Object-Oriented Software Construction

Bertrand Meyer

Lecture 7: 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

- * deferred
- + effective
- ++ redefined

- extent*
- center*
- rotate *
- display*

- perimeter*

- OPEN* FIGURE
- SEGMENT
- POLYLINE

- CLOSED* FIGURE
- perimeter*

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

- RECTANGLE

- SQUARE
- perimeter++

- ELLIPSE
- perimeter+

- CIRCLE
- perimeter++
Example: POLYGON

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

OOSC - Summer Semester 2005
Example: RECTANGLE by redefining POLYGON

class
  RECTANGLE
inherit
  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:
  \[ p: POLYGON; \quad r: RECTANGLE; \quad t: TRIANGLE; \]
  \[ x: REAL \]

- Permitted:
  \[ x := p\text{.perimeter} \]
  \[ x := r\text{.perimeter} \]
  \[ x := r\text{.diagonal} \]
  \[ p := r \]

- NOT permitted:
  \[ x := p\text{.diagonal} \quad (\text{even just after} \quad p := r \quad !) \]
  \[ r := p \]
Polymorphism and dynamic binding

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

  ```
  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

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

and similarly for all other routines!

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

With:

\[
\begin{align*}
\text{figure_list: LIST} & \quad [\text{FIGURE}] \\
c & \quad \text{CIRCLE} \\
p & \quad \text{POLYGON} \\
f & \quad \text{FIGURE}
\end{align*}
\]

Initialize:

\[
\begin{align*}
\text{figure_list.extend} & \quad (c) \\
\text{figure_list.extend} & \quad (p)
\end{align*}
\]

and:

\[
\begin{align*}
\text{create c.make} \\
\text{create p.make} \\
\text{create figure_list.make}
\end{align*}
\]

Then just use:

\[
\begin{align*}
f & \quad := \text{figure_list.i_th} \quad (i) \\
f & \quad \text{move} \quad (\ldots) \\
f & \quad \text{rotate} \quad (\ldots) \\
f & \quad \text{display} \\
& \quad \text{-- and so on for every} \\
& \quad \text{-- operation on f}
\end{align*}
\]
The dangers of static binding

- For every creation procedure \( cp \):
  \[
  \{ \text{Pre}_cp \} \, \text{do}_cp \, \{ \text{Post}_cp \, \text{and} \, \text{INV} \}
  \]

- 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

- \{\text{INV}_A\} \ \text{do}_{r_A} \ \{\text{INV}_A\}

- \{\text{INV}_B\} \ \text{do}_{r_B} \ \{\text{INV}_B\}

- Consider a call of the form \text{a1.r} where \text{a1} is polymorphic:
  - No guarantee on the effect of \text{do}_{r_A} on an instance of \text{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
May have arguments.

class
  B
inhibit
  A

redefine my_feature end

feature

my_feature (args: SOME_TYPE) is
  do
    -- Something here
    Precursor {A} (args)
    -- Something else here
  end
end
Genericity vs. Inheritance

Abstraction

Set of Books

Type parameterization

List of People

List of Books

Type parameterization

Liked List of Books

Specialization

List of Journals
Genericity: \textit{LIST} \([G]\)

class \textit{LIST} \([G]\)

feature

...  
\texttt{last: G is ...}
\texttt{extend (x: G) is ...}

end

\texttt{figure_list: LIST [FIGURE]}
\texttt{r: RECTANGLE}
\texttt{s: SQUARE}
\texttt{t: TRIANGLE}
\texttt{p: POLYGON}
\texttt{figure_list.extend (p)}
\texttt{figure_list.extend (t)}
\texttt{figure_list.extend (s)}
\texttt{figure_list.extend (r)}
\texttt{figure_list.last.display}
Example: Inheritance hierarchy

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

* deferred
+ effective
++ redefined
Genericity: Forcing a type - the problem

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

But:
- If \( x \) is declared of type \( \text{RECTANGLE} \), [1] is invalid.
- If \( x \) is declared of type \( \text{FIGURE} \), [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)

\[
f: \text{FIGURE} \\
r: \text{RECTANGLE} \\
\ldots \\
figure\_list := \text{retrieved} ("FILE\_NAME") \\
f := figure\_list\_last \\
r \neq f \\
\text{if } r \neq \text{Void then} \\
\quad \text{print} (r\_\text{diagonal}) \\
\text{else} \\
\quad \text{print} ("Too bad.") \\
\text{end}
\]
End of lecture 7