Programming in the large

Bertrand Meyer
Lecture 10: More on inheritance
Agenda for today

- Constrained genericity
- Creating with a specified type
- Once routines
- Multiple inheritance
Agenda for today

- Constrained genericity
  - Creating with a specified type
  - Once routines
  - Multiple inheritance
Adding two vectors

\[ u + v = w \]
Constrained genericity

```plaintext
class VECTOR [G]
feature
    infix "+" (other: VECTOR [G]): VECTOR [G] is
        -- Sum of current vector and other
        require
            lower = other.lower
            upper = other.upper
        local
            a, b, c: G
        do
            ... See next ...
        end
    ... Other features ...
end
```

Chair of Software Engineering
Constrained genericity

- The body of infix "+":

```plaintext
create Result.make (lower, upper)
from
  i := lower
until
  i > upper
loop
  a := item (i)
  b := other.item (i)
  c := a + b  -- Requires a "+" operation on G!
  Result.put (c, i)
  i := i + 1
end
```
Adding two vectors

\[ u + v = w \]
Constrained genericity: The solution

- Declare class `VECTOR` as

  ```
  class VECTOR [G -> NUMERIC]
  feature
      ... The rest as before ...
  end
  ```

- Class `NUMERIC` (from the Kernel Library) provides features `infix "+"`, `infix "*"` and so on.
Improving the solution

- Make \texttt{VECTOR} itself a descendant of \texttt{NUMERIC}, effecting the corresponding features:

  \begin{verbatim}
  class VECTOR [G -> NUMERIC]
  inherit NUMERIC
  feature
    ... The rest as before, including infix "+"...
  end
  \end{verbatim}

- Then it is possible to define e.g.

  \texttt{v: VECTOR [VECTOR [VECTOR [INTEGER]]]}
Agenda for today

- Constrained genericity
- Creating with a specified type
- Once routines
- Multiple inheritance
Creating with a specified type

- To avoid this:

  \[
  a1 : A \\
  b1 : B \\
  ...
  \]

  create \( b1 . make \) (...)  
  \[ a1 := b1 \]

- Simply use

  \[
  a1 : A \\
  ...
  \]

  create \( \{ B \} \) \( a1 . make \) (...)  

(See factory pattern)
Agenda for today

- Constrained genericity
- Creating with a specified type
- Once routines
- Multiple inheritance
Once routines

- If instead of
  
  \[
  r \text{ is } \\
  \text{do} \\
  \text{end} \\
  \]

  you write
  
  \[
  r \text{ is } \\
  \text{once} \\
  \text{do} \\
  \text{end} \\
  \]

  then *Instructions* will be executed only for the first call by any client during execution. Subsequent calls return immediately.

- In the case of a function, subsequent calls return the result computed by the first call.
class SHARED_OBJECTS
feature
  error_window: WINDOW is
    once
      create Result.make (...)
    end
  exit_button: BUTTON is
    once
      create Result.make (...)
    end
end

class MY_APPLICATION_CLASS
inherit
  SHARED_OBJECTS
feature
  r is
    do
      error_window.put (my_error_message)
    end
end
Agenda for today

- Constrained genericity
- Creating with a specified type
- Once routines
- **Multiple inheritance**
Multiple inheritance

- Allow a class to have two or more parents.
- Examples that come to mind: ASSISTANT inherits from TEACHER and STUDENT.
Example: Teaching assistant

- This is in fact a case of repeated inheritance:

```
  UNIVERSITY_MEMBER
     id

  TEACHER
     ??

  STUDENT
     ??

  ASSISTANT
     ???
```

Other examples of multiple inheritance

- Combining separate abstractions:
  - Restaurant, train car
  - Calculator, watch
  - Plane, asset
Multiple inheritance: Combining abstractions

- COMPARABLE
- NUMERIC
- INTEGER
- REAL
- STRING
- DOUBLE
- COMPLEX
Multiple inheritance: Composite figures

Simple figures

A composite figure
Defining the notion of composite figure

- FIGURE
  - display
  - hide
  - rotate
  - move

- LIST
  - put
  - remove

- COMPOSITE FIGURE

Red arrows indicate the relationships between the concepts.
Composite figures through multiple inheritance

- **FIGURE**
  - **OPEN FIGURE**
    - **SEGMENT**
    - **POLYLINE**
  - **CLOSED FIGURE**
    - **POLYGON**
      - **TRIANGLE**
      - **RECTANGLE**
    - **ELLIPSE**
      - **CIRCLE**
    - **SQUARE**

- **COMPOSITE FIGURE**
  - **LIST [FIGURE]**

- Perimeter operations:
  - `perimeter+`
  - `perimeter*`
  - `perimeter++`
A composite figure as a list

```
start

item

forth

after
```
class

COMPOSITE_FIGURE

inherit

FIGURE

redefine display, move, rotate, ... end

LIST [FIGURE]

feature

display is

-- Display each constituent figure in turn.

do

from start until after loop

item.display

forth

end

end

... Similarly for move, rotate etc. ...
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: Name clashes

Diagram:

- Class A
- Class B
- Class C

foo

foo

foo
Resolving name clashes

A

foo

rename foo as fog

B

foo

rename foo as zoo

C
Resolving name clashes

class C
inherit A

rename foo as fog
end

rename foo as zoo
end

feature ...

Chair of Software Engineering
Results of renaming

\[\begin{align*}
a1 & : A \\
b1 & : B \\
c1 & : C \\
... \\
c1.fog \\
c1.zoo \\
a1.foo \\
b1.foo
\end{align*}\]

Invalid:
\[a1.fog, a1.zoo, b1.zoo, b1.fog, c1.foo\]
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)`.
End of lecture 10