Introduction to Programming

Bertrand Meyer

Last changed: 19 December 2005

Lecture 14: More about inheritance

Stephanie Balzer

Agenda

- Another example of genericity and inheritance
- Assertions and inheritance
- Deferred features and classes
- Multiple inheritance
- Repeated inheritance
- Project assignment
Example hierarchy

Genericity + Inheritance 2: Polymorphic data structure

class LIST [G]
feature
  ...  
  last: G is ...
  extend (x: G) is ...
end

fl: LIST [FIGURE]
l: RECTANGLE
s: SQUARE
r: TRIANGLE
p: POLYGON
...
fl.extend (p); fl.extend (l); fl.extend (s); fl.extend (r)
fl.last.display

Forcing a type: the problem

fl.store ("FILE_NAME")
...
-- Two years later:
  fl := retrieved ("FILE_NAME")
  x := fl.last -- [1]
  print (x.diagonal) -- [2]

But:
- If x is declared of type RECTANGLE, [1] is invalid.
- If x is declared of type FIGURE, [2] is invalid.
The solution: Assignment attempt

```plaintext
f: FIGURE
r: RECTANGLE
...
fl.retrieve ("FILE_NAME")
f := fl.last
r ?= f
if r /= Void then
  print (r.diagonal)
else
  print ("Too bad.")
end
```

Assignment attempt

```plaintext
x ?= y
with
x : A

- If y is attached to an object whose type conforms to A, perform normal reference assignment.
- Otherwise, make x void.
```

Inheritance and assertions

```
Correct call:
if a1.a then
  a1.r (...)
else
  ...
end
```
```
```
**Assertion redeclaration rule**

- Redefined version may **not** have **require** or **ensure**.
- May have nothing (assertions kept by default), or

```
require else new_pre
ensure then new_post
```

- Resulting assertions are:
  - original_precondition or new_pre
  - original_postcondition and new_post

---

**Invariant accumulation**

- Every class inherits all the invariant clauses of its parents.
- These clauses are conceptually "and"-ed.

---

**An inheritance hierarchy**

- Deferred class: STACK
- Effective classes: LINKED_STACK, ARRAYED_STACK etc.
The role of deferred classes

- Express abstract concepts independently of implementation
- Express common elements of various implementations

Terminology: Effective = non-deferred (i.e. fully implemented)

A deferred feature

In e.g. LIST:

```plaintext
forth is require not after deferred ensure index = old index + 1 end
```

Mixing deferred and effective features

In the same class

```plaintext
(macro (x: G) is
  -- Move to first position after current
  -- where x appears, or after if none.
  do from until after or else item = x loop
     end
end

"Programs with holes"
```
"Don’t call us, we’ll call you!"

- A powerful form of reuse:
  - The reusable element defines a general scheme
  - Specific cases fill in the holes in that scheme
- Combine reuse with adaptation

Applications of deferred classes

- Analysis and design, top-down
- Taxonomy
- Capturing common behaviors

Deferred classes in EiffelBase
Java and .NET solution

- Single inheritance only for classes
- Multiple inheritance from interfaces

An interface is like a fully deferred class, with no implementations (do clauses), no attributes (and also no contracts)

Combining abstractions

- Given the classes
  - TRAIN_CAR, RESTAURANT
- how would you implement a DINER?

Examples of multiple inheritance

- Combining separate abstractions:
  - Restaurant, train car
  - Calculator, watch
  - Plane, asset
  - Home, vehicle
Composite figures through multiple inheritance

A composite figure as a list

Composite figures

```plaintext
class COMPOSITE FIGURE inherit
    FIGURE
    redefine display, move, rotate, ...
end

LIST [FIGURE]

feature
    display is
        do
            -- Display each constituent figure in turn.
            from start until after loop
                item.display forth
        end
    end

... Similarly for move, rotate etc. ...
end
```
Complex figures

- A simpler form of procedures display, move etc. can be obtained through the use of iterators.
- We'll learn to use agents for that purpose.

Multiple inheritance: Combining abstractions

```
defered class COMPARABLE [G] feature
  infix "<" (other: COMPARABLE [G]): BOOLEAN is deferred end
```

```
defered class COMPARABLE [G] feature
  infix "<=" (other: COMPARABLE [G]): BOOLEAN is
  do
    Result := Current < other or equal (Current, other)
  end
```

```
defered class COMPARABLE [G] feature
  infix ">=" (other: COMPARABLE [G]) is ...
  infix "<>" (other: COMPARABLE [G]) is ...
  ...
end
```
**Lessons from this example**

- We need the full spectrum from fully abstract (fully deferred) to fully implemented classes
- Multiple inheritance is there to help us combine abstractions

---

**Multiple inheritance: Name clashes**

![Diagram of multiple inheritance]

---

**Resolving name clashes**

![Diagram of resolving name clashes]
### Results of renaming

- \( a1: A \)
- \( b1: B \)
- \( c1: C \)

...  
- \( c1.fog \)
- \( c1.zoo \)
- \( a1.foo \)
- \( b1.foo \)

Invalid:  
- \( a1.fog, a1.zoo, b1.zoo, b1.fog, c1.foo \)

### Feature merging

- \( r^* \)

### Acceptable name clashes

If inherited features have all the same names, there is no harmful name clash if:

- They all have compatible signatures
- At most one of them is effective

**Semantics of such a case:**

- Merge all features into one
- If there is an effective feature, it imposes its implementation
Feature merging: with different names

class D
  inherit A
    rename g as f
    end
  B
  C
    rename h as f
    end
  feature
    ...
  end

Feature merging: effective features
### 1. Undefining a feature

```cpp
defered class T
inherit S
  undefine v
end feature
...
end
```

### 2. Feature merging: effective features

```cpp
class D inherit A
  rename g as f
  undefine f
end feature
B
C
rename h as f
undefine f
end feature
...
```
A special case of multiple inheritance

- Allow a class to have two or more parents.
- Examples that come to mind: ASSISTANT inherits from TEACHER and STUDENT.
- This is in fact a case of repeated inheritance

Indirect and direct repeated inheritance

Multiple is also repeated inheritance

- A typical case:
Repeated inheritance

- Assume class TAXPAYER with attributes
  
  age: INTEGER  
  address: STRING  
  bank_account: ACCOUNT  
  tax_id: INTEGER

- and routines such as

  pass_birthday is
  
  do
  
  age := age + 1
  
  end

  pay_taxes is ...

  deposit_to_account (sum: INTEGER) is ...

Repeated inheritance

- Heirs may include SWISS_TAXPAYER and US_TAXPAYER.

Repeated inheritance

- The two above classes may in turn have a common heir: SWISS_US_TAXPAYER.
Repeated inheritance issues

- What happens with features inherited twice from the common ancestor TAXPAYER, such as address, age, tax_id, pass_birthday?

Sharing and replication

- Features such as age and birthday, not renamed along any of the inheritance paths, will be shared.
- Features such as tax_id, inherited under different names, will be replicated.

The inheritance clause

```java
inhibit
  SWISS_TAXPAYER
  rename
    address as swiss_address,
    tax_id as swiss_tax_id,
    pay_taxes as pay_swiss_taxes,
    bank_account as swiss_bank_account,
  end
  ...

  US_TAXPAYER
  rename
    address as us_address,
    tax_id as us_tax_id,
    pay_taxes as pay_us_taxes,
    bank_account as us_bank_account,
  end
  ...
```
The need for select

- Assume there is a redefinition somewhere along the way:

```
TAXPAYER
  
  address
  
US_TAXPAYER
    address
    
SWISS_TAXPAYER
      address
      
      
TAXPAYER
```

The need for select

- A potential ambiguity arises because of polymorphism and dynamic binding:

```plaintext
t: TAXPAYER
su: SWISS_US_TAXPAYER

...

t := su
print (t.address)
```

Removing the ambiguity

```plaintext
class SWISS_US_TAXPAYER
  inherits SWISS_TAXPAYER
  
  address as swiss_address,
  tax_id as swiss_tax_id,
  pay_taxes as pay_taxes,
  bank_account as swiss_bank_account,
  deposit_to_account as deposit_to_swiss_account,
  
  select
    swiss_address,
    swiss_tax_id,
    pay_taxes,
    swiss_bank_account,
    deposit_to_swiss_account
  end

end

US_TAXPAYER
  
  address as us_address,
  tax_id as us_tax_id,
  
end
```
When is a name clash acceptable?

- (Between n features of a class, all with the same name, immediate or inherited.)
  - They must all have compatible signatures.
  - If more than one is effective, they must all come from a common ancestor feature under repeated inheritance.

Another application of renaming

- Provide locally better adapted terminology.
- Example: child (TREE); subwindow (WINDOW).

Project Presentation

Ilinca Ciupa
Organization

- Team of 2 people from the same exercise group
- Team members should have similar programming experience

Content

- Extension to Traffic/FlatHunt
- You choose what this extension is
- Examples:
  - Extend the 3D model of Traffic
  - Add some graph algorithms
  - Add information about events happening in the city (e.g. movies playing in cinemas)

Alternative

- Extension to EiffelMedia
- Examples:
  - Program new themes for the widget toolkit
  - Create a screenshot factory
  - More on EiffelMedia in tomorrow’s lecture
Tasks

- Initial description of the idea
- Project description
- Analysis and design
- Implementation and documentation

Initial description of the idea

- A few sentences about what you want to do in the project
- Deadline: 23 December 2005

Project description

- A detailed description of the requirements
- A clear statement of how the work will be divided between the 2 group members
- Deadline: 10 January 2006
Analysis and design

- Write a short report describing your design decisions and the overall architecture of your system
- Should include a BON diagram of the class structure
- Deadline: 17 January 2005

Implementation and documentation

- Implement and test your application
- Submit:
  - Your code
    - Only .e files and project ace file
    - Only relative paths in the ace file
  - Documentation: developer guide
    - Can reuse parts of the design document
- Deadline: 6 February 2006

Presentations of the project

- In the exercise sessions:
  - Either in week 13 or in week 14
  - If in week 13, you may get a chance to present it in front of everybody
- In the lecture:
  - In the last lecture of the semester (Tuesday, 7 February 2006)
  - Object oriental bazaar
Submission procedure

- Everything goes on the wiki
- Initial idea – as inline text on the page of your project
- Project description, design document, and developer guide – as uploaded PDF files
- Source code – as uploaded .zip file

Assessment criteria

- Design
  - Extensibility
  - Ease of use
- Functionality
  - Does the implementation satisfy the specification
- Quality of contracts
  - Preconditions
  - Postconditions
  - Class invariants
  - Loop invariants and variants

Assessment criteria (continued)

- Documentation
  - Project description
  - Design document
  - Developer guide
- Quality of code
  - Style guidelines
  - Quality of code
- Effort devoted to the project
End of lecture 14