`

15 `Regexp.compile(*args)`

16 **Visibility:** public

17 **Behavior:** Same as the method `new` (see §15.2.3.3.3).

18 15.2.15.6.2 `Regexp.escape`

19 `Regexp.escape(string)`

20 **Visibility:** public

21 **Behavior:**

22 a) If the *string* is not an instance of the class `String`, the behavior is unspecified.

23 b) Let S be the content of the *string*.

24 c) Return a new instance of the class `String` whose content is same as S , except that every
25 occurrences of characters on the left of Table 4 are replaced with the corresponding
26 sequences of characters on the right of the Table 4.

27 15.2.15.6.3 `Regexp.last_match`

Table 4 – Regexp escaped characters

Characters replaced	Escaped sequence
0x0a	\n
0x09	\t
0x0d	\r
0x0c	\f
0x20	\0x20
#	\#
\$	\\$
(\(
)	\)
*	*
+	\+
-	\-
.	\.
?	\?
[\[
\	\\
]	\]
^	\^
{	\{
	\
}	\}

1 `Regexp.last_match(*index)`

2 **Visibility:** public

3 **Behavior:**

- 4 a) Search for a binding of a local variable with name “~” as described in §9.2 as if “~”
5 were a *local-variable-identifier*.
- 6 b) If the binding is found and its value is an instance of the class `MatchData`, let *M* be
7 the instance. Otherwise, return **nil**.
- 8 c) If the length of the *index* is 0, return *M*.
- 9 d) If the length of the *index* is larger than 1, raise a direct instance of the class `ArgumentError`.
- 10 e) If the length of the *index* is 1, let *A* be the only argument.
- 11 f) If *A* is not an instance of the class `Integer`, the behavior of the method is unspecified.

1 g) Let R be the result returned as if by invoking the method `[]` on M with A as the only
2 argument (see §15.2.16.3.1).

3 h) Return R .

4 **15.2.15.6.4 Regexp.quote**

5 `Regexp.quote`

6 **Visibility:** public

7 **Behavior:** Same as the method `escape` (see §15.2.15.6.2).

8 **15.2.15.7 Instance methods**

9 **15.2.15.7.1 Regexp#initialize**

10 `initialize(source, flag)`

11 **Visibility:** private

12 **Behavior:**

13 a) If the *source* is an instance of the class `Regexp`, let S be the pattern of the *source*.
14 If the *source* is an instance of the class `String`, let S be the content of the *source*.
15 Otherwise, the behavior is unspecified.

16 b) If S cannot be derived from the *pattern* (§15.2.15.4), raise a direct instance of the class
17 `RegexpError`.

18 c) Set the pattern of the receiver to S .

19 d) If the *flag* is an instance of the class `Integer`, let n be its value.

20 1) If computing bitwise AND of the value of `Regexp::IGNORECASE` and n results in
21 non-zero value, set the ignorecase property of the receiver to true.

22 2) If computing bitwise AND of the value of `Regexp::MULTILINE` and n results in
23 non-zero value, set the multiline property of the receiver to true.

24 e) If the *flag* is trueish value other than an instance of the class `Integer`, set the ignorecase
25 property of the receiver to true.

26 f) Return an implementation-defined value.

27 **15.2.15.7.2 Regexp#initialize_copy**

1 `initialize_copy(original)`

2 **Visibility:** private

3 **Behavior:**

- 4 a) If the *original* is not an instance of the class of the receiver, raise a direct instance of
- 5 the class `TypeError`.
- 6 b) Set the pattern of the receiver to the patten of the *original*.
- 7 c) Set the ignorecase property of the receiver to the ignorecase property of the *original*.
- 8 d) Set the multiline property of the receiver to the multiline property of the *original*.
- 9 e) Return an implementation-defined value.

10 **15.2.15.7.3 Regexp#==**

11 `==(other)`

12 **Visibility:** public

13 **Behavior:**

- 14 a) If the *other* is not an instance of the class `Regexp`, return **false**.
- 15 b) If the corresponding properties of the receiver and the *other* are the same, return **true**.
- 16 c) Otherwise, return **false**.

17 **15.2.15.7.4 Regexp#===**

18 `===(string)`

19 **Visibility:** public

20 **Behavior:**

- 21 a) If the *string* is not an instance of the class `String`, the behavior is unspecified.
- 22 b) Let *S* be the content of the *string*.
- 23 c) Match the pattern of the receiver against *S* (see §15.2.15.4 and Step 15.2.15.5). Let *M*
- 24 be the result of the matching process.
- 25 d) If *M* is an instance of the class `MatchData`, return **true**.
- 26 e) Otherwise, return **false**.

1 **15.2.15.7.5** `Regexp#=~`

2 `=~(string)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the *string* is not an instance of the class `String`, the behavior is unspecified.
- 6 b) Let *S* be the content of the *string*.
- 7 c) Match the pattern of the receiver against *S* (see §15.2.15.4 and Step 15.2.15.5). Let *M*
8 be the result of the matching process.
- 9 d) If *M* is `nil` return `nil`.
- 10 e) If *M* is an instance of the class `MatchData`, let *P* be first element of the match result
11 property of *M*, and let *i* be the second element of *P*.
- 12 f) Return an instance of the class `Integer` whose value is *i*.

13 **15.2.15.7.6** `Regexp#casefold?`

14 `casefold?`

15 **Visibility:** public

16 **Behavior:** The method returns the value of the `ignorecase` property of the receiver.

17 **15.2.15.7.7** `Regexp#match`

18 `match(string)`

19 **Visibility:** public

20 **Behavior:**

- 21 a) If the *string* is not an instance of the class `String`, the behavior is unspecified.
- 22 b) Let *S* be the content of the *string*.
- 23 c) Match the pattern of the receiver against *S* (see §15.2.15.4 and §15.2.15.5). Let *M* be
24 the result of the matching process.
- 25 d) Return *M*.

1 **15.2.15.7.8** `Regexp#source`

2 `source`

3 **Visibility:** public

4 **Behavior:** The method returns a direct instance of the class `String` whose content is the
5 pattern of the receiver.

6 **15.2.16** `MatchData`

7 **15.2.16.1** General description

8 Instances of the class `MatchData` represent results of successful matches of instances of the class
9 `Regexp` against instances of the class `String`.

10 An instance of the class `MatchData` has the properties called *string* and *match result*, which
11 are initialized as described in §15.2.15.5. Elements of the match result are indexed by integers
12 starting from 0.

13 Given an instance of the class `MatchData` M , the *matched substring*, *pre-match* and *post-*
14 *match* of M are defined as follows:

15 Let S be the string of M . Let F be the first element of the match result of M . Let B and O be
16 the first portion (the substring matched) and the second portion (offset of that substring) of F .
17 Let i be the sum of O and the length of B .

18 **matched substring:** The matched substring of M is B .

19 **pre-match:** The pre-match of M is a part of S , from the first up to, but not including the
20 O th character of S .

21 **post-match:** The post-match of M is a part of S , from the i th up to the last character of
22 S .

23 **15.2.16.2** Direct superclass

24 The class `Object`

25 **15.2.16.3** Instance methods

26 **15.2.16.3.1** `MatchData#[]`

27 `[] (*args)`

28 **Visibility:** public

29 **Behavior:** The method behaves as if the method `to_a` were invoked on the receiver (see
30 §15.2.16.3.12), and then, the method `[]` were invoked on the resulting instance of the class
31 `Array` with the same arguments passed to an invocation of this method (see §15.2.12.5.4).

1 15.2.16.3.2 MatchData#begin

2 begin(*index*)

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the *index* is not an instance of the class `Integer`, the behavior is unspecified.
- 6 b) Let *L* be the match result of the receiver, and let *i* be the value of the *index*.
- 7 c) If *i* is smaller than 0 or equal to, or larger than the number of elements of *L*, raise a
8 direct instance of the class `IndexError`.
- 9 d) Otherwise, return the second portion of the *i*th element of *L*.

10 15.2.16.3.3 MatchData#captures

11 captures

12 **Visibility:** public

13 **Behavior:**

- 14 a) Let *L* be the match result of the receiver.
- 15 b) Create an empty direct instance of the class `Array` *A*.
- 16 c) Except for the first element, for each element *e* of *L*, in the same order in the list,
17 append to *A* a direct instance of the class `String` whose content is the first portion of
18 *e*.
- 19 d) Return *A*.

20 15.2.16.3.4 MatchData#end

21 end(*index*)

22 **Visibility:** public

23 **Behavior:**

- 24 a) If the *index* is not an instance of the class `Integer`, the behavior is unspecified.
- 25 b) Let *L* be the match result of the receiver, and let *i* be the value of the *index*.

- 1 c) If i is smaller than 0 or equal to, or larger than the number of elements of L , raise a
- 2 direct instance of the class `IndexError`.
- 3 d) Let F and S be the first and the second portions of the i th element of L .
- 4 e) If F is `nil`, return `nil`.
- 5 f) Otherwise, let f be the length of F . Return an instance of the class `Integer` whose
- 6 value is the sum of S and f .

7 **15.2.16.3.5 MatchData#initialize_copy**

8 `initialize_copy(original)`

9 **Visibility:** private

10 **Behavior:**

- 11 a) If the *original* is not an instance of the class of the receiver, raise a direct instance of
- 12 the class `TypeError`.
- 13 b) Set the string property of the receiver to the string property of the *original*.
- 14 c) Set the match result property of the receiver to the match result property of the
- 15 *original*.
- 16 d) Return an implementation-defined value.

17 **15.2.16.3.6 MatchData#length**

18 `length`

19 **Visibility:** public

20 **Behavior:** The method returns the number of elements of the match result of the receiver.

21 **15.2.16.3.7 MatchData#offset**

22 `offset(index)`

23 **Visibility:** public

24 **Behavior:**

- 25 a) If the *index* is not an instance of the class `Integer`, the behavior is unspecified.
- 26 b) Let L be the match result of the receiver, and let i be the value of the *index*.

- 1 c) If i is smaller than 0 or equal to, or larger than the number of elements of L , raise a
2 direct instance of the class `IndexError`.
- 3 d) Let S and b be the first and the second portions of the i th element of L . Let e be the
4 sum of b and the length of S .
- 5 e) Return a new instance of the class `Array` which contains two instances of the class
6 `Integer`, the one whose value is b and the other whose value is e , in this order.

7 **15.2.16.3.8 MatchData#post_match**

8 `post_match`

9 **Visibility:** public

10 **Behavior:** The method returns an instance of the class `String` the content of which is the
11 post-match of the receiver.

12 **15.2.16.3.9 MatchData#pre_match**

13 `pre_match`

14 **Visibility:** public

15 **Behavior:** The method returns an instance of the class `String` the content of which is the
16 pre-match of the receiver.

17 **15.2.16.3.10 MatchData#size**

18 `size`

19 **Visibility:** public

20 **Behavior:** Same as the method `length` (see §15.2.16.3.6).

21 **15.2.16.3.11 MatchData#string**

22 `string`

23 **Visibility:** public

24 **Behavior:** The method returns an instance of the class `String` the content of which is the
25 string of the receiver.

26 **15.2.16.3.12 MatchData#to_a**

1 to_a

2 **Visibility:** public

3 **Behavior:**

4 a) Let L be the match result of the receiver.

5 b) Create an empty direct instance of the class `Array` A .

6 c) For each element e of L , in the same order in the list, append to A an instance of the
7 class `String` whose content is the first portion of e .

8 d) Return A .

9 **15.2.16.3.13 MatchData#to_s**

10 to_s

11 **Visibility:** public

12 **Behavior:** The method returns an instance of the class `String` the content of which is the
13 matched substring of the receiver.

14 **15.2.17 Proc**

15 **15.2.17.1 General description**

16 Instances of the class `Proc` represent *blocks*.

17 An instance of the class `Proc` has the following property.

18 **block:** The block represented by the instance.

19 **15.2.17.2 Direct superclass**

20 The class `Object`

21 **15.2.17.3 Singleton methods**

22 **15.2.17.3.1 Proc.new**

23 `Proc.new(&block)`

24 **Visibility:** public

25 **Behavior:**

- 1 a) If the *block* is given, let *B* be the *block*.
- 2 b) Otherwise:
 - 3 1) If the top of `[[block]]` is *block-not-given*, raise a direct instance of the class `ArgumentError`.
 - 4 2) Otherwise, let *B* be the top of `[[block]]`.
- 5 c) Create a new direct instance of the class `Proc` which has *B* as its *block*.
- 6 d) Return the instance.

7 15.2.17.4 Instance methods

8 15.2.17.4.1 Proc#[]

9 `[] (*args)`

10 **Visibility:** public

11 **Behavior:** Same as the method `call` (see §15.2.17.4.3).

12 15.2.17.4.2 Proc#arity

13 `arity`

14 **Visibility:** public

15 **Behavior:** Let *B* be the block represented by the receiver.

- 16 a) If a *block-parameter* is omitted in *B*, return an instance of the class `Integer` whose
17 value is implementation-defined.
- 18 b) If a *block-parameter* is present in *B*:
 - 19 1) If a *block-parameter-list* is omitted in the *block-parameter*, return an instance of
20 the class `Integer` whose value is 0.
 - 21 2) If a *block-parameter-list* is present in the *block-parameter*:
 - 22 i) If the *block-parameter-list* is of the form *left-hand-side*, return an instance of
23 the class `Integer` whose value is 1.
 - 24 ii) If the *block-parameter-list* is of the form *multiple-left-hand-side*:
 - 25 I) If the *multiple-left-hand-side* is of the form *grouped-left-hand-side*, return
26 an instance of the class `Integer` whose value is implementation-defined.

- 1 II) If the *multiple-left-hand-side* is of the form *packing-left-hand-side*, return
 2 an instance of the class `Integer` whose value is -1 .
- 3 III) Otherwise, let n be the number of *multiple-left-hand-side-items* of the
 4 *multiple-left-hand-side*.
- 5 IV) If the *multiple-left-hand-side* ends with a *packing-left-hand-side*, return
 6 an instance of the class `Integer` whose value is $-(n+1)$.
- 7 V) Otherwise, return an instance of the class `Integer` whose value is n .

8 **15.2.17.4.3 Proc#call**

9 `call(*args)`

10 **Visibility:** public

11 **Behavior:** Let B be the block of the receiver. Let L be an empty list.

- 12 a) Append each element of the *args*, in the indexing order, to L .
- 13 b) Call B with L as the arguments (see §11.3.3). Let V be the result of the call.
- 14 c) Return V .

15 **15.2.17.4.4 Proc#clone**

16 `clone`

17 **Visibility:** public

18 **Behavior:**

- 19 a) Create a direct instance of the class of the receiver which has no bindings of instance
 20 variables. Let O be the newly created instance.
- 21 b) For each binding B of the instance variables of the receiver, create a variable binding
 22 with the same name and value as B in the set of bindings of instance variables of O .
- 23 c) If the receiver is associated with an eigenclass, let E_o be the eigenclass, and take the
 24 following steps:
- 25 1) Create an eigenclass whose direct superclass is the direct superclass of E_o . Let E_n
 26 be the eigenclass.
- 27 2) For each binding B_{v1} of the constants of E_o , create a variable binding with the
 28 same name and value as B_{v1} in the set of bindings of constants of E_n .
- 29 3) For each binding B_{v2} of the class variables of E_o , create a variable binding with
 30 the same name and value as B_{v2} in the set of bindings of class variables of E_n .

- 1 4) For each binding B_m of the instance methods of E_o , create a method binding with
 2 the same name and value as B_m in the set of bindings of instance methods of E_n .
- 3 5) Associate O with E_n .
- 4 d) Set the block of O to the block of the receiver.
- 5 e) Return O .

6 **15.2.17.4.5 Proc#dup**

7 dup

8 **Visibility:** public

9 **Behavior:**

- 10 a) Create a direct instance of the class of the receiver which has no bindings of instance
 11 variables. Let O be the newly created instance.
- 12 b) Set the block of O to the block of the receiver.
- 13 c) Return O .

14 **15.2.18 Struct**

15 **15.2.18.1 General description**

16 The class **Struct** is a generator of a structure type which is a class defining a set of fields
 17 and methods for accessing these fields. Fields are indexed by integers starting from 0 (see
 18 §15.2.18.3.1). An instance of a generated class has values for the set of fields. Those values can
 19 be referenced and updated with accessor methods for their fields.

20 **15.2.18.2 Direct superclass**

21 The class **Object**

22 **15.2.18.3 Singleton methods**

23 **15.2.18.3.1 Struct.new**

24 **Struct.new**(*string*, **symbol_list*)

25 **Visibility:** public

26 **Behavior:** The method creates a class defining a set of fields and accessor methods for
 27 these fields.

28 When the method is invoked, take the following steps:

- 1 a) Create a direct instance of the class `Class` which has the class `Struct` as its direct
2 superclass. Let C be that class.
- 3 b) If the *string* is not an instance of the class `String` or the class `Symbol`, the behavior is
4 unspecified.
- 5 c) If the *string* is an instance of the class `String`, let N be the content of the instance.
 - 6 1) If N is not of the form *constant-identifier*, raise a direct instance of the class
7 `ArgumentError`.
 - 8 2) Otherwise,
 - 9 i) If the binding with name N exists in the set of bindings of constants in the
10 class `Struct`, replace the value of the binding with C .
 - 11 ii) Otherwise, create a constant binding in the class `Struct` with name N and
12 value C .
- 13 d) If the *string* is an instance of the class `Symbol`, prepend the instance to the *symbol_list*.
- 14 e) Let i be 0.
- 15 f) For each element S of the *symbol_list*, take the following steps:
 - 16 1) Let N be the name designated by S .
 - 17 2) Define a field, which is named N and is indexed by i , in C .
 - 18 3) If N is of the form *local-variable-identifier* or *constant-identifier*:
 - 19 i) Define a method named N in C which takes no arguments, and when invoked,
20 returns the value of the field named N .
 - 21 ii) Define a method named $N=$ (i.e. N postfixed by “=”) in C which takes one
22 argument, and when invoked, sets the value of the field named N to the given
23 argument and returns the argument.
 - 24 4) Increment i by 1.
- 25 g) Return C .

26 Singleton methods of classes created by the `Struct.new`

27 Classes created by the method `Struct.new` are equipped with public singleton methods `new`,
28 `[]=`, and `members`. The following describes those methods, assuming that the name of a class
29 created by the method `Struct.new` is C .

30 `C.new(*args)`

1 **Visibility:** public

2 **Behavior:**

- 3 a) Create a direct instance of the class with the set of fields the receiver defines. Let I be
4 the instance.
- 5 b) Invoke the method `initialize` on I with the $args$ as the list of arguments.
- 6 c) Return I .

7 $C . [] (*args)$

8 **Visibility:** public

9 **Behavior:** Same as the method `new` described above.

10 $C . members$

11 **Visibility:** public

12 **Behavior:**

- 13 a) Create a direct instance of the class `Array` A . For each field of the receiver, in the
14 indexing order of the fields, create a direct instance of the class `String` whose content
15 is the name of the field and append the instance to A .
- 16 b) Return A .

17 **15.2.18.4 Instance methods**

18 **15.2.18.4.1 Struct#==**

19 $==(other)$

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the $other$ and the receiver is the same object, return **true**.
- 23 b) If the class of the $other$ and that of the receiver are different, return **false**.
- 24 c) Otherwise, for each field named f of the receiver, take the following steps:
- 25 1) Let R and O be the values of the fields named f of the receiver and the $other$
26 respectively.

- 1 2) If R and O are not the same object,
- 2 i) Invoke the method `==` on R with O as the only argument. Let V be the
- 3 resulting value of the invocation.
- 4 ii) If V is a falseish value, return **false**.
- 5 d) Return **true**.

6 **15.2.18.4.2 Struct#[]**

7 [](*name*)

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the *name* is an instance of the class `Symbol` or the class `String`:
- 11 1) Let N be the name designated by the *name*.
- 12 2) If the receiver has the field named N , return the value of the field.
- 13 3) Otherwise, let S be an instance of the class `Symbol` with name N and raise a direct
- 14 instance of the class `NameError` which has S as its name property.
- 15 b) If the *name* is an instance of the class `Integer`, let i be the value of the *name*. Let n
- 16 be the number of the fields of the receiver.
- 17 1) If i is negative, replace i with $n + i$.
- 18 2) If i is still negative or i equal or larger than n , raise a direct instance of the class
- 19 `IndexError`.
- 20 3) Otherwise, return the value of the field whose index is i .
- 21 c) Otherwise, the behavior of the method is unspecified.

22 **15.2.18.4.3 Struct#[]=**

23 []=(*name*, *obj*)

24 **Visibility:** public

25 **Behavior:**

- 26 a) If the *name* is an instance of the class `Symbol` or an instance of the class `String`:

- 1) Let N be the name designated by the *name*.
 - 2) If the receiver has the field named N ,
 - i) Replace the value of the field with the *obj*,
 - ii) Return the *obj*.
 - 3) Otherwise, let S be an instance of the class `Symbol` with name N and raise a direct instance of the class `NameError` which has S as its name property.
- b) If the *string* is an instance of the class `Integer`, let i be the value of the *name*. Let n be the number of the fields of the receiver.
- 1) If i is negative, replace i with $n + i$.
 - 2) If i is still negative or i equal or larger than n , raise a direct instance of the class `IndexError`.
 - 3) Otherwise,
 - i) Replace the value of the field whose index is i with the *obj*
 - ii) Return the *obj*.
- c) Otherwise, the behavior of the method is unspecified.

15.2.18.4.4 `Struct#each`

```
each(&block)
```

Visibility: public

Behavior:

- a) If the *block* is not given, the behavior is unspecified.
- b) For each field of the receiver, in the indexing order, call the *block* with the value of the field as the only argument.
- c) Return the receiver.

15.2.18.4.5 `Struct#each_pair`

```
each_pair(&block)
```

Visibility: public

1 **Behavior:**

- 2 a) If the *block* is not given, the behavior is unspecified.
- 3 b) For each field of the receiver, in the indexing order, take the following steps:
- 4 1) Let *N* and *V* be the name and the value of the field respectively. Let *S* be an
5 instance of the class `Symbol` with name *N*.
- 6 2) Call the *block* with the list of arguments which contains *S* and *V* in this order.
- 7 c) Return the receiver.

8 **15.2.18.4.6 Struct#members**

9 **members**

10 **Visibility:** public

11 **Behavior:** Same as the method `members` described in §15.2.18.3.1.

12 **15.2.18.4.7 Struct#select**

13 **select(&block)**

14 **Visibility:** public

15 **Behavior:**

- 16 a) If the *block* is not given, the behavior is unspecified.
- 17 b) Create an empty direct instance of the class `Array`. Let *A* be the instance.
- 18 c) For each field of the receiver, in the indexing order, take the following steps:
- 19 1) Let *V* be the value of the field.
- 20 2) Call the *block* with *V* as the only argument. Let *R* be the resulting value of the
21 call.
- 22 3) If *R* is a trueish value, append *V* to *A*.
- 23 d) Return *A*.

24 **15.2.18.4.8 Struct#initialize**

1 `initialize(*args)`

2 **Visibility:** private

3 **Behavior:** Let N_a be the length of the *args*, and let N_f be the number of the fields of the
4 receiver.

- 5 a) If N_a is larger than N_f , raise a direct instance of the class `ArgumentError`.
- 6 b) Otherwise, for each field *f* of the receiver, let *i* be the index of *f*, and set the value of *f*
7 to the *i*th element of the *args*, or to `nil` when *i* is equal to or larger than N_a .
- 8 c) Return an implementation-defined value.

9 **15.2.18.4.9 Struct#initialize_copy**

10 `initialize_copy(original)`

11 **Visibility:** private

12 **Behavior:**

- 13 a) If the receiver and the *original* are the same object, return an implementation-defined
14 value.
- 15 b) If the *original* is not an instance of the class of the receiver, raise a direct instance of
16 the class `TypeError`.
- 17 c) If the number of the fields of the receiver and the number of the fields of the *original*
18 are different, raise a direct instance of the class `TypeError`.
- 19 d) For each field *f* of the *original*, let *i* be the index of *f*, and set the value of the *i*th field
20 of the receiver to the value of *f*.
- 21 e) Return an implementation-defined value.

22 **15.2.19 Time**

23 **15.2.19.1 General description**

24 Instances of the class `Time` represent dates and times.

25 An instance of the class `Time` holds the following data.

26 **Microseconds:** Microseconds since January 1, 1970 00:00 UTC. Microseconds is an inte-
27 ger whose range is implementation-defined. If an out of range value is given as microsec-
28 onds when creating an instance of the class `Time`, a direct instance of either of the class
29 `ArgumentError` or the class `RangeError` shall be raised.

30 **Time zone:** The time zone.

1 **15.2.19.2 Direct superclass**

2 The class `Object`

3 **15.2.19.3 Time computation**

4 Mathematical functions introduced in this subclass are used throughout the descriptions in
5 §15.2.19. These functions are assumed to compute exact mathematical results using mathemat-
6 ical real numbers.

7 Leap seconds are ignored in this document. However, a conforming processor may support leap
8 seconds in an implementation-defined way.

9 **15.2.19.3.1 Day**

The number of microseconds of a day is computed as follows:

$$\text{MicroSecPerDay} = 24 \times 60 \times 60 \times 10^6$$

The number of days since January 1, 1970 00:00 UTC which corresponds to microseconds t is computed as follows:

$$\text{Day}(t) = \text{floor} \left(\frac{t}{\text{MicroSecPerDay}} \right)$$

$\text{floor}(t)$ = The integer x such that $x \leq t < x + 1$

The weekday which corresponds to microseconds t is computed as follows:

$$\text{WeekDay}(t) = (\text{Day}(t) + 4) \text{ modulo } 7$$

10 **15.2.19.3.2 Year**

11 A year has 365 days, except for leap years, which have 366 days. Leap years are those which
12 are either:

- 13 • divisible by 4 and not divisible by 100, or
- 14 • divisible by 400.

The number of days from January 1, 1970 00:00 UTC to the beginning of a year y is computed as follows:

$$\text{DayFromYear}(y) = 365 \times (y - 1970) + \text{floor} \left(\frac{y - 1969}{4} \right) - \text{floor} \left(\frac{y - 1901}{100} \right) + \text{floor} \left(\frac{y - 1601}{400} \right)$$

Microseconds elapsed since January 1, 1970 00:00 UTC until the beginning of y is computed as follows:

$$\text{MicroSecFromYear}(y) = \text{DayFromYear}(y) \times \text{MicroSecPerDay}$$

The year number y which corresponds to microseconds t measured from January 1, 1970 00:00 UTC is computed as follows.

$$YearFromTime(t) = y \text{ such that, } MicroSecFromYear(y - 1) < t \leq MicroSecFromYear(y)$$

The number of days from the beginning of the year for the given microseconds t is computed as follows.

$$DayWithinYear(t) = Day(t) - DayFromYear(YearFromTime(t))$$

1 15.2.19.3.3 Month

- 2 Months have usual number of days. Leap years have the extra day in February. Each month is
 3 identified by the number in the range 1 to 12, in the order from January to December.

The month number which corresponds to microseconds t measured from January 1, 1970 00:00 UTC is computed as follows.

$$MonthFromTime(t) = \begin{cases} 1 & \text{if } 0 \leq DayWithinYear(t) < 31 \\ 2 & \text{if } 31 \leq DayWithinYear(t) < 59 \\ 3 & \text{if } 59 + LeapYear(t) \leq DayWithinYear(t) < 90 + LeapYear(t) \\ 4 & \text{if } 90 + LeapYear(t) \leq DayWithinYear(t) < 120 + LeapYear(t) \\ 5 & \text{if } 120 + LeapYear(t) \leq DayWithinYear(t) < 151 + LeapYear(t) \\ 6 & \text{if } 151 + LeapYear(t) \leq DayWithinYear(t) < 180 + LeapYear(t) \\ 7 & \text{if } 181 + LeapYear(t) \leq DayWithinYear(t) < 212 + LeapYear(t) \\ 8 & \text{if } 212 + LeapYear(t) \leq DayWithinYear(t) < 243 + LeapYear(t) \\ 9 & \text{if } 243 + LeapYear(t) \leq DayWithinYear(t) < 273 + LeapYear(t) \\ 10 & \text{if } 273 + LeapYear(t) \leq DayWithinYear(t) < 304 + LeapYear(t) \\ 11 & \text{if } 304 + LeapYear(t) \leq DayWithinYear(t) < 334 + LeapYear(t) \\ 12 & \text{if } 334 + LeapYear(t) \leq DayWithinYear(t) < 365 + LeapYear(t) \end{cases}$$

$$LeapYear(t) = \begin{cases} 1 & \text{if } YearFromTime(t) \text{ is a leap year} \\ 0 & \text{otherwise} \end{cases}$$

1 15.2.19.3.4 Days of month

The day of the month which corresponds to microseconds t measured from January 1, 1970 00:00 UTC is computed as follows.

$$DayWithinMonth(t) = \begin{cases} DayWithinYear(t) + 1 & \text{if } MonthFromTime(t) = 1 \\ DayWithinYear(t) - 30 & \text{if } MonthFromTime(t) = 2 \\ DayWithinYear(t) - 58 - LeapYear(t) & \text{if } MonthFromTime(t) = 3 \\ DayWithinYear(t) - 89 - LeapYear(t) & \text{if } MonthFromTime(t) = 4 \\ DayWithinYear(t) - 119 - LeapYear(t) & \text{if } MonthFromTime(t) = 5 \\ DayWithinYear(t) - 150 - LeapYear(t) & \text{if } MonthFromTime(t) = 6 \\ DayWithinYear(t) - 180 - LeapYear(t) & \text{if } MonthFromTime(t) = 7 \\ DayWithinYear(t) - 211 - LeapYear(t) & \text{if } MonthFromTime(t) = 8 \\ DayWithinYear(t) - 242 - LeapYear(t) & \text{if } MonthFromTime(t) = 9 \\ DayWithinYear(t) - 272 - LeapYear(t) & \text{if } MonthFromTime(t) = 10 \\ DayWithinYear(t) - 303 - LeapYear(t) & \text{if } MonthFromTime(t) = 11 \\ DayWithinYear(t) - 333 - LeapYear(t) & \text{if } MonthFromTime(t) = 12 \end{cases}$$

2 15.2.19.3.5 Hours, Minutes, and Seconds

The number of microseconds in an hour, a minute, a second are as follows:

$$\begin{aligned} MicroSecPerHour &= 60 \times 60 \times 10^6 \\ MicroSecPerMinute &= 60 \times 10^6 \\ MicroSecPerSecond &= 10^6 \end{aligned}$$

The hour, the minute, and the second which correspond to microseconds t measured from January 1, 1970 00:00 UTC are computed as follows.

$$\begin{aligned} HourFromTime(t) &= \text{floor} \left(\frac{t}{MicroSecPerHour} \right) \text{ modulo } 24 \\ MinuteFromTime(t) &= \text{floor} \left(\frac{t}{MicroSecPerMinute} \right) \text{ modulo } 60 \\ SecondFromTime(t) &= \text{floor} \left(\frac{t}{MicroSecPerSecond} \right) \text{ modulo } 60 \end{aligned}$$

3 15.2.19.4 Time zone and Local time

4 The current time zone is determined from time zone information provided by the underlying
5 system. If the system does not provide information on the current time zone, the time zone of
6 an instance of the class `Time` is implementation-defined.

The local time for an instance of the class `Time` is computed from its microseconds t and time zone z as follows.

$$\begin{aligned} LocalTime &= t + ZoneOffset(z) \\ ZoneOffset(z) &= \text{UTC offset of } z \text{ measured in microseconds} \end{aligned}$$

1 **15.2.19.5 Daylight saving time**

2 On system where it is possible to determine the daylight saving time for each time zone, a
3 conforming processor should adjust the microseconds of an instance of the class `Time` if that
4 microseconds falls within the daylight saving time of the time zone of the instance. An algorithm
5 used for the adjustment is implementation-defined.

6 **15.2.19.6 Singleton methods**

7 **15.2.19.6.1 Time.at**

8 `Time.at(*args)`

9 **Visibility:** public

10 **Behavior:**

11 a) If the length of the *args* is 0 or larger than 2, raise a direct instance of the class
12 `ArgumentError`.

13 b) If the length of the *args* is 1, let *A* be the only argument.

14 1) If *A* is an instance of the class `Time`, return a new instance of the class `Time` which
15 represents the same time and has the same time zone as *A*.

16 2) If *A* is an instance of the class `Integer` or an instance of the class `Float`:

17 i) If *A* is an instance of the class `Integer`, let N_S be the value of *A*. Let N_M be
18 0.

19 ii) If *A* is an instance of the class `Float`, let *F* be the value of *A*. Let N_S be the
20 largest integer such that $N_S \leq F$. Let N_M be an integer which is the result
21 of computing $(F - N_S) \times 10^6$, rounded off the first decimal place.

22 iii) Create a direct instance of the class `Time` which represents the time at $N_S \times$
23 $10^6 + N_M$ microseconds since January 1, 1970 00:00 UTC, with the current
24 local time zone.

25 iv) Return the resulting instance.

26 3) Otherwise, the behavior is unspecified.

27 c) If the length of the *args* is 2, let *S* and *M* be the first and second element of the *args*.

28 1) i) If *S* is an instance of the class `Integer`, let N_S be the value of *S*.

29 ii) If *S* is an instance of the class `Float`, let *F* be the value of *S*. If *F* is positive,
30 let N_S be the largest integer such that $N_S \leq F$. Otherwise, let N_S be the
31 smallest integer such that $N_S \geq F$.

32 iii) Otherwise, the behavior is unspecified.

- 1 2) Compute an integer which corresponds to M in the same way as S as described
2 in Step c-1-i and c-1-ii. Let N_M be the integer.
- 3 3) Create a direct instance of the class `Time` which represents the time at $N_S \times 10^6 +$
4 N_M microseconds since January 1, 1970 00:00 UTC, with the current local time
5 zone.
- 6 4) Return the resulting instance.

7 **15.2.19.6.2 Time.gm**

8 `Time.gm(year, month=1, day=0, hour=0, min=0, sec=0, usec=0)`

9 **Visibility:** public

10 **Behavior:**

- 11 a) Compute an integer value for the *year*, *day*, *hour*, *min*, *sec*, and *usec* as described
12 below. Let Y , D , H , Min , S , and U be integers thus converted.

13 An integer I is determined from the given object O as follows:

- 14 1) If O is an instance of the class `Integer`, let I be the value of O .
- 15 2) If O is an instance of the class `Float`, let I be the integral part of the value of O .
- 16 3) If O is an instance of the class `String`:
 - 17 i) If the content of O is a sequence of *decimal-digits*, let I be the value of those
18 sequence of digits computed using base 10.
 - 19 ii) Otherwise, the behavior is unspecified.
- 20 4) Otherwise, the behavior is unspecified.

- 21 b) Compute an integer value from the *month* as follows:

- 22 1) If the *month* is not an instance of the class `String`, the behavior is unspecified.
- 23 2) If the content of the *month* is the same as one of the names of the months in the
24 lower row on Table 5, ignoring the differences in case, let Mon be the integer which
25 corresponds to the *month* in the upper row on the same table.
- 26 3) If the first character of the *month* is *decimal-digit*, compute an integer value from
27 the *month* as in Step a. Let Mon be the resulting integer.
- 28 4) Otherwise, raise a direct instance of the class `ArgumentError`.

- 29 c) If Y is an integer such that $0 \leq Y \leq 38$, replace Y with $2000 + Y$.

- 1 d) If Y is an integer such that $69 \leq Y \leq 138$, replace Y with $1900 + Y$.
- 2 e) If each integer computed above is outside the range as listed below, raise a direct
3 instance of the class `ArgumentError`.

- 4 • $1 \leq Mon \leq 12$
- 5 • $1 \leq D \leq 31$
- 6 • $0 \leq H \leq 23$
- 7 • $0 \leq Min \leq 59$
- 8 • $0 \leq S \leq 60$

9 Whether any conditions are placed on Y is implementation-defined.

- 10 f) Let t be an integer which satisfies all of the following equations.

- 11 • $YearFromTime(t) = Y$
- 12 • $MonthFromTime(t) = Mon$
- 13 • $DayWithinMonth(t) = 1$

- g) Compute microseconds T as follows.

$$T = t + D \times MicroSecPerDay + H \times MicroSecPerHour + \\ Min \times MicroSecPerMinute + S \times 10^6 + U$$

- 14 h) Create a direct instance of the class `Time` which represents the time at T since January
15 1, 1970 00:00 UTC, with the UTC time zone.
- 16 i) Return the resulting instance.

Table 5 – The names of months and corresponding integer

1	2	3	4	5	6	7	8	9	10	11	12
Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec

17 **15.2.19.6.3 Time.local**

18 `Time.local(year, month=1, day=0, hour=0, min=0, sec=0, usec=0)`

19 **Visibility:** public

20 **Behavior:** Same as the method `Time.gm` (see §15.2.19.6.2), except that the method returns
21 a direct instance of the class `Time` which has the current local time zone as its time zone.

1 **15.2.19.6.4 Time.mktime**

2 `Time.mktime(year, month=1, day=0, hour=0, min=0, sec=0, usec=0)`

3 **Visibility:** public

4 **Behavior:** Same as the method `Time.local` (see §15.2.19.6.3).

5 **15.2.19.6.5 Time.now**

6 `Time.now`

7 **Visibility:** public

8 **Behavior:** This method returns a direct instance of the class `Time` which represents the
9 current time with the current time zone.

10 The behavior of this method is the same as the method `new` (see §15.2.3.3.3).

11 **15.2.19.6.6 Time.utc**

12 `Time.utc(year, month=1, day=0, hour=0, min=0, sec=0, usec=0)`

13 **Visibility:** public

14 **Behavior:** Same as the method `Time.gm` (see §15.2.19.6.2).

15 **15.2.19.7 Instance methods**

16 **15.2.19.7.1 Time#+**

17 `+(offset)`

18 **Visibility:** public

19 **Behavior:**

20 a) If the *offset* is not an instance of the class `Integer` or the class `Float`, the behavior is
21 unspecified.

22 b) Let *V* be the value of the *offset*.

23 c) Let *o* be an integer which is the result of computing $V \times 10^6$, rounded off the first
24 decimal place, if any.

25 d) Let *t* and *z* be the microseconds and time zone of the receiver.

- 1 e) Create a direct instance of the class `Time` which represents the time at $(t+o)$ microsec-
2 onds since January 1, 1970 00:00 UTC, with z as its time zone.
- 3 f) Return the resulting instance.

4 **15.2.19.7.2** `Time#-`

5 `-(offset)`

6 **Visibility:** public

7 **Behavior:**

- 8 a) If the *offset* is not an instance of the class `Integer` or the class `Float`, the behavior is
9 unspecified.
- 10 b) Let V be the value of the *offset*.
- 11 c) Let o be an integer which is the result of computing $V \times 10^6$, rounded of the first
12 decimal place, if any.
- 13 d) Let t and z be the microseconds and time zone of the receiver.
- 14 e) Create a direct instance of the class `Time` which represents the time at $t-o$ microseconds
15 since January 1, 1970 00:00 UTC, with z as its time zone.
- 16 f) Return the resulting instance.

17 **15.2.19.7.3** `Time#<=>`

18 `<=>(other)`

19 **Visibility:** public

20 **Behavior:**

- 21 a) If the *other* is not an instance of the class `Time`, return **nil**.
- 22 b) Otherwise, let T_r and T_o be microseconds of the receiver and the *other*, respectively.
- 23 1) If $T_r > T_o$, return an instance of the class `Integer` whose value is 1.
- 24 2) If $T_r = T_o$, return an instance of the class `Integer` whose value is 0.
- 25 3) If $T_r < T_o$, return an instance of the class `Integer` whose value is -1 .

26 **15.2.19.7.4** `Time#asctime`

1
2
3
4
5
6
7
8
9
10
11
12
13
14

`asctime`

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
- b) Let W be the name of the day of the week in the second row on Table 6 which corresponds to $WeekDay(t)$ in the upper row on the same table.
- c) Let Mon be the name of the month in the second row on Table 5 which corresponds to $MonthFromTime(t)$ in the upper row on the same table.
- d) Let $D, H, M, S,$ and Y be as follows:

$$D = DayWithinMonth(t)$$

$$H = HourFromTime(t)$$

$$M = MinuteFromTime(t)$$

$$S = SecondFromTime(t)$$

$$Y = YearFromTime(t)$$

- e) Create a direct instance of the class `String`, the content of which is the following sequence of characters:

$W Mon D H:M:S Y <line-terminator>$

- f) Return the resulting instance.

Table 6 – The names of the days of the week corresponding integer

0	1	2	3	4	5	6
Sun	Mon	Tue	Wed	Thu	Fly	Sat

15.2.19.7.5 Time#ctime

`ctime`

Visibility: public

Behavior: Same as the method `asctime` (see §15.2.19.7.4).

15.2.19.7.6 Time#day

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

day

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
- b) Compute *DayWithinMonth*(t).
- c) Return an instance of the class `Integer` whose value is the result of Step b.

15.2.19.7.7 Time#dst?

dst?

Visibility: public

Behavior: Let T and Z be the microseconds and time zone of the receiver.

- a) If T falls within the daylight saving time of Z , return **true**.
- b) Otherwise, return **false**.

15.2.19.7.8 Time#getgm

getgm

Visibility: public

Behavior: Same as the method `getutc` (see §15.2.19.7.10).

15.2.19.7.9 Time#getlocal

getlocal

Visibility: public

Behavior: The method returns a new instance of the class `Time` which has the same microseconds as the receiver, but has current local time zone as its time zone.

15.2.19.7.10 Time#getutc

1 `getutc`

2 **Visibility:** public

3 **Behavior:** The method returns a new instance of the class `Time` which has the same
4 microseconds as the receiver, but has UTC as its time zone.

5 **15.2.19.7.11 Time#gmt?**

6 `gmt?`

7 **Visibility:** public

8 **Behavior:** Same as the method `utc?` (see §15.2.19.7.26).

9 **15.2.19.7.12 Time#gmt_offset**

10 `gmt_offset`

11 **Visibility:** public

12 **Behavior:** Same as the method `utc_offset` (see §15.2.19.7.27).

13 **15.2.19.7.13 Time#gmtime**

14 `gmtime`

15 **Visibility:** public

16 **Behavior:** Same as the method `utc` (see §15.2.19.7.25).

17 **15.2.19.7.14 Time#gmtoff**

18 `gmtoff`

19 **Visibility:** public

20 **Behavior:** Same as the method `utc_offset` (see §15.2.19.7.27).

21 **15.2.19.7.15 Time#hour**

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

hour

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
- b) Compute $HourFromTime(t)$.
- c) Return an instance of the class `Integer` whose value is the result of Step b.

15.2.19.7.16 Time#localtime

localtime

Visibility: public

Behavior:

- a) Change the time zone of the receiver to the current local time zone.
- b) Return the receiver.

15.2.19.7.17 Time#mday

mday

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
- b) Compute $DayWithinMonth(t)$.
- c) Return an instance of the class `Integer` whose value is the result of Step b.

15.2.19.7.18 Time#min

min

Visibility: public

Behavior:

- 1 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
2 b) Compute *MinuteFromTime*(t).
3 c) Return an instance of the class `Integer` whose value is the result of Step b.

4 **15.2.19.7.19 Time#mon**

5 `mon`

6 **Visibility:** public

7 **Behavior:**

- 8 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
9 b) Compute *MonthFromTime*(t).
10 c) Return an instance of the class `Integer` whose value is the result of Step b.

11 **15.2.19.7.20 Time#month**

12 `month`

13 **Visibility:** public

14 **Behavior:** Same as the method `mon` (see §15.2.19.7.19).

15 **15.2.19.7.21 Time#sec**

16 `sec`

17 **Visibility:** public

18 **Behavior:**

- 19 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
20 b) Compute *SecondFromTime*(t).
21 c) Return an instance of the class `Integer` whose value is the result of Step b.

22 **15.2.19.7.22 Time#to_f**

1 to_f

2 **Visibility:** public

3 **Behavior:** Let t the microseconds of the receiver.

4 a) Compute $t/10^6$.

5 b) Return a direct instance of the class `Float` whose value is the result of Step a.

6 **15.2.19.7.23 Time#to_i**

7 to_i

8 **Visibility:** public

9 **Behavior:** Let t the microseconds of the receiver.

10 a) Compute $\text{floor}(t/10^6)$.

11 b) Return an instance of the class `Integer` whose value is the result of Step a.

12 **15.2.19.7.24 Time#usec**

13 usec

14 **Visibility:** public

15 **Behavior:**

16 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.

17 b) Compute t modulo 10^6 .

18 c) Return an instance of the class `Integer` whose value is the result of Step b.

19 **15.2.19.7.25 Time#utc**

20 utc

21 **Visibility:** public

22 **Behavior:**

23 a) Change the time zone of the receiver to UTC.

24 b) Return the receiver.

1 **15.2.19.7.26 Time#utc?**

2 `utc?`

3 **Visibility:** public

4 **Behavior:** Let Z be the time zone of the receiver.

5 a) If Z is UTC, return **true**.

6 b) Otherwise, return **false**.

7 **15.2.19.7.27 Time#utc_offset**

8 `utc_offset`

9 **Visibility:** public

10 **Behavior:** Let Z be the time zone of the receiver.

11 a) Compute $\text{floor}(\text{ZoneOffset}(Z)/10^6)$.

12 b) Return an instance of the class `Integer` whose value is the result of Step a.

13 **15.2.19.7.28 Time#wday**

14 `wday`

15 **Visibility:** public

16 **Behavior:**

17 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.

18 b) Compute $\text{WeekDay}(t)$.

19 c) Return an instance of the class `Integer` whose value is the result of Step b

20 **15.2.19.7.29 Time#yday**

21 `yday`

22 **Visibility:** public

23 **Behavior:**

- 1 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
- 2 b) Compute *DayWithinYear*(t).
- 3 c) Return an instance of the class `Integer` whose value is the result of Step b.

4 **15.2.19.7.30 Time#year**

5 `year`

6 **Visibility:** public

7 **Behavior:**

- 8 a) Compute the local time from the receiver (see §15.2.19.4). Let t be the result.
- 9 b) Compute *YearFromTime*(t).
- 10 c) Return an instance of the class `Integer` whose value is the result of Step b.

11 **15.2.19.7.31 Time#zone**

12 `zone`

13 **Visibility:** public

14 **Behavior:** Let Z be the time zone of the receiver.

- 15 a) Create a direct instance of the class `String`, the content of which represents Z . The
- 16 exact content of the instance is unspecified.
- 17 b) Return the resulting instance.

18 **15.2.19.7.32 Time#initialize**

19 `initialize`

20 **Visibility:** private

21 **Behavior:**

- 22 a) Set the microseconds of the receiver to microseconds elapsed since January 1, 1970
- 23 00:00 UTC.
- 24 b) Set the time zone of the receiver to the current time zone.
- 25 c) Return an implementation-defined value.

1 **15.2.19.7.33 Time#initialize_copy**

2 initialize_copy(*original*)

3 **Visibility:** private

4 **Behavior:**

- 5 a) If the *original* is not an instance of the class **Time**, raise a direct instance of the class
6 **TypeError**.
- 7 b) Set the microseconds of the receiver to the microseconds of the *original*.
- 8 c) Set the time zone of the receiver to the time zone of the *original*.
- 9 d) Return an implementation-defined value.

10 **15.2.20 IO**

11 **15.2.20.1 General description**

12 An instance of the class **IO** represents a stream, which is a source and/or a sink of data.

13 An instance of the class **IO** has the following properties:

14 **readability flag:** A boolean value which denotes whether the stream can handle input
15 operations.

16 An instance of the class **IO** is said to be **readable** if and only if this flag is true.

17 Reading from a stream which is not readable raises a direct instance of the class **IOError**.

18 **writability flag:** A boolean value which denotes whether the stream can handle output
19 operations.

20 An instance of the class **IO** is said to be **writable** if and only if this flag is true.

21 Writing to a stream which is not writable raises a direct instance of the class **IOError**.

22 **openness flag:** A boolean value which denotes whether the stream is open.

23 An instance of the class **IO** is said to be **open** if and only if this flag is true. An instance
24 of the class **IO** is said to be **closed** if and only if this flag is false.

25 A closed stream is neither readable nor writable. Reading from or writing to a stream which
26 is not open raises an instance of the class **IOError**.

27 **buffering flag:** A boolean value which denotes whether the data to be written to the
28 stream is buffered.

29 When this flag is true, a conforming processor may delay the output to the receiver until
30 the instance methods **flush** or **close** is invoked.

1 A conforming processor may raise an instance of the class `SystemCallError` when the underlying
2 system reported an error during the execution of methods of the class `IO`.

3 In the following description of the methods of the class `IO`, a *byte* means an integer from 0 to
4 255.

5 **15.2.20.2 Direct superclass**

6 The class `Object`

7 **15.2.20.3 Included modules**

8 The following module is included in the class `IO`.

- 9 • `Enumerable`

10 **15.2.20.4 Singleton methods**

11 **15.2.20.4.1 `IO.open`**

12 `IO.open(*args, &block)`

13 **Visibility:** public

14 **Behavior:**

- 15 a) Invoke the method `new` on the receiver with all the elements of the *args* as the argu-
16 ments. Let *I* be the resulting value.
- 17 b) If the *block* is not given, return *I*.
- 18 c) Otherwise, call the *block* with *I* as the argument. Let *V* be the resulting value.
- 19 d) Invoke the method `close` on *I* with no arguments, even when an exception is raised
20 and not handled in Step c.
- 21 e) Return *V*.

22 **15.2.20.5 Instance methods**

23 **15.2.20.5.1 `IO#close`**

24 `close`

25 **Visibility:** public

26 **Behavior:**

- 27 a) If the receiver is closed, raise a direct instance of the class `IOError`.

- 1 b) If the buffering flag of the receiver is true, and the receiver is buffering any output,
- 2 immediately write all the buffered data to the stream which the receiver represents.
- 3 c) Set the openness flag of the receiver to false.
- 4 d) Return **nil**.

5 **15.2.20.5.2 IO#closed?**

6 **closed?**

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the receiver is closed, return **true**.
- 10 b) Otherwise, return **false**.

11 **15.2.20.5.3 IO#each**

12 **each(&block)**

13 **Visibility:** public

14 **Behavior:**

- 15 a) If the *block* is not given, the behavior is unspecified.
- 16 b) If the receiver is not readable, raise a direct instance of the class **IOError**.
- 17 c) If the receiver has reached its end, return the receiver.
- 18 d) Otherwise, read characters from the receiver until 0x0a is read or the receiver reaches
- 19 its end.
- 20 e) Create a direct instance of the class **String** whose content is the sequence of characters
- 21 read in Step d. Call the *block* with this instance as an argument.
- 22 f) Continue processing from Step c.

23 **15.2.20.5.4 IO#each_byte**

24 **each_byte(&block)**

25 **Visibility:** public

- 1 **Behavior:**
- 2 a) If the *block* is not given, the behavior is unspecified.
- 3 b) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 4 c) If the receiver has reached its end, return the receiver.
- 5 d) Otherwise, read a single byte from the receiver. Call the *block* with an argument, an
6 instance of the class `Integer` whose value is the byte.
- 7 Continue processing from Step c.

8 **15.2.20.5.5 IO#each_line**

9 `each_line(&block)`

10 **Visibility:** public

11 **Behavior:** Same as the method `each` (see §15.2.20.5.3).

12 **15.2.20.5.6 IO#eof?**

13 `eof?`

14 **Visibility:** public

15 **Behavior:**

- 16 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 17 b) If the receiver has reached its end, return **true**. Otherwise, return **false**.

18 **15.2.20.5.7 IO#flush**

19 `flush`

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the receiver is not writable, raise a direct instance of the class `IOError`.
- 23 b) If the buffering flag of the receiver is true, and the receiver is buffering any output,
24 immediately write all the buffered data to the stream which the receiver represents.
- 25 c) Return the receiver.

1 **15.2.20.5.8 IO#getc**

2 `getc`

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 6 b) If the receiver has reached its end, return `nil`.
- 7 c) Otherwise, read a character from the receiver. Return an instance of the class `Object`
8 which represents the character.

9 **15.2.20.5.9 IO#gets**

10 `gets`

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 14 b) If the receiver has reached its end, return `nil`.
- 15 c) Otherwise, read characters from the receiver until `0x0a` is read or the receiver reaches
16 its end.
- 17 d) Create a direct instance of the class `String` whose content is the sequence of characters
18 read in Step c and return this instance.

19 **15.2.20.5.10 IO#initialize_copy**

20 `initialize_copy(original)`

21 **Visibility:** private

22 **Behavior:** The behavior of the method is unspecified.

23 **15.2.20.5.11 IO#print**

24 `print(*args)`

1 **Visibility:** public

2 **Behavior:**

3 a) For each element of the *args* in the indexing order:

4 1) Let *V* be the element. If the element is **nil**, a conforming processor may create
5 a direct instance of the class **String** whose content is “nil” and let *V* be the
6 instance.

7 2) Invoke the method **write** on the receiver with *V* as the argument.

8 b) Return **nil**.

9 **15.2.20.5.12 IO#putc**

10 **putc**(*obj*)

11 **Visibility:** public

12 **Behavior:**

13 a) If the *obj* is not an instance of the class **Integer** or an instance of the class **String**,
14 the behavior is unspecified. If the *obj* is an instance of the class **Integer** whose value
15 is smaller than 0 or larger than 255, the behavior is unspecified.

16 b) If the *obj* is an instance of the class **Integer**, create a direct instance of the class
17 **String** *S* whose content is a single character, whose character code is the value of the
18 *obj*.

19 c) If the *obj* is an instance of the class **String**, create a direct instance of the class **String**
20 *S* whose content is the first character of the *obj*.

21 d) Invoke the method **write** on the receiver with *S* as the argument.

22 e) Return the *obj*.

23 **15.2.20.5.13 IO#puts**

24 **puts**(**args*)

25 **Visibility:** public

26 **Behavior:**

27 a) If the length of the *args* is 0, create a direct instance of the class **String** whose content is
28 a single character 0x0a and invoke the method **write** on the receiver with this instance
29 as an argument.

- 1 b) Otherwise, for each element *E* of the *args* in the indexing order:
- 2 1) If *E* is an instance of the class **Array**, for each element *X* of *E* in the indexing
- 3 order:
- 4 i) If *X* is the same object as *E*, i.e. if *E* contains itself, invoke the method
- 5 **write** on the receiver with an instance of the class **String**, whose content is
- 6 implementation-defined.
- 7 ii) Otherwise, invoke the method **write** on the receiver with *X* as the argument.
- 8 2) Otherwise:
- 9 i) If *E* is **nil**, a conforming processor may create a direct instance of the class
- 10 **String** whose content is “nil” and let *E* be the instance.
- 11 ii) If *E* is not an instance of the class **String**, invoke the method **to_s** on the *E*.
- 12 If the resulting value is an instance of the class **String**, let *E* be the resulting
- 13 value. Otherwise, the behavior is unspecified.
- 14 iii) Invoke the method **write** on the receiver with *E* as the argument.
- 15 iv) If the last character of *E* is not 0x0a, create a direct instance of the class
- 16 **String** whose content is a single character 0x0a and invoke the method **write**
- 17 on the receiver with this instance as an argument.
- 18 c) Return **nil**.

19 **15.2.20.5.14 IO#read**

20 `read(length=nil)`

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the receiver is not readable, raise a direct instance of the class **IOError**.
- 24 b) If the receiver has reached its end:
- 25 1) If the *length* is **nil**, create an empty instance of the class **String** and return that
- 26 instance.
- 27 2) If the *length* is not **nil**, return **nil**.
- 28 c) Otherwise:
- 29 1) If the *length* is **nil**, read characters from the receiver until the receiver reaches its
- 30 end.

- 1 2) If the *length* is an instance of the class `Integer`, let *N* be the value of the *length*.
2 Otherwise, the behavior is unspecified.
- 3 3) If *N* is smaller than 0, raise a direct instance of the class `ArgumentError`.
- 4 4) Read bytes from the receiver until *N* bytes are read or the receiver reaches its end.
- 5 d) Create a direct instance of the class `String` whose content is the sequence of characters
6 read in Step c and return this instance.

7 **15.2.20.5.15 IO#readchar**

8 **readchar**

9 **Visibility:** public

10 **Behavior:**

- 11 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 12 b) If the receiver has reached its end, raise a direct instance of the class `EOFError`.
- 13 c) Otherwise, read a character from the receiver. Return an instance of the class `Object`
14 which represents the character.

15 **15.2.20.5.16 IO#readline**

16 **readline**

17 **Visibility:** public

18 **Behavior:**

- 19 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 20 b) If the receiver has reached its end, raise a direct instance of the class `EOFError`.
- 21 c) Otherwise, read characters from the receiver until `0x0a` is read or the receiver reaches
22 its end.
- 23 d) Create a direct instance of the class `String` whose content is the sequence of characters
24 read in Step c and return this instance.

25 **15.2.20.5.17 IO#readlines**

26 **readlines**

27 **Visibility:** public

- 1 **Behavior:**
- 2 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 3 b) Create an empty direct instance of the class `Array A`.
- 4 c) If the receiver has reached to its end, return `A`.
- 5 d) Otherwise, read characters from the receiver until `0x0a` is read or the receiver reaches
6 its end.
- 7 e) Create a direct instance of the class `String` whose content is the sequence of characters
8 read in Step d and append this instance to `A`.
- 9 f) Continue processing from Step c.

10 **15.2.20.5.18 IO#sync**

11 **sync**

12 **Visibility:** public

13 **Behavior:**

- 14 a) If the receiver is closed, raise a direct instance of the class `IOError`.
- 15 b) If the buffering flag of the receiver is true, return **false**. Otherwise, return **true**.

16 **15.2.20.5.19 IO#sync=**

17 **sync=(bool)**

18 **Visibility:** public

19 **Behavior:**

- 20 a) If the receiver is closed, raise a direct instance of the class `IOError`.
- 21 b) If the `bool` is a trueish value, set the buffering flag of the receiver to false. If the `bool` is
22 a falseish value, set the buffering flag of the receiver to true.
- 23 c) Return the `bool`.

24 **15.2.20.5.20 IO#write**

1 `write(str)`

2 **Visibility:** public

3 **Behavior:**

- 4 a) If the *str* is an instance of the class **String**, let *S* be the *str*.
- 5 b) Otherwise, invoke the method `to_s` on the *str*, and let *S* be the resulting value. If *S* is
6 not an instance of the class **String**, the behavior is unspecified.
- 7 c) If *S* is empty, return an instance of the class **Integer** whose value is 0.
- 8 d) If the receiver is not writable, raise a direct instance of the class **IOError**.
- 9 e) Write all the characters in *S* to the stream which the receiver represents, preserving
10 their order.
- 11 f) Return an instance of the class **Integer**, whose value is implementation-defined.

12 **15.2.21 File**

13 **15.2.21.1 General description**

14 Instances of the class **File** represent opened files.

15 A conforming processor may raise an instance of the class **SystemCallError** during the execution
16 of the methods of the class **File** if the underlying system reports an error.

17 A **path** of a file is a sequence of characters which represents the location of the file. The correct
18 syntax of paths is implementation-defined.

19 An instance of the class **File** has the following property:

20 **path:** The path of the file.

21 **15.2.21.2 Direct superclass**

22 The class **IO**

23 **15.2.21.3 Singleton methods**

24 **15.2.21.3.1 File.exist?**

25 `File.exist?(path)`

26 **Visibility:** public

27 **Behavior:**

- 1 a) If the file specified by the *path* exists, return **true**.
2 b) Otherwise, return **false**.

3 **15.2.21.4 Instance methods**

4 **15.2.21.4.1 File#initialize**

5 `initialize(path, mode="r")`

6 **Visibility:** private

7 **Behavior:**

- 8 a) If the *path* is not an instance of the class **String**, the behavior is unspecified.
- 9 b) If the *mode* is not an instance of the class **String** whose content is a single character
10 "r" or "w", the behavior is unspecified.
- 11 c) Open the file specified by the *path* in an implementation-defined way, and associate it
12 with the receiver.
- 13 d) Set the path of the receiver to the content of the *path*.
- 14 e) Set the openness flag and the buffering flag of the receiver to true.
- 15 f) Set the readability flag and the writability flag of the receiver as follows:
- 16 1) If the *mode* is an instance of the class **String** whose content is a single character
17 "r", set the readability flag of the receiver to true and set the writability flag of
18 the receiver to false.
- 19 2) If the *mode* is an instance of the class **String** whose content is a single character
20 "w", set the readability flag of the receiver to false and set the writability flag of
21 the receiver to true.
- 22 g) Return an implementation-defined value.

23 **15.2.21.4.2 File#path**

24 `path`

25 **Visibility:** public

26 **Behavior:** The method creates a direct instance of the class **String** whose content is the
27 path of the receiver, and returns this instance.

1 15.2.22 Exception

2 15.2.22.1 General description

3 Instances of the class `Exception` represent exceptions. The class `Exception` is a superclass of
4 all the other exception classes.

5 Instances of the class `Exception` have the following property.

6 **message:** An object returned by the method `to_s` (see §15.2.22.5.3).

7 When the method `clone` (see §15.3.1.3.8) or the method `dup` (see §15.3.1.3.9) of the class `Kernel`
8 is invoked on an instance of the class `Exception`, the message property shall be copied from the
9 receiver to the resulting value.

10 15.2.22.2 Direct superclass

11 The class `Object`

12 15.2.22.3 Built-in exception classes

13 This document defines several built-in subclasses of the class `Exception`. Figure 1 shows the
14 list of these subclasses and their class hierarchy. A conforming processor shall raise instances
15 of these built-in subclasses in various erroneous conditions as described in this document. The
16 class hierarchy shown in Figure 1 is used to handle an exception (see §14).

17 15.2.22.4 Singleton methods

18 15.2.22.4.1 `Exception.exception`

19 `Exception.exception(*args, &block)`

20 **Visibility:** public

21 **Behavior:** Same as the method `new` (see §15.2.3.3.3).

22 15.2.22.5 Instance methods

23 15.2.22.5.1 `Exception#exception`

24 `exception(*string)`

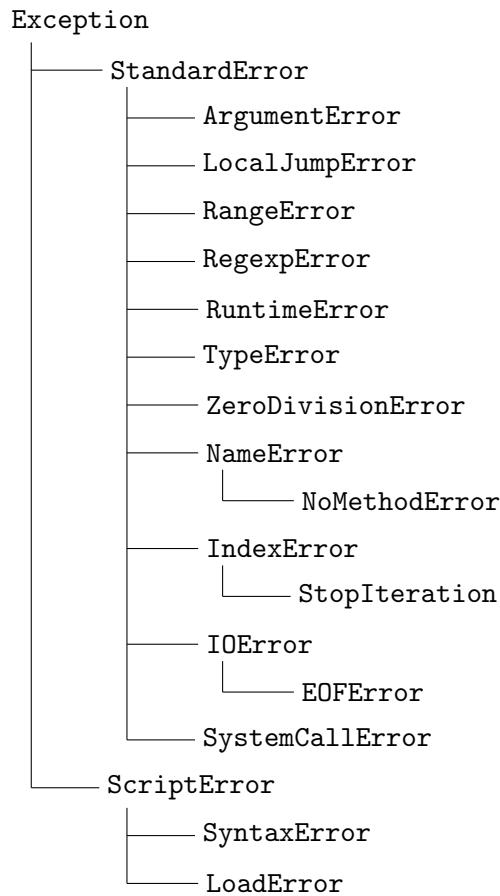
25 **Visibility:** public

26 **Behavior:**

27 a) If the length of the *string* is 0, return the receiver.

28 b) If the length of the *string* is 1:

Figure 1 – The exception class hierarchy



- 1 1) If the only argument is the same object as the receiver, return the receiver.
- 2 2) Otherwise let M be the argument.
 - 3 i) Create a direct instance of the class of the receiver. Let E be the instance.
 - 4 ii) Set the message property of E to M .
 - 5 iii) Return E .
- 6 c) If the length of the *string* is larger than 1, raise a direct instance of the class **ArgumentError**.

7 **15.2.22.5.2 Exception#message**

8 message

9 **Visibility:** public

10 **Behavior:**

- 11 a) Invoke the method `to_s` on the receiver with no arguments.
- 12 b) Return the resulting value of the invocation.

1 15.2.22.5.3 `Exception#to_s`

2 `to_s`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Let M be the message property of the receiver.
- 6 b) If M is **nil**, return an implementation-defined value.
- 7 c) If M is not an instance of the class `String`, the behavior is unspecified.
- 8 d) Otherwise, return M .

9 15.2.22.5.4 `Exception#initialize`

10 `initialize(message=nil)`

11 **Visibility:** private

12 **Behavior:**

- 13 a) Set the message property of the receiver to the *message*.
- 14 b) Return an implementation-defined value.

15 15.2.23 `StandardError`

16 15.2.23.1 General description

17 Instances of the class `StandardError` represent standard errors, which can be handled in a
18 *rescue-clause* without a *exception-class-list* (see §11.5.2.4.1).

19 15.2.23.2 Direct superclass

20 The class `Exception`

21 15.2.24 `ArgumentError`

22 15.2.24.1 General description

23 Instances of the class `ArgumentError` represent argument errors.

24 15.2.24.2 Direct superclass

25 The class `StandardError`

1 15.2.25 LocalJumpError

2 Instances of the class `LocalJumpError` represent errors which occur while evaluating *blocks* and
3 *jump-expressions*.

4 15.2.25.1 Direct superclass

5 The class `StandardError`

6 15.2.25.2 Instance methods

7 15.2.25.2.1 LocalJumpError#exit_value

8 `exit_value`

9 **Visibility:** public

10 **Behavior:** The method returns the value of the instance variable `@exit_value` of the
11 receiver.

12 15.2.25.2.2 LocalJumpError#reason

13 `reason`

14 **Visibility:** public

15 **Behavior:** The method returns the value of the instance variable `@reason` of the receiver.

16 15.2.26 RangeError

17 15.2.26.1 General description

18 Instances of the class `RangeError` represent range errors.

19 15.2.26.2 Direct superclass

20 The class `StandardError`

21 15.2.27 RegexpError

22 15.2.27.1 General description

23 Instances of the class `RegexpError` represent regular expression errors.

24 15.2.27.2 Direct superclass

25 The class `StandardError`

1 **15.2.28 RuntimeError**

2 **15.2.28.1 General description**

3 Instances of the class `RuntimeError` represent runtime errors, which are raised by the method
4 `raise` of the class `Kernel` by default (see §15.3.1.2.13).

5 **15.2.28.2 Direct superclass**

6 The class `StandardError`

7 **15.2.29 TypeError**

8 **15.2.29.1 General description**

9 Instances of the class `TypeError` represent type errors.

10 **15.2.29.2 Direct superclass**

11 The class `StandardError`

12 **15.2.30 ZeroDivisionError**

13 **15.2.30.1 General description**

14 Instances of the class `ZeroDivisionError` represent zero division errors.

15 **15.2.30.2 Direct superclass**

16 The class `StandardError`

17 **15.2.31 NameError**

18 Instances of the class `NameError` represent errors which occur while resolving names to values.

19 Instances of the class `NameError` have the following property.

20 **name:** The name a reference to which causes this exception to be raised.

21 When the method `clone` (see §15.3.1.3.8) or the method `dup` (see §15.3.1.3.9) of the class `Kernel`
22 is invoked on an instance of the class `NameError`, the `name` property shall be copied from the
23 receiver to the resulting value.

24 **15.2.31.1 Direct superclass**

25 The class `StandardError`

26 **15.2.31.2 Instance methods**

27 **15.2.31.2.1 NameError#name**

1 **name**

2 **Visibility:** public

3 **Behavior:** The method returns the name property of the receiver.

4 **15.2.31.2.2 NameError#initialize**

5 **initialize**(*message=nil, name=nil*)

6 **Visibility:** public

7 **Behavior:**

8 a) Set the name property of the receiver to the *name*.

9 b) Invoke the method **initialize** defined in the class **Exception**, which is a superclass
10 of the class **NameError**, as if *super-with-argument* were evaluated with the *message* as
11 the *argument* of the *super-with-argument*.

12 c) Return an implementation-defined value.

13 **15.2.32 NoMethodError**

14 Instances of the class **NoMethodError** represent errors which occur while invoking methods which
15 do not exist or which cannot be invoked.

16 Instances of the class **NoMethodError** have properties called **name** (see §15.2.31) and **argu-**
17 **ments**. The values of these properties are set in the method **initialize** (see §15.2.32.2.2).

18 When the method **clone** (see §15.3.1.3.8) or the method **dup** (see §15.3.1.3.9) of the class **Kernel**
19 is invoked on an instance of the class **NoMethodError**, those properties shall be copied from the
20 receiver to the resulting value.

21 **15.2.32.1 Direct superclass**

22 The class **NameError**

23 **15.2.32.2 Instance methods**

24 **15.2.32.2.1 NoMethodError#args**

25 **args**

26 **Visibility:** public

27 **Behavior:** The method returns the value of the arguments property of the receiver.

1 15.2.32.2 NoMethodError#initialize

```
2 initialize( message=nil, name=nil, args=nil )
```

3 **Visibility:** public

4 **Behavior:**

- 5 a) Set the arguments property of the receiver to the *args*.
- 6 b) Perform all the steps of the method `initialize` described in §15.2.31.2.2.
- 7 c) Return an implementation-defined value.

8 15.2.33 IndexError

9 15.2.33.1 General description

10 Instances of the class `IndexError` represent index errors.

11 15.2.33.2 Direct superclass

12 The class `StandardError`

13 15.2.34 StopIteration

14 15.2.34.1 General description

15 Instances of the class `StopIteration` represent exceptions, which are raised to terminate the
16 method `loop` of the class `Kernel` (see §15.3.1.2.8).

17 15.2.34.2 Direct superclass

18 The class `IndexError`

19 15.2.35 IOError

20 15.2.35.1 General description

21 Instances of the class `IOError` represent input/output errors.

22 15.2.35.2 Direct superclass

23 The class `StandardError`

24 15.2.36 EOFError

25 15.2.36.1 General description

26 Instances of the class `EOFError` represent errors which occur when a stream has reached its end.

1 **15.2.36.2 Direct superclass**

2 The class IOError

3 **15.2.37 SystemCallError**

4 **15.2.37.1 General description**

5 Instances of the class SystemCallError represent errors which occur while invoking the methods
6 of the class IO.

7 **15.2.37.2 Direct superclass**

8 The class StandardError

9 **15.2.38 ScriptError**

10 **15.2.38.1 General description**

11 Instances of the class ScriptError represent programming errors.

12 **15.2.38.2 Direct superclass**

13 The class Exception

14 **15.2.39 SyntaxError**

15 **15.2.39.1 General description**

16 Instances of the class SyntaxError represent syntax errors.

17 **15.2.39.2 Direct superclass**

18 The class ScriptError

19 **15.2.40 LoadError**

20 **15.2.40.1 General description**

21 Instances of the class LoadError represent errors which occur while loading external programs
22 (see §15.3.1.2.14).

23 **15.2.40.2 Direct superclass**

24 The class ScriptError

1 15.3 Built-in modules

2 15.3.1 Kernel

3 15.3.1.1 General description

4 The module `Kernel` is included in the class `Object`. Unless overridden, instance methods defined
5 in the module `Kernel` can be invoked on any instance of the class `Object`.

6 15.3.1.2 Singleton methods

7 15.3.1.2.1 `Kernel.‘`

8 `Kernel.‘(string)`

9 The method `‘` is invoked in the form described in §8.7.6.3.7.

10 **Visibility:** public

11 **Behavior:** The method `‘` executes an external command corresponding to the *string*. The
12 external command executed by the method is implementation-defined.

13 When the method is invoked, take the following steps:

- 14 a) If the *string* is not an instance of the class `String`, the behavior is unspecified.
- 15 b) Execute the command which corresponds to the content of the *string*. Let *R* be the
16 output of the command.
- 17 c) Create a direct instance of the class `String` whose content is *R*, and return the instance.

18 15.3.1.2.2 `Kernel.block_given?`

19 `Kernel.block_given?`

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the top of `[[block]]` is block-not-given, return **false**.
- 23 b) Otherwise, return **true**.

24 15.3.1.2.3 `Kernel.eval`

25 `Kernel.eval(string)`

1 **Visibility:** public

2 **Behavior:**

- 3 a) If the *string* is not an instance of the class **String**, the behavior is unspecified.
- 4 b) Parse the content of the *string* as a *program* (see §10.1). If it fails, raise a direct instance
5 of the class **SyntaxError**.
- 6 c) Evaluate the *program*. Let *V* be the resulting value of the evaluation.
- 7 d) Return *V*.

8 In Step c, the *string* is evaluated under the new local variable scope in which references to
9 *local-variable-identifiers* are resolved in the same way as in scopes created by *blocks* (see
10 §9.2).

11 15.3.1.2.4 **Kernel.global_variables**

12 `Kernel.global_variables`

13 **Visibility:** public

14 **Behavior:** The method returns a new direct instance of the class **Array** which consists of
15 names of all the global variables. These names are represented by instances of either the
16 class **String** or the class **Symbol**. Which of those classes is chosen is implementation-defined.

17 15.3.1.2.5 **Kernel.iterator?**

18 `Kernel.iterator?`

19 **Visibility:** public

20 **Behavior:** Same as the method `Kernel.block_given?` (see §15.3.1.2.2).

21 15.3.1.2.6 **Kernel.lambda**

22 `Kernel.lambda(&block)`

23 **Visibility:** public

24 **Behavior:** The method creates an instance of the class **Proc** as `Proc.new` does (see §15.2.17.3.1).
25 However, the way in which the *block* is evaluated differs from the one described in §11.3.3
26 except when the *block* is called by a *yield-expression*.

27 The differences are as follows.

- 28 a) Before Step d of §11.3.3, the number of arguments is checked as follows:

- 1) Let A be the list of arguments passed to the block. Let N_a be the length of A .
- 2) If the *block-parameter-list* is of the form *left-hand-side*, and if N_a is not 1, the behavior is unspecified.
- 3) If the *block-parameter-list* is of the form *multiple-left-hand-side*:
 - i) If the *multiple-left-hand-side* is not of the form *grouped-left-hand-side* or *packing-left-hand-side*:
 - I) Let N_p be the number of *multiple-left-hand-side-items* of the *multiple-left-hand-side*.
 - II) If $N_a < N_p$, raise a direct instance of the class `ArgumentError`.
 - III) If a *packing-left-hand-side* is omitted, and if $N_a > N_p$, raise a direct instance of the class `ArgumentError`.
 - ii) If the *multiple-left-hand-side* is of the form *grouped-left-hand-side*, and if N_a is not 1, the behavior is unspecified.
- b) In Step e) of §11.3.3, when the evaluation of the block associated with a `lambda` invocation is terminated by a *return-expression* or *break-expression*, the execution context is restored to E_o (i.e. the saved execution context), and the evaluation of the `lambda` invocation is terminated.

The value of the `lambda` invocation is determined as follows:

- 1) If the *jump-argument* of the *return-expression* or the *break-expression* is present, the value of the `lambda` invocation is the value of the *jump-argument*.
- 2) Otherwise, the value of the `lambda` invocation is `nil`.

15.3.1.2.7 Kernel.local_variables

`Kernel.local_variables`

Visibility: public

Behavior: The method returns a new direct instance of the class `Array` which contains all the names of local variable bindings which meet the following conditions.

- The name of the binding is of the form *local-variable-identifier*.
- The binding can be resolved in the scope of local variables which includes the point of invocations of this method by the process described in §9.2.

In the instance of the class `Array` returned by the method, names of the local variables are represented by instances of either the class `String` or the class `Symbol`. Which of those classes is chosen is implementation-defined.

1 15.3.1.2.8 Kernel.loop

2 Kernel.loop(&block)

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the *block* is not given, the behavior is unspecified.
- 6 b) Otherwise, repeat calling the *block*.
- 7 c) If a direct instance of the class `StopIteration` is raised and not handled in Step b,
8 handle the exception and return **nil**.

9 15.3.1.2.9 Kernel.method_missing

10 Kernel.method_missing(*symbol*, *args)

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the *symbol* is not an instance of the class `Symbol`, raise a direct instance of the class
14 `ArgumentError`.
- 15 b) Otherwise, raise a direct instance of the class `NoMethodError` which has the *symbol*
16 as its name property and the *args* as its arguments property. A conforming processor
17 may raise a direct instance of the class `NameError` which has the *symbol* as its name
18 property instead of `NoMethodError` if the method is invoked in Step e of §13.3.3 during
19 an evaluation of a *local-variable-identifier* as a method invocation.

20 15.3.1.2.10 Kernel.p

21 Kernel.p(*args)

22 **Visibility:** public

23 **Behavior:**

- 24 a) For each element *E* of the *args*, in the indexing order, take the following steps:
- 25 1) Invoke the method `inspect` on *E* with no arguments and let *X* be the resulting
26 value of this invocation.
- 27 2) If *X* is not an instance of the class `String`, the behavior is unspecified.

- 1 3) Invoke the method `write` on `Object::STDOUT` with X as the argument.
- 2 4) Invoke the method `write` on `Object::STDOUT` with an argument, which is a new
3 direct instance of the class `String` whose content is a single character `0x0a`.
- 4 b) Return **nil**. A conforming processor may return the *args* instead of **nil**.

5 **15.3.1.2.11 Kernel.print**

6 `Kernel.print(*args)`

7 **Visibility:** public

8 **Behavior:** The method behaves as if the method `print` of the class `IO` (see §15.2.20.5.11)
9 were invoked on `Object::STDOUT` with the same arguments.

10 **15.3.1.2.12 Kernel.puts**

11 `Kernel.puts(*args)`

12 **Visibility:** public

13 **Behavior:** The method behaves as if the method `puts` of the class `IO` (see §15.2.20.5.13)
14 were invoked on `Object::STDOUT` with the same arguments.

15 **15.3.1.2.13 Kernel.raise**

16 `Kernel.raise(*args)`

17 **Visibility:** public

18 **Behavior:**

- 19 a) If the length of the *args* is larger than 2, the behavior is unspecified.
- 20 b) If the length of the *args* is 0:
 - 21 1) If the location of the method invocation is within an *operator-expression₂* of an
22 *assignment-with-rescue-modifier*, a *fallback-statement-of-rescue-modifier-statement*,
23 or a *rescue-clause*, let E be the current exception (see §14.3).
 - 24 2) Otherwise, invoke the method `new` on the class `RuntimeError` with no argument.
25 Let E be the resulting value.
- 26 c) If the length of the *args* is 1, let A be the only argument.

- 1 1) If A is an instance of the class `String`, invoke the method `new` on the class
2 `RuntimeError` with A as the only argument. Let E be the resulting instance.
- 3 2) Otherwise, invoke the method `exception` on A . Let E be the resulting value.
- 4 3) If E is not an instance of the class `Exception`, raise a direct instance of the class
5 `TypeError`.
- 6 d) If the length of the *args* is 2, let F and S be the first and the second argument,
7 respectively.
 - 8 1) Invoke the method `exception` on F with S as the only argument. Let E be the
9 resulting value.
 - 10 2) If E is not an instance of the class `Exception`, raise a direct instance of the class
11 `TypeError`.
- 12 e) Raise E .

13 **15.3.1.2.14 Kernel.require**

14 `Kernel.require(string)`

15 **Visibility:** public

16 **Behavior:** The method `require` evaluates the external program P corresponding to the
17 *string*. The way in which P is determined from the *string* is implementation-defined.

18 When the method is invoked, take the following steps:

- 19 a) If the *string* is not an instance of the class `String`, the behavior is unspecified.
- 20 b) Search for the external program P corresponding to the *string*.
- 21 c) If the program does not exist, raise a direct instance of the class `LoadError`.
- 22 d) If P cannot be derived from the *program* (§10.1), raise a direct instance of the class
23 `SyntaxError`.
- 24 e) Change the state of the execution context temporarily for the evaluation of P as follows:
 - 25 1) `[[self]]` contains only one object which is the object at the bottom of `[[self]]` in the
26 current execution context.
 - 27 2) `[[class-module-list]]` contains only one list whose only element is the class `Object`.
 - 28 3) `[[default-visibility]]` contains only one visibility, which is the private visibility.
 - 29 4) All the other attributes of the execution context are empty logical stacks.

- 1 f) Evaluate P under the execution context set up in Step e.
- 2 g) Restore the state of the execution context as it is just before Step e, even when an
3 exception is raised and not handled during the evaluation of P .
- 4 Note that the evaluation of P affects the restored execution context if it changes the
5 attributes of objects in the original execution context.
- 6 h) Unless an exception is raised and not handled in Step f, return **true**.

7 15.3.1.3 Instance methods

8 15.3.1.3.1 Kernel#==

9 ==(*other*)

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the receiver and the *other* is the same object, return **true**.
- 13 b) Otherwise, return **false**.

14 15.3.1.3.2 Kernel#===

15 ===(*other*)

16 **Visibility:** public

17 **Behavior:**

- 18 a) If the receiver and the *other* is the same object, return **true**.
- 19 b) Otherwise, invoke the method == on the receiver with the *other* as the only argument.
20 Let V be the resulting value.
- 21 c) If V is a trueish value, return **true**. Otherwise, return **false**.

22 15.3.1.3.3 Kernel#_id_

23 _id_

24 **Visibility:** public

25 **Behavior:** Same as the method `object_id` (see §15.3.1.3.33).

1 **15.3.1.3.4 Kernel#__send__**

2 `__send__(symbol, *args, &block)`

3 **Visibility:** public

4 **Behavior:** Same as the method `send` (see §15.3.1.3.44).

5 **15.3.1.3.5 Kernel#‘**

6 `‘ (string)`

7 The method `‘` is invoked in the form described in §8.7.6.3.7.

8 **Visibility:** private

9 **Behavior:** Same as the method `Kernel.‘` (see §15.3.1.2.1).

10 **15.3.1.3.6 Kernel#block_given?**

11 `block_given?`

12 **Visibility:** private

13 **Behavior:** Same as the method `Kernel.block_given?` (see §15.3.1.2.2).

14 **15.3.1.3.7 Kernel#class**

15 `class`

16 **Visibility:** public

17 **Behavior:** The method returns the class of the receiver.

18 **15.3.1.3.8 Kernel#clone**

19 `clone`

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the receiver is an instance of one of the following classes: `NilClass`, `TrueClass`,
23 `FalseClass`, `Integer`, `Float`, or `Symbol`, then raise a direct instance of the class
24 `TypeError`.

- 1 b) Create a direct instance of the class of the receiver which has no bindings of instance
2 variables. Let O be the newly created instance.
- 3 c) For each binding B of the instance variables of the receiver, create a variable binding
4 with the same name and value as B in the set of bindings of instance variables of O .
- 5 d) If the receiver is associated with an eigenclass, let E_o be the eigenclass, and take the
6 following steps:
- 7 1) Create an eigenclass whose direct superclass is the direct superclass of E_o . Let E_n
8 be the eigenclass.
- 9 2) For each binding B_{v1} of the constants of E_o , create a variable binding with the
10 same name and value as B_{v1} in the set of bindings of constants of E_n .
- 11 3) For each binding B_{v2} of the class variables of E_o , create a variable binding with
12 the same name and value as B_{v2} in the set of bindings of class variables of E_n .
- 13 4) For each binding B_m of the instance methods of E_o , create a method binding with
14 the same name and value as B_m in the set of bindings of instance methods of E_n .
- 15 5) Associate O with E_n .
- 16 e) Invoke the method `initialize_copy` on O with the receiver as the argument.
- 17 f) Return O .

18 **15.3.1.3.9 Kernel#dup**

19 **dup**

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the receiver is an instance of one of the following classes: `NilClass`, `TrueClass`,
23 `FalseClass`, `Integer`, `Float`, or `Symbol`, then raise a direct instance of the class
24 `TypeError`.
- 25 b) Create a direct instance of the class of the receiver which has no bindings of instance
26 variables. Let O be the newly created instance.
- 27 c) For each binding B of the instance variables of the receiver, create a variable binding
28 with the same name and value as B in the set of bindings of instance variables of O .
- 29 d) Invoke the method `initialize_copy` on O with the receiver as the argument.
- 30 e) Return O .

31 **15.3.1.3.10 Kernel#eql?**

1 `eql?(other)`

2 **Visibility:** public

3 **Behavior:** Same as the method `==` (see §15.3.1.3.1).

4 **15.3.1.3.11 Kernel#equal?**

5 `equal?(other)`

6 **Visibility:** public

7 **Behavior:** Same as the method `==` (see §15.3.1.3.1).

8 **15.3.1.3.12 Kernel#eval**

9 `eval(string)`

10 **Visibility:** private

11 **Behavior:** Same as the method `Kernel.eval` (see §15.3.1.2.3).

12 **15.3.1.3.13 Kernel#extend**

13 `extend(*module_list)`

14 **Visibility:** public

15 **Behavior:** Let *R* be the receiver of the method.

16 a) If the length of the *module_list* is 0, raise a direct instance of the class `ArgumentError`.

17 b) For each element *A* of the *module_list*, take the following steps:

18 1) If *A* is not an instance of the class `Module`, raise a direct instance of the class
19 `TypeError`.

20 2) If *A* is an instance of the class `Class`, raise a direct instance of the class `TypeError`.

21 3) Invoke the method `extend_object` on *A* with *R* as the only argument.

22 4) Invoke the method `extended` on *A* with *R* as the only argument.

23 c) Return *R*.

1 **15.3.1.3.14 Kernel#global_variables**

2 `global_variables`

3 **Visibility:** private

4 **Behavior:** Same as the method `Kernel.global_variables` (see §15.3.1.2.4).

5 **15.3.1.3.15 Kernel#hash**

6 `hash`

7 **Visibility:** public

8 **Behavior:** The method returns an instance of the class `Integer`. When invoked on the
9 same object, the method shall always return an instance of the class `Integer` whose values
10 is same.

11 When a conforming processor overrides the method `eq1?` (see §15.3.1.3.10), it shall override
12 the method `hash` in the same class or module in which the method `eq1?` is overridden in
13 such a way that, if an invocation of the method `eq1?` on an object with an argument returns
14 a trueish value, invocations of the method `hash` on the object and the argument return the
15 instances of the class `Integer` with the same value.

16 **15.3.1.3.16 Kernel#initialize_copy**

17 `initialize_copy(original)`

18 **Visibility:** private

19 **Behavior:** The method `initialize_copy` is invoked when an object is created by the
20 method `clone` (see §15.3.1.3.8) or the method `dup` (see §15.3.1.3.9).

21 When the method is invoked, take the following steps:

- 22 a) If the classes of the receiver and the *original* are not the same class, raise a direct
23 instance of the class `TypeError`.
- 24 b) Return an implementation-defined value.

25 **15.3.1.3.17 Kernel#inspect**

26 `inspect`

27 **Visibility:** public

1 **Behavior:** The method returns a new direct instance of the class `String`, the content of
2 which represents the state of the receiver. The content of the resulting instance of the class
3 `String` is implementation-defined.

4 15.3.1.3.18 Kernel#instance_eval

5 `instance_eval(string = nil, &block)`

6 **Visibility:** public

7 **Behavior:**

- 8 a) If the receiver is an instance of one of the implementation-defined set of classes as
9 described in Step a of §13.4.2, or if the receiver is one of **nil**, **true**, or **false**, then the
10 behavior is unspecified.
- 11 b) If the receiver is not associated with an eigenclass, create a new eigenclass. Let M be
12 the newly created eigenclass.
- 13 c) If the receiver is associated with an eigenclass, let M be that eigenclass.
- 14 d) Take Step b through the last step of the method `class_eval` of the class `Module` (see
15 §15.2.2.4.15).

16 15.3.1.3.19 Kernel#instance_of?

17 `instance_of?(module)`

18 **Visibility:** public

19 **Behavior:** Let C be the class of the receiver.

- 20 a) If the *module* is not an instance of the class `Class` or the class `Module`, raise a direct
21 instance of the class `TypeError`.
- 22 b) If the *module* and C are the same object, return **true**.
- 23 c) Otherwise, return **false**.

24 15.3.1.3.20 Kernel#instance_variable_defined?

25 `instance_variable_defined?(symbol)`

26 **Visibility:** public

27 **Behavior:**

- 1 a) Let N be the name designated by the *symbol*.
- 2 b) If N is not of the form *instance-variable-identifier*, raise a direct instance of the class
3 `NameError` which has the *symbol* as its name property.
- 4 c) If a binding of an instance variable with name N exists in the set of bindings of instance
5 variables of the receiver, return **true**.
- 6 d) Otherwise, return **false**.

7 **15.3.1.3.21 Kernel#instance_variable_get**

8 `instance_variable_get(symbol)`

9 **Visibility:** public

10 **Behavior:**

- 11 a) Let N be the name designated by the *symbol*.
- 12 b) If N is not of the form *instance-variable-identifier*, raise a direct instance of the class
13 `NameError` which has the *symbol* as its name property.
- 14 c) If a binding of an instance variable with name N exists in the set of bindings of instance
15 variables of the receiver, return the value of the binding.
- 16 d) Otherwise, return **nil**.

17 **15.3.1.3.22 Kernel#instance_variable_set**

18 `instance_variable_set(symbol, obj)`

19 **Visibility:** public

20 **Behavior:**

- 21 a) Let N be the name designated by the *symbol*.
- 22 b) If N is not of the form *instance-variable-identifier*, raise a direct instance of the class
23 `NameError` which has the *symbol* as its name property.
- 24 c) If a binding of an instance variable with name N exists in the set of bindings of instance
25 variables of the receiver, replace the value of the binding with the *obj*.
- 26 d) Otherwise, create a variable binding with name N and value *obj* in the set of bindings
27 of instance variables of the receiver.
- 28 e) Return the *obj*.

1 **15.3.1.3.23 Kernel#instance_variables**

2 `instance_variables`

3 **Visibility:** public

4 **Behavior:** The method returns a new direct instance of the class `Array` which consists
5 of names of all the instance variables of the receiver. These names are represented by
6 instances of either the class `String` or the class `Symbol`. Which of those classes is chosen is
7 implementation-defined.

8 **15.3.1.3.24 Kernel#is_a?**

9 `is_a?(module)`

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the *module* is not an instance of the class `Class` or the class `Module`, raise a direct
13 instance of the class `TypeError`.
- 14 b) Let *C* be the class of the receiver.
- 15 c) If the *module* is an instance of the class `Class` and one of the following conditions
16 holds, return **true**.
- 17 • The *module* and *C* are the same object.
 - 18 • The *module* is a superclass of *C*.
 - 19 • The *module* and the eigenclass of the receiver are the same object.
- 20 d) If the *module* is an instance of the class `Module` and is included in *C* or one of the
21 superclasses of *C*, return **true**.
- 22 e) Otherwise, return **false**.

23 **15.3.1.3.25 Kernel#iterator?**

24 `iterator?`

25 **Visibility:** private

26 **Behavior:** Same as the method `Kernel.iterator?` (see §15.3.1.2.5).

27 **15.3.1.3.26 Kernel#kind_of?**

1 `kind_of?(module)`

2 **Visibility:** public

3 **Behavior:** Same as the method `is_a?` (see §15.3.1.3.24).

4 **15.3.1.3.27 Kernel#lambda**

5 `lambda(&block)`

6 **Visibility:** private

7 **Behavior:** Same as the method `Kernel.lambda` (see §15.3.1.2.6).

8 **15.3.1.3.28 Kernel#local_variables**

9 `local_variables`

10 **Visibility:** private

11 **Behavior:** Same as the method `Kernel.local_variables` (see §15.3.1.2.7).

12 **15.3.1.3.29 Kernel#loop**

13 `loop(&block)`

14 **Visibility:** private

15 **Behavior:** Same as the method `Kernel.loop` (see §15.3.1.2.8).

16 **15.3.1.3.30 Kernel#method_missing**

17 `method_missing(symbol, *args)`

18 **Visibility:** private

19 **Behavior:** Same as the method `Kernel.method_missing` (see §15.3.1.2.9).

20 **15.3.1.3.31 Kernel#methods**

1 `methods(all=true)`

2 **Visibility:** public

3 **Behavior:** Let *C* be the class of the receiver.

4 a) If the *all* is a trueish value, the method behaves as if the method `instance_methods`
5 were invoked on *C* with no arguments (see §15.2.2.4.33).

6 b) If the *all* is a falseish value, the method behaves as if the method `singleton_methods`
7 were invoked on the receiver with **false** as the only argument (see §15.3.1.3.45).

8 **15.3.1.3.32 Kernel#nil?**

9 `nil?`

10 **Visibility:** public

11 **Behavior:**

12 a) If the receiver is **nil**, return **true**.

13 b) Otherwise, return **false**.

14 **15.3.1.3.33 Kernel#object_id**

15 `object_id`

16 **Visibility:** public

17 **Behavior:** The method returns an instance of the class `Integer` with the same value
18 whenever it is invoked on the same object. When invoked on two distinct objects, the
19 method returns an instance of the class `Integer` with different value for each invocation.

20 **15.3.1.3.34 Kernel#p**

21 `p(*args)`

22 **Visibility:** private

23 **Behavior:** Same as the method `Kernel.p` (see §15.3.1.2.10).

24 **15.3.1.3.35 Kernel#print**

1 `print(*args)`

2 **Visibility:** private

3 **Behavior:** Same as the method `Kernel.print` (see §15.3.1.2.11).

4 **15.3.1.3.36 Kernel#private_methods**

5 `private_methods(all=true)`

6 **Visibility:** public

7 **Behavior:**

- 8 a) Create an empty direct instance of the class `Array` *A*.
- 9 b) If the receiver is associated with an eigenclass, let *C* be the eigenclass.
- 10 c) Let *I* be the set of bindings of instance methods of *C*.

11 For each binding *B* of *I*, let *N* and *V* be the name and the value of *B* respectively, and
12 take the following steps:

- 13 1) If *V* is undef, or the visibility of *V* is not private, skip the next two steps.
- 14 2) Let *S* be either a new direct instance of the class `String` whose content is *N* or a
15 direct instance of the class `Symbol` whose name is *N*. Which of these classes of
16 instance is chosen as the value of *S* is implementation-defined.
- 17 3) Unless *A* contains the element of the same name (if *S* is an instance of the class
18 `Symbol`) or the same content (if *S* is an instance of the class `String`) as *S*, append
19 *S* to *A*.

20 d) For each module *M* in included module list of *C*, take Step c, assuming that *C* in that
21 step to be *M*.

22 e) Replace *C* with the class of the receiver, and take Step c.

23 f) If the *all* is a trueish value:

- 24 1) Take Step d.
- 25 2) If *C* does not have a direct superclass, return *A*.
- 26 3) Replace *C* with the direct superclass of current *C*.
- 27 4) Take Step c, and then, repeat from Step f-1.

28 g) Return *A*.

1 **15.3.1.3.37 Kernel#protected_methods**

2 `protected_methods(all=true)`

3 **Visibility:** public

4 **Behavior:** Same as the method `private_methods` (see §15.3.1.3.36), except that the method
5 returns a direct instance of the class `Array` which contains names of protected methods.

6 **15.3.1.3.38 Kernel#public_methods**

7 `public_methods`

8 **Visibility:** public

9 **Behavior:** Same as the method `private_methods` (see §15.3.1.3.36), except that the method
10 returns a direct instance of the class `Array` which contains names of public methods.

11 **15.3.1.3.39 Kernel#puts**

12 `puts(*args)`

13 **Visibility:** private

14 **Behavior:** Same as the method `Kernel.puts` (see §15.3.1.2.12).

15 **15.3.1.3.40 Kernel#raise**

16 `raise(*args)`

17 **Visibility:** private

18 **Behavior:** Same as the method `Kernel.raise` (see §15.3.1.2.13).

19 **15.3.1.3.41 Kernel#remove_instance_variable**

20 `remove_instance_variable(symbol)`

21 **Visibility:** private

22 **Behavior:**

23 a) Let N be the name designated by the *symbol*.

- 1 b) If N is not of the form *instance-variable-identifier*, raise a direct instance of the class
2 `NameError` which has the *symbol* as its name property.
- 3 c) If a binding of an instance variable with name N exists in the set of bindings of instance
4 variables of the receiver, let V be the value of the binding.
- 5 1) Remove the binding from the set of bindings of instance variables of the receiver.
- 6 2) Return V .
- 7 d) Otherwise, raise a direct instance of the class `NameError` which has the *symbol* as its
8 name property.

9 **15.3.1.3.42 Kernel#require**

10 `require(*args)`

11 **Visibility:** private

12 **Behavior:** Same as the method `Kernel.require` (see §15.3.1.2.14).

13 **15.3.1.3.43 Kernel#respond_to?**

14 `respond_to?(symbol, include_private=false)`

15 **Visibility:** public

16 **Behavior:**

- 17 a) Let N be the name designated by the *symbol*.
- 18 b) Search for a binding of an instance method named N starting from the receiver of the
19 method as described in §13.3.4.
- 20 c) If a binding is found, let V be the value of the binding.
- 21 1) If V is undef, return **false**.
- 22 2) If the visibility of V is private:
- 23 i) If the *include_private* is a trueish value, return **true**.
- 24 ii) Otherwise, return **false**.
- 25 3) Otherwise, return **true**.
- 26 d) Otherwise, return **false**.

1 **15.3.1.3.44 Kernel#send**

2 `send(symbol, *args, &block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Let N be the name designated by the *symbol*.
- 6 b) Invoke the method named N on the receiver with the *args* as arguments and the *block*
7 as the block, if any.
- 8 c) Return the resulting value of the invocation.

9 **15.3.1.3.45 Kernel#singleton_methods**

10 `singleton_methods(all=true)`

11 **Visibility:** public

12 **Behavior:** Let E be the eigenclass of the receiver.

- 13 a) Create an empty direct instance of the class **Array** A .
- 14 b) Let I be the set of bindings of instance methods of E .

15 For each binding B of I , let N and V be the name and the value of B respectively, and
16 take the following steps:

- 17 1) If V is undef, or the visibility of V is private, skip the next two steps.
- 18 2) Let S be either a new direct instance of the class **String** whose content is N or a
19 direct instance of the class **Symbol** whose name is N . Which of these classes of
20 instance is chosen as the value of S is implementation-defined.
- 21 3) Unless A contains the element of the same name (if S is an instance of the class
22 **Symbol**) or the same content (if S is an instance of the class **String**), append S
23 to A .
- 24 c) If the *all* is a trueish value, for each module M in included module list of E , take Step
25 b, assuming that E in that step to be M .
- 26 d) Return A .

27 **15.3.1.3.46 Kernel#to_s**

1 to_s

2 **Visibility:** public

3 **Behavior:** The method returns a newly created a direct instance of the class **String**, the
4 content of which is the string representation of the receiver. The content of the resulting
5 instance of the class **String** is implementation-defined.

6 15.3.2 Enumerable

7 15.3.2.1 General description

8 The module **Enumerable** provides methods which iterates over the elements of the object using
9 the method **each**.

10 In the following description of the methods of the module **Enumerable**, an *element* of the
11 receiver means one of the values which is yielded by the method **each**.

12 15.3.2.2 Instance methods

13 15.3.2.2.1 Enumerable#all?

14 all?(&block)

15 **Visibility:** public

16 **Behavior:**

17 a) Invoke the method **each** on the receiver.

18 b) For each element *X* which the method **each** yields:

19 1) If the *block* is given, call the *block* with *X* as the argument.

20 If this call results in a falseish value, return **false**.

21 2) If the *block* is not given, and *X* is a falseish value, return **false**.

22 c) Return **true**.

23 15.3.2.2.2 Enumerable#any?

24 any?(&block)

25 **Visibility:** public

26 **Behavior:**

- 1 a) Invoke the method **each** on the receiver.
- 2 b) For each element X which **each** yields:
 - 3 1) If the *block* is given, call the *block* with X as the argument.
 - 4 If this call results in a trueish value, return **true**.
 - 5 2) If the *block* is not given, and X is a trueish value, return **true**.
- 6 c) Return **false**.

7 **15.3.2.2.3 Enumerable#collect**

8 `collect(&block)`

9 **Visibility:** public

10 **Behavior:**

- 11 a) If the *block* is not given, the behavior is unspecified.
- 12 b) Create an empty direct instance of the class **Array** A .
- 13 c) Invoke the method **each** on the receiver.
- 14 d) For each element X which **each** yields, call the *block* with X as the argument and
15 append the resulting value to A .
- 16 e) Return A .

17 **15.3.2.2.4 Enumerable#detect**

18 `detect(ifnone=nil, &block)`

19 **Visibility:** public

20 **Behavior:**

- 21 a) If the *block* is not given, the behavior is unspecified.
- 22 b) Invoke the method **each** on the receiver.
- 23 c) For each element X which **each** yields, call the *block* with X as the argument. If this
24 call results in a trueish value, return X .
- 25 d) Return the *ifnone*.

1 15.3.2.2.5 Enumerable#each_with_index

2 `each_with_index(&block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the *block* is not given, the behavior is unspecified.
- 6 b) Let *i* be 0.
- 7 c) Invoke the method `each` on the receiver.
- 8 d) For each element *X* which `each` yields:
- 9 1) Call the *block* with *X* and *i* as the arguments.
- 10 2) Increase *i* by 1.
- 11 e) Return the receiver.

12 15.3.2.2.6 Enumerable#entries

13 `entries`

14 **Visibility:** public

15 **Behavior:**

- 16 a) Create an empty direct instance of the class `Array` *A*.
- 17 b) Invoke the method `each` on the receiver.
- 18 c) For each element *X* which `each` yields, append *X* to *A*.
- 19 d) Return *A*.

20 15.3.2.2.7 Enumerable#find

21 `find(ifnone=nil, &block)`

22 **Visibility:** public

23 **Behavior:** Same as the method `detect` (see §15.3.2.2.4).

24 15.3.2.2.8 Enumerable#find_all

1 `find_all(&block)`

2 **Visibility:** public

3 **Behavior:**

- 4 a) If the *block* is not given, the behavior is unspecified.
- 5 b) Create an empty direct instance of the class `Array` *A*.
- 6 c) Invoke the method `each` on the receiver.
- 7 d) For each element *X* which `each` yields, call the *block* with *X* as the argument. If this
- 8 call results in a trueish value, append *X* to *A*.
- 9 e) Return *A*.

10 **15.3.2.2.9 Enumerable#grep**

11 `grep(pattern, &block)`

12 **Visibility:** public

13 **Behavior:**

- 14 a) Create an empty direct instance of the class `Array` *A*.
- 15 b) Invoke the method `each` on the receiver.
- 16 c) For each element *X* which `each` yields, invoke the method `===` on the *pattern* with *X*
- 17 as the argument.
- 18 If this invocation results in a trueish value:
- 19 1) If the *block* is given, call the *block* with *X* as the argument and append the resulting
- 20 value to *A*.
- 21 2) Otherwise, append *X* to *A*.
- 22 d) Return *A*.

23 **15.3.2.2.10 Enumerable#include?**

24 `include?(obj)`

25 **Visibility:** public

1 **Behavior:**

- 2 a) Invoke the method `each` on the receiver.
- 3 b) For each element X which `each` yields, invoke the method `==` on X with the *obj* as the
4 argument. If this invocation results in a trueish value, return **true**.
- 5 c) Return **false**.

6 **15.3.2.2.11 Enumerable#inject**

7 `inject(*args, &block)`

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the *block* is not given, the behavior is unspecified.
- 11 b) If the length of the *args* is 2, the behavior is unspecified. If the length of the *args* is
12 smaller than 0 or larger than 2, raise a direct instance of the class `ArgumentError`.
- 13 c) Invoke the method `each` on the receiver. If the method `each` does not yield any element,
14 return **nil**.
- 15 d) For each element X which `each` yields:
- 16 1) If X is the first element, and the length of the *args* is 0, let V be X .
- 17 2) If X is the first element, and the length of the *args* is 1, call the *block* with two
18 arguments, which are the only element of the *args* and X . Let V be the resulting
19 value of this call.
- 20 3) If X is not the first element, call the *block* with V and X as the arguments. Let
21 new V be the resulting value of this call.
- 22 e) Return V .

23 **15.3.2.2.12 Enumerable#map**

24 `map(&block)`

25 **Visibility:** public

26 **Behavior:** Same as the method `collect` (see §15.3.2.2.3).

27 **15.3.2.2.13 Enumerable#max**

1 `max(&block)`

2 **Visibility:** public

3 **Behavior:**

4 a) Invoke the method `each` on the receiver.

5 b) If the method `each` does not yield any elements, return `nil`.

6 c) For each element X which the method `each` yields:

7 1) If X is the first element, let V be X .

8 2) Otherwise, if the *block* is given:

9 i) Call the *block* with X and V as the arguments. Let D be the result of this
10 call.

11 ii) If D is not an instance of the class `Integer`, the behavior is unspecified.

12 iii) If the value of D is larger than 0, let new V be X .

13 If the *block* is not given:

14 i) Invoke the method `<=>` on X with V as the argument. Let D be the result
15 of this invocation.

16 ii) If D is not an instance of the class `Integer`, the behavior is unspecified.

17 iii) If the value of D is larger than 0, let new V be X .

18 d) Return V .

19 **15.3.2.2.14 Enumerable#min**

20 `min(&block)`

21 **Visibility:** public

22 **Behavior:**

23 a) Invoke the method `each` on the receiver.

24 b) If the method `each` does not yield any elements, return `nil`.

25 c) For each element X which the method `each` yields:

26 1) If X is the first element, let V be X .

- 1 2) Otherwise, if the *block* is given:
- 2 i) Call the *block* with *X* and *V* as the arguments. Let *D* be the result of this
- 3 call.
- 4 ii) If *D* is not an instance of the class `Integer`, the behavior is unspecified.
- 5 iii) If the value of *D* is smaller than 0, let new *V* be *X*.
- 6 If the *block* is not given:
- 7 i) Invoke the method `<=>` on *X* with *V* as the argument. Let *D* be the result
- 8 of this invocation.
- 9 ii) If *D* is not an instance of the class `Integer`, the behavior is unspecified.
- 10 iii) If the value of *D* is smaller than 0, let new *V* be *X*.
- 11 d) Return *V*.

12 **15.3.2.2.15 Enumerable#member?**

13 `member?(obj)`

14 **Visibility:** public

15 **Behavior:** Same as the method `include?` (see §15.3.2.2.10).

16 **15.3.2.2.16 Enumerable#partition**

17 `partition(&block)`

18 **Visibility:** public

19 **Behavior:**

- 20 a) If the *block* is not given, the behavior is unspecified.
- 21 b) Create two empty direct instances of the class `Array` *T* and *F*.
- 22 c) Invoke the method `each` on the receiver.
- 23 d) For each element *X* which `each` yields, call the *block* with *X* as the argument.
- 24 If this call results in a trueish value, append *X* to *T*. If this call results in a falseish
- 25 value, append *X* to *F*.
- 26 e) Return a newly created an instance of the class `Array`, which contains only *T* and *F*
- 27 in this order.

1 **15.3.2.2.17 Enumerable#reject**

2 `reject(&block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the *block* is not given, the behavior is unspecified.
- 6 b) Create an empty direct instance of the class `Array` *A*.
- 7 c) Invoke the method `each` on the receiver.
- 8 d) For each element *X* which `each` yields, call the *block* with *X* as the argument. If this
9 call results in a falseish value, append *X* to *A*.
- 10 e) Return *A*.

11 **15.3.2.2.18 Enumerable#select**

12 `select(&block)`

13 **Visibility:** public

14 **Behavior:** Same as the method `find_all` (see §15.3.2.2.8).

15 **15.3.2.2.19 Enumerable#sort**

16 `sort(&block)`

17 **Visibility:** public

18 **Behavior:**

- 19 a) Create an empty direct instance of the class `Array` *A*.
- 20 b) Invoke the method `each` on the receiver.
- 21 c) Insert all the elements which the method `each` yields into *A*. For any two elements E_i
22 and E_j of *A*, all of the following conditions shall hold:
 - 23 1) Let *i* and *j* be the index of E_i and E_j , respectively.
 - 24 2) If the *block* is given:
 - 25 i) Suppose the *block* is called with E_i and E_j as the arguments.

- 1 ii) If this invocation does not result in an instance of the class `Integer`, the
2 behavior is unspecified.
- 3 iii) If this invocation results in an instance of the class `Integer` whose value is
4 larger than 0, j shall be larger than i .
- 5 iv) If this invocation results in an instance of the class `Integer` whose value is
6 smaller than 0, i shall be larger than j .
- 7 3) If the *block* is not given:
- 8 i) Suppose the method `<=>` is invoked on E_i with E_j as the argument.
- 9 ii) If this invocation does not result in an instance of the class `Integer`, the
10 behavior is unspecified.
- 11 iii) If this invocation results in an instance of the class `Integer` whose value is
12 larger than 0, j shall be larger than i .
- 13 iv) If this invocation results in an instance of the class `Integer` whose value is
14 smaller than 0, i shall be larger than j .
- 15 d) Return A .

16 **15.3.2.2.20** `Enumerable#to_a`

17 `to_a`

18 **Visibility:** public

19 **Behavior:** Same as the method `entries` (see §15.3.2.2.6).

20 **15.3.3** `Comparable`

21 **15.3.3.1** `General description`

22 The module `Comparable` provides methods which compare the receiver and an argument using
23 the method `<=>`.

24 **15.3.3.2** `Instance methods`

25 **15.3.3.2.1** `Comparable#<`

26 `<(other)`

27 **Visibility:** public

28 **Behavior:**

- 1 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
2 resulting value of this invocation.
- 3 b) If *I* is not an instance of the class `Integer`, the behavior is unspecified.
- 4 c) If the value of *I* is smaller than 0, return **true**. Otherwise, return **false**.

5 **15.3.3.2.2 Comparable#<=**

6 `<=(other)`

7 **Visibility:** public

8 **Behavior:**

- 9 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
10 resulting value of this invocation.
- 11 b) If *I* is not an instance of the class `Integer`, the behavior is unspecified.
- 12 c) If the value of *I* is smaller than or equal to 0, return **true**. Otherwise, return **false**.

13 **15.3.3.2.3 Comparable#==**

14 `==(other)`

15 **Visibility:** public

16 **Behavior:**

- 17 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
18 resulting value of this invocation.
- 19 b) If *I* is not an instance of the class `Integer`, the behavior is unspecified.
- 20 c) If the value of *I* is 0, return **true**. Otherwise, return **false**.

21 **15.3.3.2.4 Comparable#>**

22 `>(other)`

23 **Visibility:** public

24 **Behavior:**

- 25 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
26 resulting value of this invocation.

- 1 b) If I is not an instance of the class `Integer`, the behavior is unspecified.
- 2 c) If the value of I is larger than 0, return **true**. Otherwise, return **false**.

3 15.3.3.2.5 `Comparable#>=`

4 `>=(other)`

5 **Visibility:** public

6 **Behavior:**

- 7 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let I be the
8 resulting value of this invocation.
- 9 b) If I is not an instance of the class `Integer`, the behavior is unspecified.
- 10 c) If the value of I is larger than or equal to 0, return **true**. Otherwise, return **false**.

11 15.3.3.2.6 `Comparable#between?`

12 `between?(left, right)`

13 **Visibility:** public

14 **Behavior:**

- 15 a) Invoke the method `<=>` on the receiver with the *left* as the argument. Let I_1 be the
16 resulting value of this invocation.
- 17 1) If I_1 is not an instance of the class `Integer`, the behavior is unspecified.
- 18 2) If the value of I_1 is smaller than 0, return **false**.
- 19 b) Invoke the method `<=>` on the receiver with the *right* as the argument. Let I_2 be the
20 resulting value of this invocation.
- 21 1) If I_2 is not an instance of the class `Integer`, the behavior is unspecified.
- 22 2) If the value of I_2 is larger than 0, return **false**. Otherwise, return **true**.

1
2
3
4 **Annex A**
(informative)
5 **Grammar Summary**

6
7
8
9 **A.1 Lexical structure**

10 **A.1.1 Source text**

11 see §8.2

12 *source-character* ::
13 [any character in ISO/IEC 646:1991 IRV]

14 **A.1.2 Line terminators**

15 see §8.3

16 *line-terminator* ::
17 0x0d? 0x0a

18 **A.1.3 Whitespace**

19 see §8.4

20 *whitespace* ::
21 0x09 | 0x0b | 0x0c | 0x0d | 0x20 | *line-terminator-escape-sequence*

22 *line-terminator-escape-sequence* ::
23 \ 0x0d? 0x0a

24 **A.1.4 Comments**

25 see §8.5

26 *comment* ::
27 *single-line-comment*
28 | *multi-line-comment*

29 *single-line-comment* ::
30 # *comment-content*?

```

1  comment-content ::
2      line-content

3  line-content ::
4      source-character +

5  multi-line-comment ::
6      multi-line-comment-begin-line multi-line-comment-line?
7      multi-line-comment-end-line

8  multi-line-comment-begin-line ::
9      [ beginning of a line ] =begin rest-of-begin-end-line? line-terminator

10 multi-line-comment-end-line ::
11 [ beginning of a line ] =end rest-of-begin-end-line?
12 ( line-terminator | [ end of a program ] )

13 rest-of-begin-end-line ::
14 whitespace + comment-content

15 line ::
16 comment-content line-terminator

17 multi-line-comment-line ::
18 line but not multi-line-comment-end-line

```

19 **A.1.5 End of program markers**

20 see §8.6

```

21 end-of-program-marker ::
22 [ beginning of a line ] __END__ ( line-terminator | [ end of a program ] )

```

23 **A.1.5.1 General description**

24 see §8.7.1

```

25 token ::
26     keyword
27     | identifier
28     | punctuator
29     | operator
30     | literal

```

1 A.1.5.2 Keywords

2 see §8.7.2

3 *keyword* ::

4 __LINE__ | __ENCODING__ | __FILE__ | BEGIN | END | alias | and | begin
5 | break | case | class | def | defined? | do | else | elsif | end
6 | ensure | for | false | if | in | module | next | nil | not | or | redo
7 | rescue | retry | return | self | super | then | true | undef | unless
8 | until | when | while | yield

9 A.1.5.3 Identifiers

10 see §8.7.3

11 *identifier* ::

12 *local-variable-identifier*
13 | *global-variable-identifier*
14 | *class-variable-identifier*
15 | *instance-variable-identifier*
16 | *constant-identifier*
17 | *method-identifier*

18 *local-variable-identifier* ::

19 (*lowercase-character* | *_*) *identifier-character**

20 *global-variable-identifier* ::

21 \$ *identifier-start-character* *identifier-character**

22 *class-variable-identifier* ::

23 @@ *identifier-start-character* *identifier-character**

24 *instance-variable-identifier* ::

25 @ *identifier-start-character* *identifier-character**

26 *constant-identifier* ::

27 *uppercase-character* *identifier-character**

28 *method-identifier* ::

29 *method-only-identifier*
30 | *assignment-like-method-identifier*
31 | *constant-identifier*
32 | *local-variable-identifier*

```

1  method-only-identifier ::
2      ( constant-identifier | local-variable-identifier ) ( ! | ? )

3  assignment-like-method-identifier ::
4      ( constant-identifier | local-variable-identifier ) =

5  identifier-character ::
6      lowercase-character
7      | uppercase-character
8      | decimal-digit
9      | -

10 identifier-start-character ::
11     lowercase-character
12     | uppercase-character
13     | -

14 uppercase-character ::
15     A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R
16     | S | T | U | V | W | X | Y | Z

17 lowercase-character ::
18     a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r
19     | s | t | u | v | w | x | y | z

20 decimal-digit ::
21     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

22 **A.1.5.4 Punctuators**

23 see §8.7.4

```

24 punctuator ::
25     [ | ] | ( | ) | { | } | :: | , | ; | .. | ... | ? | : | =>

```

26 **A.1.5.5 Operators**

27 see §8.7.5

```

28 operator ::
29     ! | != | !~ | && | ||
30     | operator-method-name
31     | =
32     | assignment-operator

```

1 *operator-method-name* ::
2 \wedge | $\&$ | $|$ | $\langle = \rangle$ | $==$ | $===$ | $=\sim$ | $>$ | $>=$ | $<$ | \leq | \ll | \gg | $+$ | $-$
3 $*$ | $/$ | $\%$ | $**$ | \sim | $+@$ | $-@$ | $[]$ | $[]=$ | $'$

4 *assignment-operator* ::
5 *assignment-operator-name* =

6 *assignment-operator-name* ::
7 $+$ | $-$ | $*$ | $**$ | $/$ | \wedge | $\%$ | \ll | \gg | $\&$ | $\&\&$ | $||$ | $|$

8 **A.1.5.5.1** General description

9 see §8.7.6.1

10 *literal* ::
11 *numeric-literal*
12 | *string-literal*
13 | *array-literal*
14 | *regular-expression-literal*
15 | *symbol*

16 **A.1.5.5.2** Numeric literals

17 see §8.7.6.2

18 *numeric-literal* ::
19 *signed-number*
20 | *unsigned-number*

21 *unsigned-number* ::
22 *integer-literal*
23 | *float-literal*

24 *integer-literal* ::
25 *decimal-integer-literal*
26 | *binary-integer-literal*
27 | *octal-integer-literal*
28 | *hexadecimal-integer-literal*

29 *decimal-integer-literal* ::
30 *unprefixed-decimal-integer-literal*
31 | *prefixed-decimal-integer-literal*

```

1  unprefixed-decimal-integer-literal ::
2      0
3      | decimal-digit-without-zero ( _? decimal-digit )*

4  prefixed-decimal-integer-literal ::
5      0 ( d | D ) digit-decimal-part

6  digit-decimal-part ::
7      decimal-digit ( _? decimal-digit )*

8  binary-integer-literal ::
9      0 ( b | B ) binary-digit ( _? binary-digit )*

10 octal-integer-literal ::
11  0 ( _ | o | O )? octal-digit ( _? octal-digit )*

12 hexadecimal-integer-literal ::
13  0 ( x | X ) hexadecimal-digit ( _? hexadecimal-digit )*

14 float-literal ::
15     float-literal-without-exponent
16     | float-literal-with-exponent

17 float-literal-without-exponent ::
18     unprefixed-decimal-integer-literal . digit-decimal-part

19 float-literal-with-exponent ::
20     significand-part exponent-part

21 significand-part ::
22     float-literal-without-exponent
23     | unprefixed-decimal-integer-literal

24 exponent-part ::
25     ( e | E ) ( + | - )? digit-decimal-part

26 signed-number ::
27     ( + | - ) unsigned-number

28 decimal-digit-without-zero ::
29     1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

30 octal-digit ::
31     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

```

1 *binary-digit* ::
2 0 | 1

3 *hexadecimal-digit* ::
4 *decimal-digit* | a | b | c | d | e | f | A | B | C | D | E | F

5 **A.1.5.5.2.1** General description

6 see §8.7.6.3.1

7 *string-literal* ::
8 *single-quoted-string*
9 | *double-quoted-string*
10 | *quoted-non-expanded-literal-string*
11 | *quoted-expanded-literal-string*
12 | *here-document*
13 | *external-command-execution*

14 **A.1.5.5.2.2** Single quoted strings

15 see §8.7.6.3.2

16 *single-quoted-string* ::
17 ' *single-quoted-string-character** '

18 *single-quoted-string-character* ::
19 *single-quoted-string-non-escaped-character*
20 | *single-quoted-escape-sequence*

21 *single-quoted-escape-sequence* ::
22 *single-escape-character-sequence*
23 | *single-quoted-string-non-escaped-character-sequence*

24 *single-escape-character-sequence* ::
25 \ *single-quoted-string-meta-character*

26 *single-quoted-string-non-escaped-character-sequence* ::
27 \ *single-quoted-string-non-escaped-character*

28 *single-quoted-string-meta-character* ::
29 ' | \

30 *single-quoted-string-non-escaped-character* ::
31 *source-character* **but not** *single-escaped-character*

1 **A.1.5.5.2.3 Double quoted strings**

2 see §8.7.6.3.3

3 *double-quoted-string* ::
4 " *double-quoted-string-character** "

5 *double-quoted-string-character* ::
6 *source-character* **but not** (" | # | \)
7 | # [lookahead \notin { "\$", "@", "{", " }"]
8 | *double-escape-sequence*
9 | *interpolated-character-sequence*

10 *double-escape-sequence* ::
11 *simple-escape-sequence*
12 | *non-escaped-sequence*
13 | *line-terminator-escape-sequence*
14 | *octal-escape-sequence*
15 | *hex-escape-sequence*
16 | *control-escape-sequence*

17 *simple-escape-sequence* ::
18 \ *double-escaped-character*

19 *non-escaped-sequence* ::
20 \ *non-escaped-double-quoted-string-character*

21 *line-terminator-escape-sequence* ::
22 \ *line-terminator*

23 *non-escaped-double-quoted-string-character* ::
24 *source-character* **but not** (*alpha-numeric-character* | *line-terminator*)

25 *double-escaped-character* ::
26 \ | n | t | r | f | v | a | e | b | s

27 *octal-escape-sequence* ::
28 \ *octal-digit* (*octal-digit* *octal-digit*?)?

29 *hex-escape-sequence* ::
30 \ x *hexadecimal-digit* *hexadecimal-digit*?

31 *control-escape-sequence* ::
32 \ (C - | c) *control-escaped-character*

1 *control-escaped-character* ::
2 *double-escape-sequence*
3 | ?
4 | *source-character* **but not** (\ | ?)

5 *interpolated-character-sequence* ::
6 # *global-variable-identifier*
7 | # *class-variable-identifier*
8 | # *instance-variable-identifier*
9 | # { *compound-statement* }

10 *alpha-numeric-character* ::
11 *uppercase-character*
12 | *lowercase-character*
13 | *decimal-digit*

14 A.1.5.5.2.4 Quoted non-expanded literal strings

15 see §8.7.6.3.4

16 *quoted-non-expanded-literal-string* ::
17 %q *literal-beginning-delimiter non-expanded-literal-string* literal-ending-delimiter*

18 *non-expanded-literal-string* ::
19 *non-expanded-literal-character*
20 | *non-expanded-delimited-string*

21 *non-expanded-delimited-string* ::
22 *literal-beginning-delimiter non-expanded-literal-string* literal-ending-delimiter*

23 *non-expanded-literal-character* ::
24 *non-escaped-literal-character*
25 | *non-expanded-literal-escape-sequence*

26 *non-escaped-literal-character* ::
27 *source-character* **but not** *quoted-literal-escape-character*

28 *non-expanded-literal-escape-sequence* ::
29 *non-expanded-literal-escape-character-sequence*
30 | *non-escaped-non-expanded-literal-character-sequence*

31 *non-expanded-literal-escape-character-sequence* ::
32 \ *non-expanded-literal-escaped-character*

1 *non-expanded-literal-escaped-character* ::
2 *literal-beginning-delimiter*
3 | *literal-ending-delimiter*
4 | \

5 *quoted-literal-escape-character* ::
6 *non-expanded-literal-escaped-character*

7 *non-escaped-non-expanded-literal-character-sequence* ::
8 \ *non-escaped-non-expanded-literal-character*

9 *non-escaped-non-expanded-literal-character* ::
10 *source-character* **but not** *non-expanded-literal-escaped-character*

11 A.1.5.5.2.5 Quoted expanded literal strings

12 see §8.7.6.3.5

13 *quoted-expanded-literal-string* ::
14 % Q? *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

15 *expanded-literal-string* ::
16 *expanded-literal-character*
17 | *expanded-delimited-string*

18 *expanded-literal-character* ::
19 *non-escaped-literal-character* **but not** #
20 | # [lookahead ∉ { "\$", "@", "{ " }]
21 | *double-escape-sequence*
22 | *interpolated-character-sequence*

23 *expanded-delimited-string* ::
24 *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

25 *literal-beginning-delimiter* ::
26 *source-character* **but not** *alpha-numeric-character*

27 *literal-ending-delimiter* ::
28 [depending on the *literal-beginning-delimiter*]

29 *matching-literal-beginning-delimiter* ::
30 (| { | < | [

1 A.1.5.5.2.6 Here documents

2 see §8.7.6.3.6

3 *here-document* ::
4 *heredoc-start-line* *heredoc-body* *heredoc-end-line*

5 *heredoc-start-line* ::
6 *heredoc-signifier* *rest-of-line*

7 *heredoc-signifier* ::
8 << *heredoc-delimiter-specifier*

9 *rest-of-line* ::
10 *line-content?* *line-terminator*

11 *heredoc-body* ::
12 *heredoc-body-line**

13 *heredoc-body-line* ::
14 *line* **but not** *heredoc-end-line*

15 *heredoc-delimiter-specifier* ::
16 -? *heredoc-delimiter*

17 *heredoc-delimiter* ::
18 *non-quoted-delimiter*
19 | *single-quoted-delimiter*
20 | *double-quoted-delimiter*
21 | *command-quoted-delimiter*

22 *non-quoted-delimiter* ::
23 *non-quoted-delimiter-identifier*

24 *non-quoted-delimiter-identifier* ::
25 *identifier-character**

26 *single-quoted-delimiter* ::
27 ' *single-quoted-delimiter-identifier* '

28 *single-quoted-delimiter-identifier* ::
29 (*source-character* **but not** ')*

30 *double-quoted-delimiter* ::
31 " *double-quoted-delimiter-identifier* "

```

1  double-quoted-delimiter-identifier ::
2      ( source-character but not " )*

3  command-quoted-delimiter ::
4      ‘ command-quoted-delimiter-identifier ‘

5  command-quoted-delimiter-identifier ::
6      ( source-character but not ‘ )*

7  heredoc-end-line ::
8      indented-heredoc-end-line
9      | non-indented-heredoc-end-line

10 indented-heredoc-end-line ::
11     [ beginning of a line ] whitespace* heredoc-delimiter-identifier line-terminator

12 non-indented-heredoc-end-line ::
13     [ beginning of a line ] heredoc-delimiter-identifier line-terminator

14 heredoc-delimiter-identifier ::
15     non-quoted-delimiter-identifier
16     | single-quoted-delimiter-identifier
17     | double-quoted-delimiter-identifier
18     | command-quoted-delimiter-identifier

```

19 A.1.5.5.2.7 External command execution

20 see §8.7.6.3.7

```

21 external-command-execution ::
22     backquoted-external-command-execution
23     | quoted-external-command-execution

24 backquoted-external-command-execution ::
25     ‘ backquoted-external-command-execution-character* ‘

26 backquoted-external-command-execution-character ::
27     source-character but not ( ‘ | # | \ )
28     | # [lookahead ∉ { "$", "@", "{", " }" } ]
29     | double-escape-sequence
30     | interpolated-character-sequence

31 quoted-external-command-execution ::
32     %x literal-beginning-delimiter expanded-literal-string* literal-ending-delimiter

```

1 **A.1.5.5.3 Array literals**

2 see §8.7.6.4

3 *array-literal* ::
4 *quoted-non-expanded-array-constructor*
5 | *quoted-expanded-array-constructor*

6 *quoted-non-expanded-array-constructor* ::
7 %w *literal-beginning-delimiter non-expanded-array-content literal-ending-delimiter*

8 *non-expanded-array-content* ::
9 *quoted-array-item-separator-list?* *non-expanded-array-item-list?*
10 *quoted-array-item-separator-list?*

11 *non-expanded-array-item-list* ::
12 *non-expanded-array-item* (*quoted-array-item-separator-list non-expanded-array-item*)*

13 *quoted-array-item-separator-list* ::
14 *quoted-array-item-separator* +

15 *quoted-array-item-separator* ::
16 *whitespace*
17 | *line-terminator*

18 *non-expanded-array-item* ::
19 *non-expanded-array-item-character* +

20 *non-expanded-array-item-character* ::
21 *non-escaped-array-item-character*
22 | *non-expanded-array-escape-sequence*

23 *non-escaped-array-item-character* ::
24 *non-escaped-array-character*
25 | *matching-literal-delimiter*

26 *non-escaped-array-character* ::
27 *non-escaped-literal-character* **but not** *quoted-array-item-separator*

28 *matching-literal-delimiter* ::
29 (| { | < | [|) | } | > |]

```

1  non-expanded-array-escape-sequence ::
2      non-expanded-literal-escape-sequence but not escaped-quoted-array-item-separator
3      | escaped-quoted-array-item-separator

4  escaped-quoted-array-item-separator ::
5      \ quoted-array-item-separator

6  quoted-expanded-array-constructor ::
7      %W literal-beginning-delimiter expanded-array-content literal-ending-delimiter

8  expanded-array-content ::
9      quoted-array-item-separator-list? expanded-array-item-list?
10     quoted-array-item-separator-list?

11 expanded-array-item-list ::
12     expanded-array-item ( quoted-array-item-separator-list expanded-array-item )*

13 expanded-array-item ::
14     expanded-array-item-character +

15 expanded-array-item-character ::
16     non-escaped-array-item-character but not #
17     | # [lookahead ∉ { "$", "@", "{" " }" ]
18     | expanded-array-escape-sequence
19     | interpolated-character-sequence

20 expanded-array-escape-sequence ::
21     double-escape-sequence but not escaped-quoted-array-item-separator
22     | escaped-quoted-array-item-separator

```

23 A.1.5.5.4 Regular expression literals

24 see §8.7.6.5

```

25 regular-expression-literal ::
26     / regular-expression-body / regular-expression-option*
27     | %r literal-beginning-delimiter expanded-literal-string*
28     literal-ending-delimiter regular-expression-option*

29 regular-expression-body ::
30     regular-expression-character*

31 regular-expression-character ::
32     source-character but not ( / | # | \ )
33     | \ \

```

1 | # [lookahead \notin { "\$", "@", "{", " " }]
2 | *line-terminator-escape-sequence*
3 | *interpolated-character-sequence*

4 *regular-expression-option* ::
5 i | m

6 **A.2 Program structure**

7 **A.2.1 Program**

8 see §10.1

9 *program* ::
10 *compound-statement*

11 **A.3 Expressions**

12 **A.3.1 General description**

13 see §11.1

14 *expression* ::
15 *keyword-logical-expression*

16 **A.3.2 Logical expressions**

17 see §11.2

18 *keyword-logical-expression* ::
19 *keyword-NOT-expression*
20 | *keyword-AND-expression*
21 | *keyword-OR-expression*

22 *keyword-NOT-expression* ::
23 *method-invocation-without-parentheses*
24 | *operator-expression*
25 | *logical-NOT-with-method-invocation-without-parentheses*
26 | **not** *keyword-NOT-expression*

27 *logical-NOT-expression* ::=
28 *logical-NOT-with-method-invocation-without-parentheses*
29 | *logical-NOT-with-unary-expression*

1 *logical-NOT-with-method-invocation-without-parentheses* ::
2 ! *method-invocation-without-parentheses*

3 *logical-NOT-with-unary-expression* ::
4 ! *unary-expression*

5 *keyword-AND-expression* ::
6 *expression* **and** *keyword-NOT-expression*

7 *keyword-OR-expression* ::
8 *expression* **or** *keyword-NOT-expression*

9 *logical-OR-expression* ::
10 *logical-AND-expression*
11 | *logical-OR-expression* || *logical-AND-expression*

12 *logical-AND-expression* ::
13 *equality-expression*
14 | *logical-AND-expression* **&&** *equality-expression*

15 **A.3.2.1 General description**

16 see §11.3.1

17 *primary-method-invocation* ::
18 *super-with-optional-argument*
19 | *indexing-method-invocation*
20 | *method-only-identifier*
21 | *method-identifier* ([no *whitespace* here] *argument-with-parentheses*)? *block*?
22 | *primary-expression* [no *line-terminator* here]
23 . *method-name* ([no *whitespace* here] *argument-with-parentheses*)? *block*?
24 | *primary-expression* [no *line-terminator* here]
25 :: *method-name* [no *whitespace* here] *argument-with-parentheses* *block*?
26 | *primary-expression* [no *line-terminator* here] :: *method-name-without-constant*
27 *block*?

28 *indexing-method-invocation* ::
29 *primary-expression* [no *line-terminator* here] *optional-whitespace*?
30 [*indexing-argument-list*?]

31 *optional-whitespace* ::
32 [*whitespace* here]

33 *method-name-without-constant* ::
34 *method-name* **but not** *constant-identifier*


```

1  method-invocation-without-parentheses ::
2      command
3      | chained-command-with-do-block
4      | chained-command-with-do-block ( . | :: ) method-name argument
5      | return-with-argument
6      | break-with-argument
7      | next-with-argument

8  command ::
9      super-with-argument
10     | yield-with-argument
11     | method-identifier argument
12     | primary-expression [no line-terminator here] ( . | :: ) method-name argument

13 chained-command-with-do-block ::
14     command-with-do-block chained-method-invocation*

15 chained-method-invocation ::
16     ( . | :: ) method-name
17     | ( . | :: ) method-name [no whitespace here]
18     [lookahead ∉ { { } ] argument-with-parentheses

19 command-with-do-block ::
20     super-with-argument-and-do-block
21     | method-identifier argument do-block
22     | primary-expression [no line-terminator here] ( . | :: ) method-name argument
23     do-block

```

24 A.3.2.2 Method arguments

25 see §11.3.2

```

26 indexing-argument-list ::
27     command
28     | operator-expression-list ,?
29     | operator-expression-list , splatting-argument
30     | association-list ,?
31     | splatting-argument

32 splatting-argument ::
33     * operator-expression

34 operator-expression-list ::
35     operator-expression ( , operator-expression )*

```

```

1  argument-with-parentheses ::
2      ( )
3      | ( argument-in-parentheses )
4      | ( operator-expression-list , chained-command-with-do-block )
5      | ( chained-command-with-do-block )

6  argument ::
7      [no line-terminator here] [lookahead ∉ { { }] optional-whitespace?
8      argument-in-parentheses

9  argument-in-parentheses ::
10     command
11     | ( operator-expression-list | association-list )
12         ( , splattling-argument )? ( , block-argument )?
13     | operator-expression-list , association-list
14         ( , splattling-argument )? ( , block-argument )?
15     | splattling-argument ( , block-argument )?
16     | block-argument

17  block-argument ::
18     & operator-expression

```

19 A.3.2.3 Blocks

20 see §11.3.3

```

21  block ::
22     brace-block
23     | do-block

24  brace-block ::
25     { block-parameter? block-body }

26  do-block ::
27     do block-parameter? block-body end

28  block-parameter ::
29     | |
30     | ||
31     | | block-parameter-list |

32  block-parameter-list ::
33     left-hand-side
34     | multiple-left-hand-side

```

1 *block-body* ::
2 *compound-statement*

3 **A.3.2.4 The super expression**

4 see §11.3.4

5 *super-expression* ::=
6 *super-with-optional-argument*
7 | *super-with-argument*
8 | *super-with-argument-and-do-block*

9 *super-with-optional-argument* ::
10 **super** ([no *whitespace* here] *argument-with-parentheses*)? *block*?

11 *super-with-argument* ::
12 **super** *argument*

13 *super-with-argument-and-do-block* ::
14 **super** *argument do-block*

15 **A.3.2.5 The yield expression**

16 see §11.3.5

17 *yield-expression* ::=
18 *yield-with-optional-argument*
19 | *yield-with-argument*

20 *yield-with-optional-argument* ::
21 *yield-with-parentheses-and-argument*
22 | *yield-with-parentheses-without-argument*
23 | **yield**

24 *yield-with-parentheses-and-argument* ::
25 **yield** [no *whitespace* here] (*argument-in-parentheses*)

26 *yield-with-parentheses-without-argument* ::
27 **yield** [no *whitespace* here] ()

28 *yield-with-argument* ::
29 **yield** *argument*

1 **A.3.2.6 General description**

2 see §11.4.1

3 *operator-expression* ::
4 *assignment-expression*
5 | *defined?-without-parentheses*
6 | *conditional-operator-expression*

7 **A.3.2.6.1 General description**

8 see §11.4.2.1

9 *assignment* ::=
10 *assignment-expression*
11 | *assignment-statement*

12 *assignment-expression* ::
13 *single-assignment-expression*
14 | *abbreviated-assignment-expression*
15 | *assignment-with-rescue-modifier*

16 *assignment-statement* ::
17 *single-assignment-statement*
18 | *abbreviated-assignment-statement*
19 | *multiple-assignment-statement*

20 **A.3.2.6.1.1 General description**

21 see §11.4.2.2.1

22 *single-assignment* ::=
23 *single-assignment-expression*
24 | *single-assignment-statement*

25 *single-assignment-expression* ::
26 *single-variable-assignment-expression*
27 | *scoped-constant-assignment-expression*
28 | *single-indexing-assignment-expression*
29 | *single-method-assignment-expression*

30 *single-assignment-statement* ::
31 *single-variable-assignment-statement*
32 | *scoped-constant-assignment-statement*

1 | *single-indexing-assignment-statement*
2 | *single-method-assignment-statement*

3 **A.3.2.6.1.2 Single variable assignments**

4 see §11.4.2.2.2

5 *single-variable-assignment ::=*
6 *single-variable-assignment-expression*
7 | *single-variable-assignment-statement*

8 *single-variable-assignment-expression ::=*
9 *variable* [no *line-terminator* here] = *operator-expression*

10 *single-variable-assignment-statement ::=*
11 *variable* [no *line-terminator* here] = *method-invocation-without-parentheses*

12 *scoped-constant-assignment ::=*
13 *scoped-constant-assignment-expression*
14 | *scoped-constant-assignment-statement*

15 *scoped-constant-assignment-expression ::=*
16 *primary-expression* [no *whitespace* here] :: *constant-identifier*
17 [no *line-terminator* here] = *operator-expression*
18 | :: *constant-identifier* [no *line-terminator* here] = *operator-expression*

19 *scoped-constant-assignment-statement ::=*
20 *primary-expression* [no *whitespace* here] :: *constant-identifier*
21 [no *line-terminator* here] = *method-invocation-without-parentheses*
22 | :: *constant-identifier* [no *line-terminator* here] = *method-invocation-without-parentheses*

23 **A.3.2.6.1.3 Single indexing assignments**

24 see §11.4.2.2.4

25 *single-indexing-assignment ::=*
26 *single-indexing-assignment-expression*
27 | *single-indexing-assignment-statement*

28 *single-indexing-assignment-expression ::=*
29 *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
30 [no *line-terminator* here] = *operator-expression*

1 *single-indexing-assignment-statement* ::
2 *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
3 [no *line-terminator* here] = *method-invocation-without-parentheses*

4 **A.3.2.6.1.4** Single method assignments

5 see §11.4.2.2.5

6 *single-method-assignment* ::=
7 *single-method-assignment-expression*
8 | *single-method-assignment-statement*

9 *single-method-assignment-expression* ::
10 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
11 [no *line-terminator* here] = *operator-expression*
12 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
13 [no *line-terminator* here] = *operator-expression*

14 *single-method-assignment-statement* ::
15 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
16 [no *line-terminator* here] = *method-invocation-without-parentheses*
17 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
18 [no *line-terminator* here] = *method-invocation-without-parentheses*

19 **A.3.2.6.1.5** General description

20 see §11.4.2.3.1

21 *abbreviated-assignment* ::=
22 *abbreviated-assignment-expression*
23 | *abbreviated-assignment-statement*

24 *abbreviated-assignment-expression* ::
25 *abbreviated-variable-assignment-expression*
26 | *abbreviated-indexing-assignment-expression*
27 | *abbreviated-method-assignment-expression*

28 *abbreviated-assignment-statement* ::
29 *abbreviated-variable-assignment-statement*
30 | *abbreviated-indexing-assignment-statement*
31 | *abbreviated-method-assignment-statement*

1 **A.3.2.6.1.6 Abbreviated variable assignments**

2 see §11.4.2.3.2

3 *abbreviated-variable-assignment ::=*
4 *abbreviated-variable-assignment-expression*
5 *| abbreviated-variable-assignment-statement*

6 *abbreviated-variable-assignment-expression ::*
7 *variable [no line-terminator here] assignment-operator operator-expression*

8 *abbreviated-variable-assignment-statement ::*
9 *variable [no line-terminator here] assignment-operator*
10 *method-invocation-without-parentheses*

11 **A.3.2.6.1.7 Abbreviated indexing assignments**

12 see §11.4.2.3.3

13 *abbreviated-indexing-assignment ::=*
14 *abbreviated-indexing-assignment-expression*
15 *| abbreviated-indexing-assignment-statement*

16 *abbreviated-indexing-assignment-expression ::*
17 *primary-expression [no line-terminator here] [indexing-argument-list?]*
18 *[no line-terminator here] assignment-operator operator-expression*

19 *abbreviated-indexing-assignment-statement ::*
20 *primary-expression [no line-terminator here] [indexing-argument-list?]*
21 *[no line-terminator here] assignment-operator method-invocation-without-parentheses*

22 **A.3.2.6.1.8 Abbreviated method assignments**

23 see §11.4.2.3.4

24 *abbreviated-method-assignment ::=*
25 *abbreviated-method-assignment-expression*
26 *| abbreviated-method-assignment-statement*

27 *abbreviated-method-assignment-expression ::*
28 *primary-expression [no line-terminator here] (. | ::) local-variable-identifier*
29 *[no line-terminator here] assignment-operator operator-expression*
30 *| primary-expression [no line-terminator here] . constant-identifier*
31 *[no line-terminator here] assignment-operator operator-expression*

1 *abbreviated-method-assignment-statement* ::
 2 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
 3 [no *line-terminator* here] *assignment-operator* *method-invocation-without-parentheses*
 4 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
 5 [no *line-terminator* here] *assignment-operator* *method-invocation-without-parentheses*

6 A.3.2.6.2 Multiple assignments

7 see §11.4.2.4

8 *multiple-assignment-statement* ::
 9 *many-to-one-assignment-statement*
 10 | *one-to-packing-assignment-statement*
 11 | *many-to-many-assignment-statement*

12 *many-to-one-assignment-statement* ::
 13 *left-hand-side* [no *line-terminator* here] = *multiple-right-hand-side*

14 *one-to-packing-assignment-statement* ::
 15 *packing-left-hand-side* [no *line-terminator* here] =
 16 (*method-invocation-without-parentheses* | *operator-expression*)

17 *many-to-many-assignment-statement* ::
 18 *multiple-left-hand-side* [no *line-terminator* here] = *multiple-right-hand-side*
 19 | (*multiple-left-hand-side* **but not** *packing-left-hand-side*)
 20 [no *line-terminator* here] =
 21 (*method-invocation-without-parentheses* | *operator-expression*)

22 *left-hand-side* ::
 23 *variable*
 24 | *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
 25 | *primary-expression* [no *line-terminator* here]
 26 (. | ::) (*local-variable-identifier* | *constant-identifier*)
 27 | :: *constant-identifier*

28 *multiple-left-hand-side* ::
 29 (*multiple-left-hand-side-item* ,)+ *multiple-left-hand-side-item?*
 30 | (*multiple-left-hand-side-item* ,)+ *packing-left-hand-side?*
 31 | *packing-left-hand-side*
 32 | *grouped-left-hand-side*

33 *packing-left-hand-side* ::
 34 * *left-hand-side?*

35 *grouped-left-hand-side* ::
 36 (*multiple-left-hand-side*)

1 *multiple-left-hand-side-item* ::
2 *left-hand-side*
3 | *grouped-left-hand-side*

4 *multiple-right-hand-side* ::
5 *operator-expression-list* (, *splatting-right-hand-side*)?
6 | *splatting-right-hand-side*

7 *splatting-right-hand-side* ::
8 *splatting-argument*

9 **A.3.2.6.3 Assignments with rescue modifiers**

10 see §11.4.2.5

11 *assignment-with-rescue-modifier* ::
12 *left-hand-side* [no line-terminator here] =
13 *operator-expression*₁ **rescue** *operator-expression*₂

14 **A.3.2.6.4 General description**

15 see §11.4.3.1

16 *unary-minus-expression* ::
17 *power-expression*₁
18 | - *power-expression*₂

19 *unary-expression* ::
20 *primary-expression*
21 | *logical-NOT-with-unary-expression*
22 | ~ *unary-expression*₁
23 | + *unary-expression*₂

24 **A.3.2.6.5 The defined? expression**

25 see §11.4.3.2

26 *defined?-expression* ::=
27 *defined?-with-parentheses*
28 | *defined?-without-parentheses*

29 *defined?-with-parentheses* ::
30 **defined?** (*expression*)

1 *defined?-without-parentheses* ::
2 *defined?* *operator-expression*

3 **A.3.2.7 Binary operators**

4 see §11.4.4

5 *equality-expression* ::
6 *relational-expression*
7 | *relational-expression* *<=>* *relational-expression*
8 | *relational-expression* *==* *relational-expression*
9 | *relational-expression* *===* *relational-expression*
10 | *relational-expression* *!=* *relational-expression*
11 | *relational-expression* *=~* *relational-expression*
12 | *relational-expression* *!~* *relational-expression*

13 *relational-expression* ::
14 *bitwise-OR-expression*
15 | *relational-expression* *>* *bitwise-OR-expression*
16 | *relational-expression* *>=* *bitwise-OR-expression*
17 | *relational-expression* *<* *bitwise-OR-expression*
18 | *relational-expression* *<=* *bitwise-OR-expression*

19 *bitwise-OR-expression* ::
20 *bitwise-AND-expression*
21 | *bitwise-OR-expression* *|* *bitwise-AND-expression*
22 | *bitwise-OR-expression* *^* *bitwise-AND-expression*

23 *bitwise-AND-expression* ::
24 *bitwise-shift-expression*
25 | *bitwise-AND-expression* *whitespace-before-operator?* *&* *bitwise-shift-expression*

26 *bitwise-shift-expression* ::
27 *additive-expression*
28 | *bitwise-shift-expression* *whitespace-before-operator?* *<<* *additive-expression*
29 | *bitwise-shift-expression* *>>* *additive-expression*

30 *additive-expression* ::
31 *multiplicative-expression*
32 | *additive-expression* *whitespace-before-operator?* *+* *multiplicative-expression*
33 | *additive-expression* *whitespace-before-operator?* *-* *multiplicative-expression*

34 *multiplicative-expression* ::
35 *unary-minus-expression*
36 | *multiplicative-expression* *whitespace-before-operator?* *** *unary-minus-expression*
37 | *multiplicative-expression* *whitespace-before-operator?* */* *unary-minus-expression*
38 | *multiplicative-expression* *whitespace-before-operator?* *%* *unary-minus-expression*

```

1  power-expression ::=
2      unary-expression
3      | - ( numeric-literal ) ** power-expression
4      | unary-expression ** power-expression

5  binary-operator ::=
6      <=> | == | === | =~ | > | >= | < | <= | | | ^
7      | & | << | >> | + | - | * | / | % | ** | != | !~

8  whitespace-before-operator ::=
9      [ whitespace here ]

```

10 A.3.2.8 General description

11 see §11.5.1

```

12 primary-expression ::=
13     class-definition
14     | eigenclass-definition
15     | module-definition
16     | method-definition
17     | singleton-method-definition
18     | yield-with-optional-argument
19     | if-expression
20     | unless-expression
21     | case-expression
22     | while-expression
23     | until-expression
24     | for-expression
25     | return-without-argument
26     | break-without-argument
27     | next-without-argument
28     | redo-expression
29     | retry-expression
30     | rescue-expression
31     | grouping-expression
32     | variable-reference
33     | scoped-constant-reference
34     | array-constructor
35     | hash-constructor
36     | literal
37     | defined?-with-parentheses
38     | primary-method-invocation

```

39 A.3.2.8.0.1 The if expression

40 see §11.5.2.1.1

1 *if-expression* ::
2 *if expression then-clause elsif-clause* else-clause? end*

3 *then-clause* ::
4 *separator compound-statement*
5 | *separator? then compound-statement*

6 *else-clause* ::
7 *else compound-statement*

8 *elsif-clause* ::
9 *elsif expression then-clause*

10 **A.3.2.8.0.2 The unless expression**

11 see §11.5.2.1.2

12 *unless-expression* ::
13 *unless expression then-clause else-clause? end*

14 **A.3.2.8.0.3 The case expression**

15 see §11.5.2.1.3

16 *case-expression* ::
17 *case-expression-with-expression*
18 | *case-expression-without-expression*

19 *case-expression-with-expression* ::
20 *case expression separator-list? when-clause + else-clause? end*

21 *case-expression-without-expression* ::
22 *case separator-list? when-clause + else-clause? end*

23 *when-clause* ::
24 *when when-argument then-clause*

25 *when-argument* ::
26 *operator-expression-list (, splatting-argument)?*
27 | *splatting-argument*

1 **A.3.2.8.0.4 Conditional operator**

2 see §11.5.2.1.4

3 *conditional-operator-expression* ::
4 *range-constructor*
5 | *range-constructor* ? *operator-expression*₁ : *operator-expression*₂

6 **A.3.2.8.0.5 General description**

7 see §11.5.2.2.1

8 *iteration-expression* ::=
9 *while-expression*
10 | *until-expression*
11 | *for-expression*
12 | *while-modifier-statement*
13 | *until-modifier-statement*

14 **A.3.2.8.0.6 The while expression**

15 see §11.5.2.2.2

16 *while-expression* ::
17 **while** *expression* *do-clause* **end**

18 *do-clause* ::
19 *separator* *compound-statement*
20 | **do** *compound-statement*

21 **A.3.2.8.0.7 The until expression**

22 see §11.5.2.2.3

23 *until-expression* ::
24 **until** *expression* *do-clause* **end**

25 **A.3.2.8.0.8 The for expression**

26 see §11.5.2.2.4

1 *for-expression* ::
2 **for** *for-variable* **in** *expression* **do-clause** **end**

3 *for-variable* ::
4 *left-hand-side*
5 | *multiple-left-hand-side*

6 **A.3.2.8.0.9** General description

7 see §11.5.2.3.1

8 *jump-expression* ::=
9 *return-expression*
10 | *break-expression*
11 | *next-expression*
12 | *redo-expression*
13 | *retry-expression*

14 **A.3.2.8.0.10** The return expression

15 see §11.5.2.3.2

16 *return-expression* ::=
17 *return-without-argument*
18 | *return-with-argument*

19 *return-without-argument* ::
20 **return**

21 *return-with-argument* ::
22 **return** *jump-argument*

23 *jump-argument* ::
24 *argument*

25 **A.3.2.8.0.11** The break expression

26 see §11.5.2.3.3

27 *break-expression* ::=
28 *break-without-argument*
29 | *break-with-argument*

1 *break-without-argument* ::
2 **break**

3 *break-with-argument* ::
4 **break** *jump-argument*

5 **A.3.2.8.0.12 The next expression**

6 see §11.5.2.3.4

7 *next-expression* ::=
8 *next-without-argument*
9 | *next-with-argument*

10 *next-without-argument* ::
11 **next**

12 *next-with-argument* ::
13 **next** *jump-argument*

14 **A.3.2.8.0.13 The redo expression**

15 see §11.5.2.3.5

16 *redo-expression* ::
17 **redo**

18 **A.3.2.8.0.14 The retry expression**

19 see §11.5.2.3.6

20 *retry-expression* ::
21 **retry**

22 **A.3.2.8.0.15 The rescue expression**

23 see §11.5.2.4.1

24 *rescue-expression* ::
25 **begin** *body-statement* **end**

1 *body-statement* ::
2 *compound-statement* *rescue-clause** *else-clause*? *ensure-clause*?

3 *rescue-clause* ::
4 **rescue** [no *line-terminator* here] *exception-class-list*?
5 *exception-variable-assignment*? *then-clause*

6 *exception-class-list* ::
7 *operator-expression*
8 | *multiple-right-hand-side*

9 *exception-variable-assignment* ::
10 => *left-hand-side*

11 *ensure-clause* ::
12 **ensure** *compound-statement*

13 **A.3.2.9** Grouping expression

14 see §11.5.3

15 *grouping-expression* ::
16 (*expression*)
17 | (*compound-statement*)

18 **A.3.2.9.1** General description

19 see §11.5.4.1

20 *variable-reference* ::
21 *variable*
22 | *pseudo-variable*

23 *variable* ::
24 *constant-identifier*
25 | *global-variable-identifier*
26 | *class-variable-identifier*
27 | *instance-variable-identifier*
28 | *local-variable-identifier*

29 *scoped-constant-reference* ::
30 *primary-expression* [no *whitespace* here] :: *constant-identifier*
31 | :: *constant-identifier*

1 **A.3.2.9.1.1 General description**

2 see §11.5.4.8.1

3 *pseudo-variable* ::
4 *nil*
5 | *true*
6 | *false*
7 | *self*

8 **A.3.2.9.1.2 The nil expression**

9 see §11.5.4.8.2

10 *nil-expression* ::
11 **nil**

12 **A.3.2.9.1.3 The true expression and the false expression**

13 see §11.5.4.8.3

14 *true-expression* ::
15 **true**

16 *false-expression* ::
17 **false**

18 **A.3.2.9.1.4 The self expression**

19 see §11.5.4.8.4

20 *self-expression* ::
21 **self**

22 **A.3.2.9.2 Array constructor**

23 see §11.5.5.1

24 *array-constructor* ::
25 [*indexing-argument-list?*]

1 **A.3.2.9.3 Hash constructor**

2 see §11.5.5.2

3 *hash-constructor* ::
4 { (*association-list* ,?)? }

5 *association-list* ::
6 *association* (, *association*)*

7 *association* ::
8 *association-key* => *association-value*

9 *association-key* ::
10 *operator-expression*

11 *association-value* ::
12 *operator-expression*

13 **A.3.2.9.4 Range constructor**

14 see §11.5.5.3

15 *range-constructor* ::
16 *logical-OR-expression*
17 | *logical-OR-expression*₁ *range-operator* *logical-OR-expression*₂

18 *range-operator* ::
19 ..
20 | ...

21 **A.4 Statements**

22 **A.4.1 General description**

23 see §12.1

24 *statement* ::
25 *expression-statement*
26 | *alias-statement*
27 | *undef-statement*
28 | *if-modifier-statement*
29 | *unless-modifier-statement*
30 | *while-modifier-statement*

1 | *until-modifier-statement*
2 | *rescue-modifier-statement*
3 | *assignment-statement*

4 **A.4.2 The expression statement**

5 see §12.2

6 *expression-statement* ::
7 *expression*

8 **A.4.3 The if modifier statement**

9 see §12.3

10 *if-modifier-statement* ::
11 *statement* [no *line-terminator* here] **if** *expression*

12 **A.4.4 The unless modifier statement**

13 see §12.4

14 *unless-modifier-statement* ::
15 *statement* [no *line-terminator* here] **unless** *expression*

16 **A.4.5 The while modifier statement**

17 see §12.5

18 *while-modifier-statement* ::
19 *statement* [no *line-terminator* here] **while** *expression*

20 **A.4.6 The until modifier statement**

21 see §12.6

22 *until-modifier-statement* ::
23 *statement* [no *line-terminator* here] **until** *expression*

1 A.5 Classes and modules

2 A.5.0.1 Module definition

3 see §13.1.2

4 *module-definition* ::
5 **module** *module-path* *module-body* **end**

6 *module-path* ::
7 *top-module-path*
8 | *module-name*
9 | *nested-module-path*

10 *module-name* ::
11 *constant-identifier*

12 *top-module-path* ::
13 :: *module-name*

14 *nested-module-path* ::
15 *primary-expression* [no line-terminator here] :: *module-name*

16 *module-body* ::
17 *body-statement*

18 A.5.0.2 Class definition

19 see §13.2.2

20 *class-definition* ::
21 **class** *class-path* [no line-terminator here] (< *superclass*)? *separator*
22 *class-body* **end**

23 *class-path* ::
24 *top-class-path*
25 | *class-name*
26 | *nested-class-path*

27 *class-name* ::
28 *constant-identifier*

29 *top-class-path* ::
30 :: *class-name*

1 *nested-class-path* ::
2 *primary-expression* [no line-terminator here] :: *class-name*

3 *superclass* ::
4 *expression*

5 *class-body* ::
6 *body-statement*

7 **A.5.0.3 Method definition**

8 see §13.3.1

9 *method-definition* ::
10 **def** *method-name* [no line-terminator here] *method-parameter-part*
11 *method-body* **end**

12 *method-name* ::
13 *method-identifier*
14 | *operator-method-name*
15 | *keyword*

16 *method-body* ::
17 *body-statement*

18 **A.5.0.4 Method parameters**

19 see §13.3.2

20 *method-parameter-part* ::
21 (*parameter-list?*)
22 | *parameter-list?* *separator*

23 *parameter-list* ::
24 *mandatory-parameter-list* , *optional-parameter-list?* ,
25 *array-parameter?* , *proc-parameter?*
26 | *optional-parameter-list* , *array-parameter?* , *proc-parameter?*
27 | *array-parameter* , *proc-parameter?*
28 | *proc-parameter*

29 *mandatory-parameter-list* ::
30 *mandatory-parameter*
31 | *mandatory-parameter-list* , *mandatory-parameter*

1 *mandatory-parameter* ::
2 *local-variable-identifier*

3 *optional-parameter-list* ::
4 *optional-parameter*
5 | *optional-parameter-list* , *optional-parameter*

6 *optional-parameter* ::
7 *optional-parameter-name* = *default-parameter-expression*

8 *optional-parameter-name* ::
9 *local-variable-identifier*

10 *default-parameter-expression* ::
11 *operator-expression*

12 *array-parameter* ::
13 * *array-parameter-name*
14 | *

15 *array-parameter-name* ::
16 *local-variable-identifier*

17 *proc-parameter* ::
18 & *proc-parameter-name*

19 *proc-parameter-name* ::
20 *local-variable-identifier*

21 **A.5.0.5 The alias statement**

22 see §13.3.6

23 *alias-statement* ::
24 **alias** *new-name* *aliased-name*

25 *new-name* ::
26 *method-name*
27 | *symbol*

28 *aliased-name* ::
29 *method-name*
30 | *symbol*

1 **A.5.0.6 The undef statement**

2 see §13.3.7

3 *undef-statement* ::
4 **undef** *undef-list*

5 *undef-list* ::
6 *method-name-or-symbol* (, *method-name-or-symbol*)*

7 *method-name-or-symbol* ::
8 *method-name*
9 | *symbol*

10 **A.5.0.7 Eigenclass definition**

11 see §13.4.2

12 *eigenclass-definition* ::
13 **class** << *expression separator eigenclass-body end*

14 *eigenclass-body* ::
15 *body-statement*

16 **A.5.0.7.1 Patterns**

17 see §15.2.15.4

18 *pattern* ::
19 *alternative*₁
20 | *pattern*₁ | *alternative*₂

21 *alternative* ::
22 [empty]
23 | *alternative*₃ *term*

24 *term* ::
25 *anchor*
26 | *atom*₁
27 | *atom*₂ *quantifier*

28 *anchor* ::
29 *left-anchor* | *right-anchor*

```

1  left-anchor ::
2      \A | ^

3  right-anchor ::
4      \z | $

5  quantifier ::
6      * | + | ?

7  atom ::
8      pattern-character
9      | grouping
10     | .
11     | atom-escape-sequence

12 pattern-character ::
13     source-character but not regexp-meta-character

14 regexp-meta-character ::
15     | | . | * | + | ^ | ? | ( | ) | # | \
16     | future-reserved-meta-character

17 future-reserved-meta-character ::
18     [ | ] | { | }

19 grouping ::
20     ( pattern )

21 atom-escape-sequence ::
22     decimal-escape-sequence
23     | regexp-character-escape-sequence

24 decimal-escape-sequence ::
25     \ decimal-digit-without-zero

26 regexp-character-escape-sequence ::
27     regexp-escape-sequence
28     | regexp-non-escaped-sequence
29     | hex-escape-sequence
30     | regexp-octal-escape-sequence
31     | regexp-control-escape-sequence

32 regexp-escape-sequence ::
33     \ regexp-escaped-character

```



```
1  regexp-escaped-character ::
2      n | t | r | f | v | a | e

3  regexp-non-escaped-sequence ::
4      \ regexp-meta-character

5  regexp-octal-escape-sequence ::
6      octal-escape-sequence but not decimal-escape-sequence

7  regexp-control-escape-sequence ::
8      \ ( C - | c ) regexp-control-escaped-character

9  regexp-control-escaped-character ::
10     regexp-character-escape-sequence
11     | ?
12     | source-character but not ( \ | ? )
```
