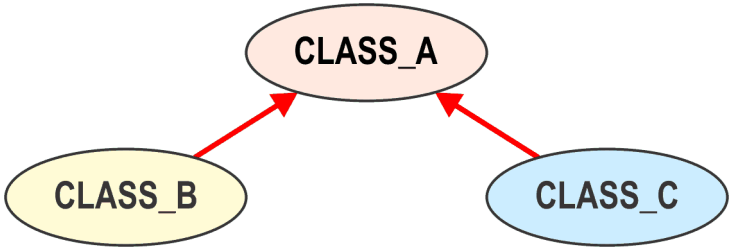
Re-What?

A Wicked Basic Guide to Inheritance in Eiffel

Roger F. Osmond

The Wicked Basics

- Eiffel supports class-based inheritance
  - A class can inherit features (attributes and routines) from another class

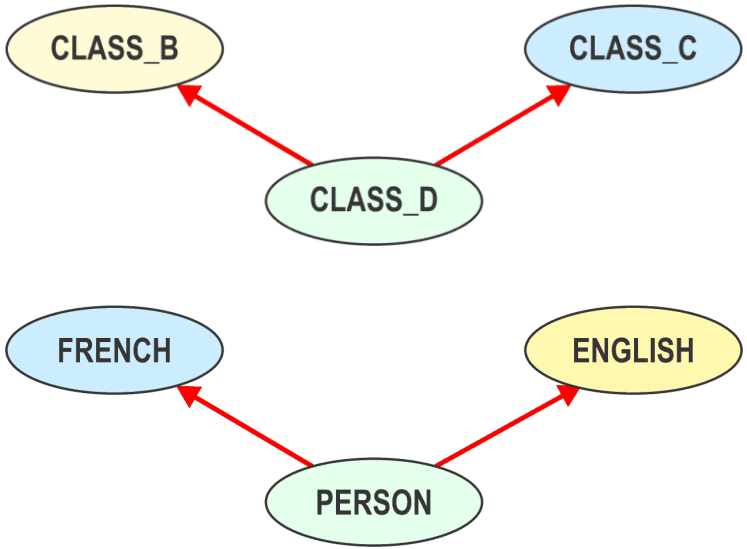


- Eiffel offers mechanisms to adapt and select inherited features
  - Rename, redefine, undefine, select, export

- Eiffel supports multiple inheritance and repeated inheritance

Multiple Inheritance

- Multiple inheritance enables a class to inherit from more than one parent
  - If you're new to Object-Oriented design, this should make perfect sense

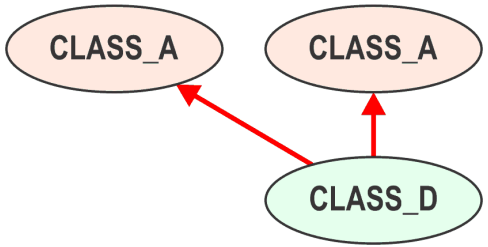


Multiple Inheritance

- Multiple inheritance enables a class to inherit from more than one parent
  - If you're new to Object-Oriented design, this should make perfect sense
  - If you've used other Object-Oriented languages, this might scare you, a little
    - Don't worry. It will be OK.

Repeated Inheritance

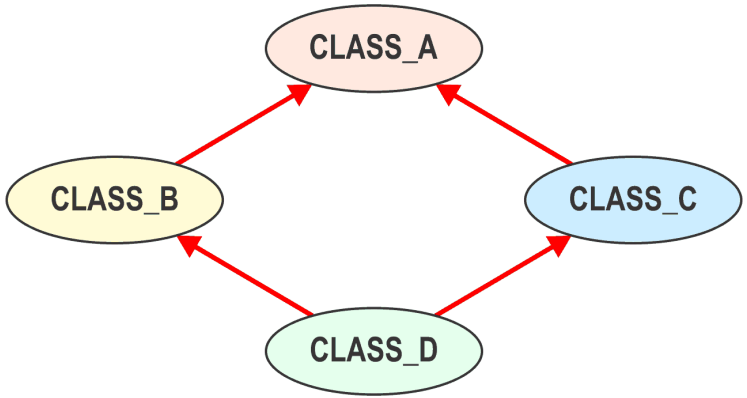
- Repeated inheritance occurs when a class inherits another class more than once, directly



Repeated Inheritance

- Repeated inheritance occurs when a class inherits another class more than once, directly

  or indirectly



Wicked Basic Syntax

- The Inherit Clause

  (No need to consult a lawyer)

  - Begin with the ‘inherit’ keyword
  - Identify parent class
  - Include appropriate keywords to identify nature of adaptations, if any
  - Complete adaptation clauses
  - Terminate, per parent, with ‘end’ keyword
  - Repeat for each additional parent

```
class CLASS_A
 inherit
  CLASS_B
   rename <old_name> as <new_name>
   export <export_scope> <features>
   undefine <features>
   redefine <features>
   select <features>
  end
  ...
```

Adapting Inherited Features - Rename

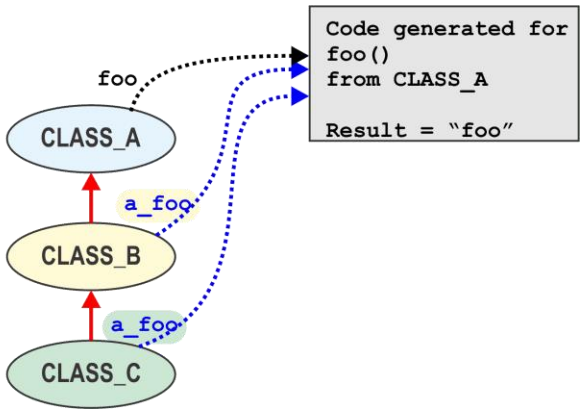
- For name conflict resolution, augmenting implementation, or cosmetics
- Inherited feature has new name in child and its descendants
- Inherited feature implementation is unchanged

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  rename
   foo as a_foo
  end
feature




end
```

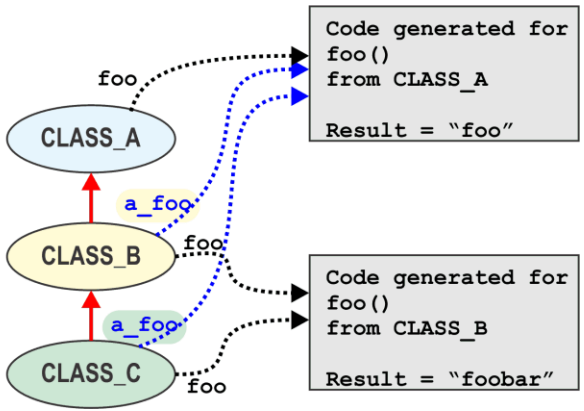


Adapting Inherited Features - Rename

- For name conflict resolution, augmenting implementation, or cosmetics
- Inherited feature has new name in child and its descendants
- Inherited feature implementation is unchanged

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  rename
   foo as a_foo
  end
feature
  foo: STRING
   do
    Result := a_foo + "bar"
   end
end
```



Descendent is free to re-use the original name

Adapting Inherited Features - Export

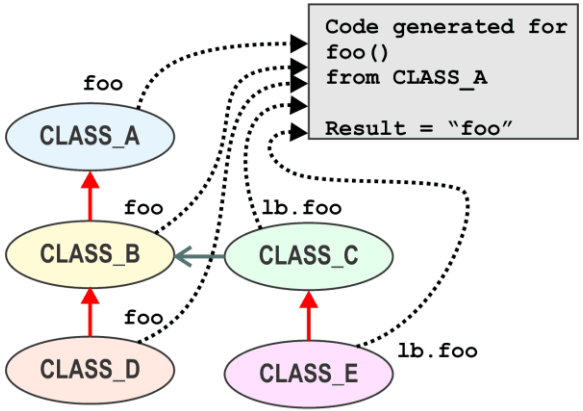
- Change export status of inherited features
  - Inherited feature's name and implementation are unchanged

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A


  end
end
```

```
class CLASS_C
feature
 bar
  local
   lb: CLASS_B
  do
   create lb
   print (lb.foo)
  end
end
```



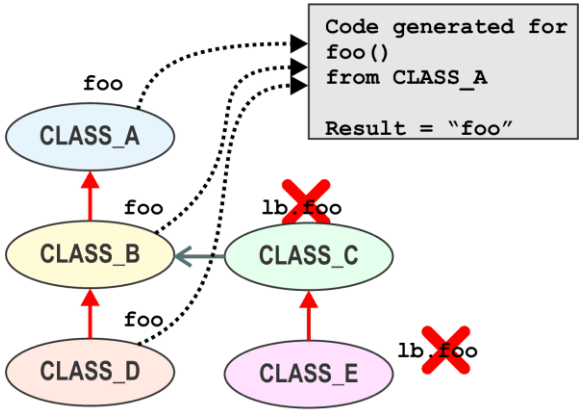
Adapting Inherited Features - Export

- Change export status of inherited features
  - Inherited feature's name and implementation are unchanged
- Restrict export status

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  export {NONE}
   foo
  end
end
```

```
class CLASS_C
feature
 bar
  local
   lb: CLASS_B
  do
   create lb
   print (lb.foo)
  end
end
```



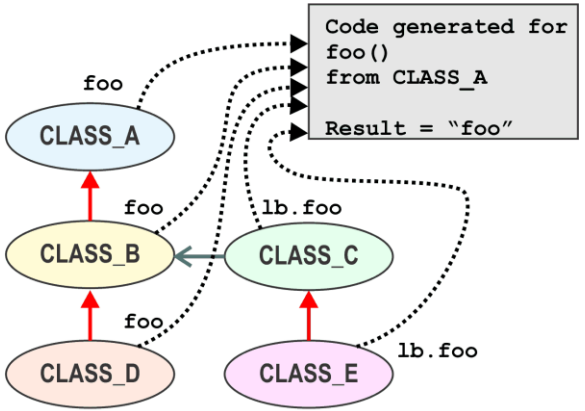
Adapting Inherited Features - Export

- Change export status of inherited features
  - Inherited feature's name and implementation are unchanged
- Restrict export status
- Expand export status

```
class CLASS_A
feature {NONE}
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  export {CLASS_C}
   foo
  end
end
```

```
class CLASS_C
feature
 bar
  local
   lb: CLASS_B
  do
   create lb
   print (lb.foo)
  end
end
```



Adapting Inherited Features - Undefine

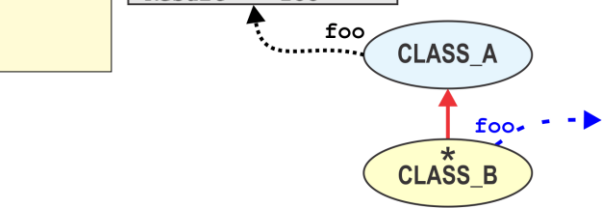
- Removes implementation of inherited feature in undefining class
  - Conceptually, turning feature into a deferred routine
  - Only routines (functions and procedures) can be undefined
- Undefining class can defer implementation to a descendent

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  undefine
   foo
  end
end
```

```
Code generated for
foo()
from CLASS_A

Result = "foo"
```



Adapting Inherited Features – Undefine (cont'd)

- Undefine is used most often to resolve conflicts between inherited features
  - When names are the same, but implementations are different, and you want only one
    - Will not compile until conflict is resolved

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_C
feature
 foo: STRING
  do
   Result := "bar"
  end
end
```

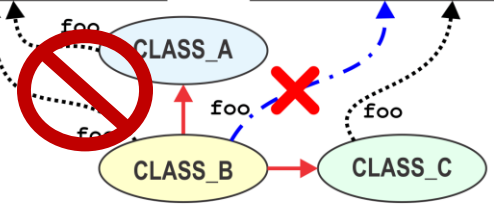
```
class CLASS_B
inherit
 CLASS_A
 CLASS_C
  undefine
   foo
  end
end
```

```
Code generated for
foo()
from CLASS_A

Result = "foo"
```

```
Code generated for
foo()
from CLASS_C

Result = "bar"
```



Adapting Inherited Features – Undefine (cont'd)

- Undefine is used most often to resolve conflicts between inherited features
  - When names are the same, but implementations are different, and you want only one
    - Will not compile until conflict is resolved
  - Undefine the one you don't want

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_C
feature
 foo: STRING
  do
   Result := "bar"
  end
end
```

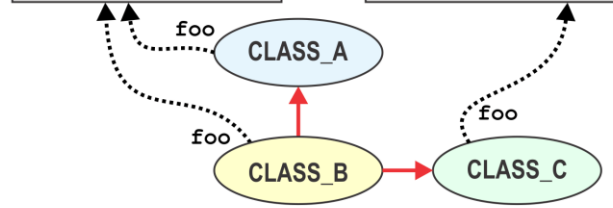
```
class CLASS_B
inherit
 CLASS_A
 CLASS_C
  undefine
   foo
  end
end
```

```
Code generated for
foo()
from CLASS_A

Result = "foo"
```

```
Code generated for
foo()
from CLASS_C

Result = "bar"
```

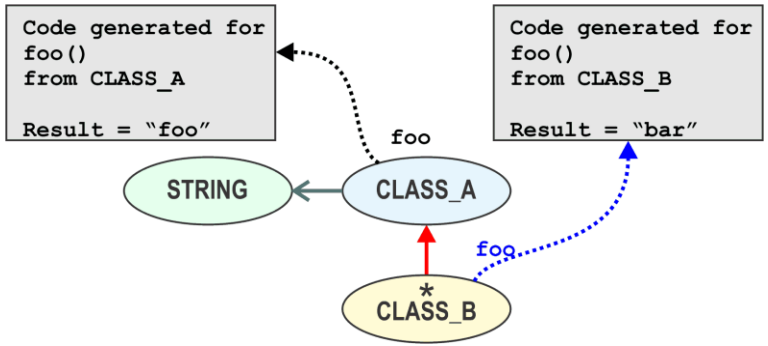


Adapting Inherited Features - Redefine

- Redefine can change the implementation of an inherited feature

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  redefine
   foo
  end
feature
 foo: STRING
  do
   Result := "bar"
  end
end
```

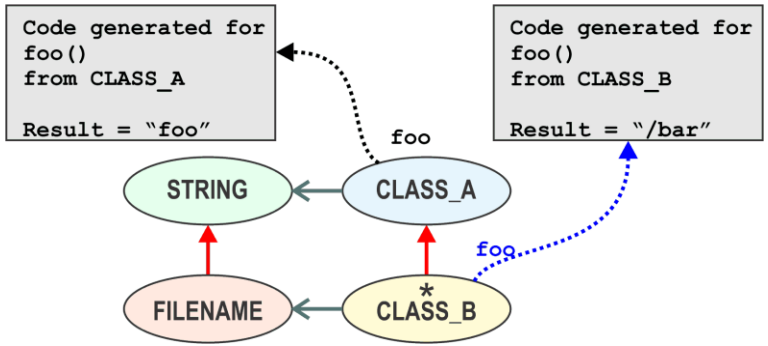


Adapting Inherited Features - Redefine

- Redefine can change the implementation of an inherited feature
- Or, change its signature (per covariance), Or both

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  redefine
   foo
  end
feature
 foo: FILENAME
  do
   Result := "/bar"
  end
end
```

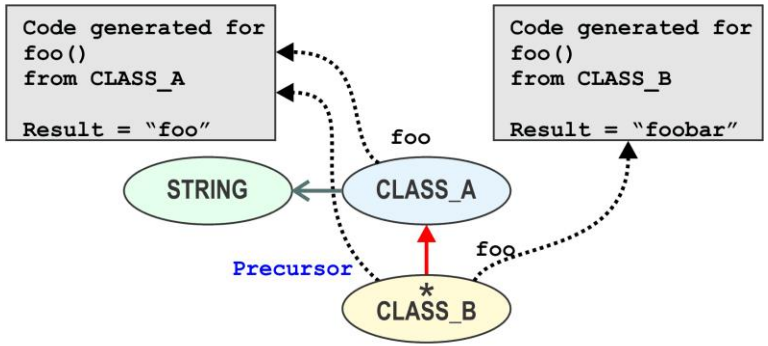


Adapting Inherited Features - Redefine

- Redefine can change the implementation of an inherited feature
- Or, change its signature (per covariance), Or both

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_B
inherit
 CLASS_A
  redefine
   foo
  end
feature
 foo: STRING
  do
   Result := Precursor +
             "bar"
  end
end
```



Precursor enables use of the original (unchanged) implementation

Adapting Inherited Features - Select

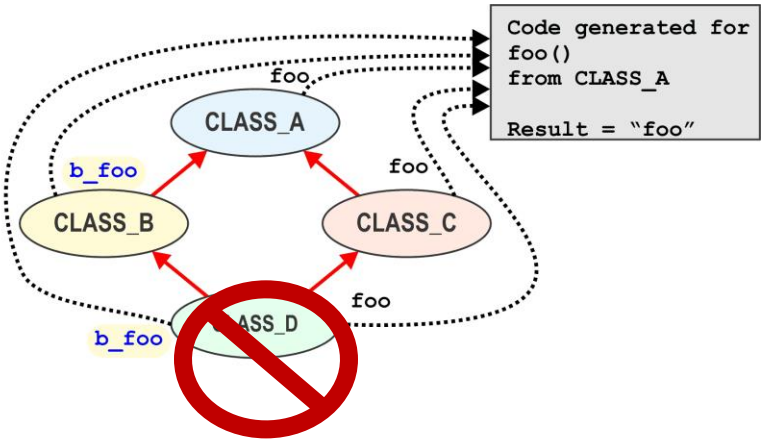
- Select is used to select the name by which an inherited feature is called
  - Rarely needed, but indispensable when it is
- Needed when the same feature from a repeatedly inherited ancestor has been renamed along the way, but not redefined

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_C
inherit
 CLASS_A
end
```

```
class CLASS_B
inherit
 CLASS_A
  rename
   foo as b_foo
  end
end
```

```
class CLASS_D
inherit
 CLASS_B
 CLASS_C
end
```



Adapting Inherited Features - Select

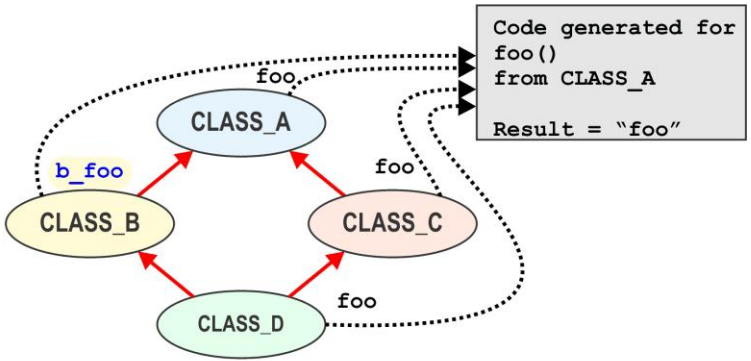
- Select is used to select the name by which an inherited feature is called
  - Rarely needed, but indispensable when it is
- Needed when the same feature from a repeated inherited ancestor has been renamed, but not redefined
- Select the name you want to use in that class, its descendants and clients

```
class CLASS_A
feature
 foo: STRING
  do
   Result := "foo"
  end
end
```

```
class CLASS_C
inherit
 CLASS_A
end
```

```
class CLASS_B
inherit
 CLASS_A
  rename
   foo as b_foo
  end
end
```

```
class CLASS_D
inherit
 CLASS_B
 CLASS_C
  select
   foo
  end
end
```



Wrapping Up

- Eiffel has comprehensive support for class-based inheritance
  - Single Inheritance, Multiple Inheritance and Repeated Inheritance
- Mechanisms for adaptation and conflict resolution
  - Rename, export, undefine, redefine, select

For more information, documentation, examples and other resources, please visit www.eiffel.org

Thank You

Roger F. Osmond
rfo@amalasoft.com