Programming in the large

Bertrand Meyer
Lecture 9: Inheritance
Agenda for today

- Inheritance
  - Example
  - Polymorphism and dynamic binding
- Genericity
  - Assignment attempt
Agenda for today

- Inheritance
  - Example
- Polymorphism and dynamic binding
- Genericity
  - Assignment attempt
Example: Inheritance hierarchy

- **FIGURE**
  - extent*
  - center*
  - rotate *
  - display*
- **CLOSED_ FIGURE**
  - perimeter*
- **POLYGON**
  - perimeter+
  - diagonal
  - side1
  - side2
- **RECTANGLE**
  - perimeter++
- **TRIANGLE**
- **SEGMENT**
- **POLYLINE**
- **SQUARE**
- **ELLIPSE**
  - perimeter+
- **CIRCLE**
  - perimeter++

* deferred
+ effective
++ redefined
Example: POLYGON

```plaintext
class POLYGON
create make
feature
  vertices: ARRAY [POINT]
  vertices_count: INTEGER

  perimeter: REAL is
    -- Perimeter length
    do
      from ... until ... loop
        Result := Result + (vertices @ i) . distance (vertices @ (i + 1))
      end
    end

  invariant
    vertices_count >= 3
    vertices_count = vertices.count
end
```

---

Chair of Software Engineering
Example: RECTANGLE by redefining POLYGON

class 
  RECTANGLE

inhibit 
  POLYGON
  redefine perimeter end

create 
  make

feature 
  diagonal, side1, side2: REAL
  perimeter: REAL is
    -- Perimeter length
    do 
      Result := 2 * (side1 + side2)
    end
  invariant 
    vertices_count = 4

end
Polymorphism and dynamic binding

- Assume:
  - \( p: POLYGON; r: RECTANGLE; t: TRIANGLE; \)
  - \( x: REAL \)

- Permitted:
  - \( x := p.perimeter \)
  - \( x := r.perimeter \)
  - \( x := r.diagonal \)
  - \( p := r \)

- NOT permitted:
  - \( x := p.diagonal \) (even just after \( p := r \) !)
  - \( r := p \)
Polymorphism and dynamic binding

- What is the effect of the following (assuming `some_test` is true)?

```plaintext
if some_test then
    p := r
else
    p := t
end
x := p.perimeter
```

- **Redefinition**: A class may change an inherited feature, as with `RECTANGLE` redefining perimeter of `POLYGON`.

- **Polymorphism**: `p` may have different forms at run-time.

- **Dynamic binding**: Effect of `p.perimeter` depends on run-time form of `p`. 
Dynamic binding: Using non-O-O techniques

\[\text{display (} f: \text{FIGURE}) \text{ is} \]
\[
\begin{align*}
\text{do} \\
\text{if } \text{“} f \text{ is a CIRCLE” } \text{ then} \\
\quad \ldots \\
\text{elseif } \text{“} f \text{ is a POLYGON” } \text{ then} \\
\quad \ldots \\
\text{end} \\
\end{align*}
\]

and similarly for all other routines!

Tedious; must be changed whenever there’s a new figure type.
Dynamic binding: In action

With:

```
figure_list: LIST [FIGURE]
c: CIRCLE
p: POLYGON
f: FIGURE
```

Initialize:

```
figure_list.extend (c)
figure_list.extend (p)
```

and:

```
c.create
p.create
figure_list.make
```

Then just use:

```
f := figure_list.i_th (i)
f.move (...)
f.rotate (...)
f.Display
-- and so on for every
-- operation on f
```
The dangers of static binding

- For every creation procedure $cp$:
  
  $\{\text{Pre}_{cp}\} \text{ do } _{cp} \{\text{Post}_{cp} \text{ and INV}\}$

- For every exported routine $r$:
  
  $\{\text{INV and Pre}_{r}\} \text{ do } _{r} \{\text{INV and Post}_{r}\}$

- The worst possible erroneous run-time situation in object-oriented software development:
  - Producing an object that does not satisfy the invariant of its class.
The dangers of static binding (cont’d)

- \( \{\text{INV}_A\} \) \textbf{do}_{r_A} \{\text{INV}_A\} \\
- \( \{\text{INV}_B\} \) \textbf{do}_{r_B} \{\text{INV}_B\} \\

Consider a call of the form \( a_1.r \) where \( a_1 \) is polymorphic:
  - No guarantee on the effect of \( \textbf{do}_{r_A} \) on an instance of \( B \)! 

\( A \)
\( B \)
A concrete example

\[ w : \text{WINDOW} \]
\[ b : \text{BUTTON} \]

create \( b \)

\[ w := b \]

\( w . \text{display} \)
Using original version of redefined feature

class
  BUTTON
inherit
  WINDOW
redefine display end

feature
  display is do
    Precursor {WINDOW}
    display_border
display_label
  end
display_label is do ... end
display_border is do ... end
end
Use of Precursor

- Not necessarily the first feature clause.
- May have arguments.

```plaintext
class B
inherit A

redefine my_feature end

feature

my_feature (args: SOME_TYPE) is
  do
    -- Something here
    Precursor {A} (args)
    -- Something else here
  end
end
```
Agenda for today

- Inheritance
  - Example
  - Polymorphism and dynamic binding
- Genericity
  - Assignment attempt
Genericity vs. Inheritance

Abstraction

Type parameterization

LIST_OF_PEOPLE

LIST_OF_BOOKS

LIST_OF_JOURNALS

Specialization

LIST_OF_BOOKS

LIKED_LIST_OF_BOOKS
Genericity: \textit{LIST} [G]

class \textit{LIST} [G]

feature

... 

\textit{item}: G is ... 

\textit{extend}: (x: G) is ... 

end

\textit{figure_list}: LIST [\textit{FIGURE}]

r: \textit{RECTANGLE}

s: \textit{SQUARE}

t: \textit{TRIANGLE}

p: \textit{POLYGON}

\textit{figure_list}.extend (p)

\textit{figure_list}.extend (t)

\textit{figure_list}.extend (s)

\textit{figure_list}.extend (r)

\textit{figure_list}.i_th (i).display

\textit{Chair of Software Engineering}
Example: Inheritance hierarchy

- **FIGURE**
  - * deferred
  - + effective
  - ++ redefined

- **OPEN FIGURE**
  - extent*
  - center*
  - rotate*
  - display*

- **CLOSED FIGURE**
  - perimeter*

- **SEGMENT**
  - perimeter+

- **POLYLINE**
  - perimeter+

- **POLYGON**
  - perimeter+
  - diagonal
  - side1
  - side2

- **RECTANGLE**
  - perimeter++

- **SQUARE**
  - perimeter++

- **TRIANGLE**

- **ELLIPSE**
  - perimeter++

- **CIRCLE**
  - perimeter++
figure_list.store ("FILE_NAME")
...

-- Two years later:
figure_list := retrieved ("FILE_NAME")
x := figure_list.i_th (i) -- [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

$x \neq y$

with

$x: A$

- If $y$ is attached to an object whose type conforms to $A$, perform normal reference assignment.
- Otherwise, make $x$ void.
Forcing a type: The solution (using an assignment attempt)

\[
\begin{align*}
f & : FIGURE \\
r & : RECTANGLE \\
\ldots \\
figure\_list & := retrieved \("FILE\_NAME"\) \\
f & := figure\_list.i\_th(i) \\
r & ?= f \\
if \ r \neq Void then \\
\quad print\ (r.diagonal) \\
else \\
\quad print\ ("Too bad."") \\
end
\end{align*}
\]
End of lecture 9