Lecture 9: Inheritance

Agenda for today

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

- Inheritance
  - Example
- Polymorphism and dynamic binding
- Generality
  - Assignment attempt

Example: Inheritance hierarchy

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
Example: RECTANGLE by redefining POLYGON

```java
class RECTANGLE
inherit POLYGON
redefine perimeter end
create
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 \texttt{some\_test} is true)?
  ```java
  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 \texttt{perimeter} of POLYGON.
- Polymorphism: \( p \) may have different forms at run-time.
- Dynamic binding: Effect of \texttt{p.perimeter} depends on run-time form of \( p \).
Dynamic binding: Using non-O-O techniques

```plaintext
display (f: FIGURE) is
do
  if "f is a CIRCLE" then
  ...
  elseif "f is a POLYGON" then
  ...
end

and similarly for all other routines!

Tedious; must be changed whenever there's a new figure type.
```

Dynamic binding: In action

With:
```plaintext
figure_list: LIST (FIGURE)
c: CIRCLE
p: POLYGON
f: FIGURE
```
and:
```plaintext
create c.make
create p.make
create figure_list.make
```

Initialize:
```plaintext
figure_list.extend (c)
figure_list.extend (p)
```
Then just use:
```plaintext
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:
  ```plaintext
  \{Pre\} do, \{Post, and INV\}
  ```

- For every exported routine r:
  ```plaintext
  \{INV and Pre\} do, \{INV and Post\}
  ```

- 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)

- \( \{ \text{INV}_n \} \text{ do}_A \{ \text{INV}_n \} \)
- \( \{ \text{INV}_n \} \text{ do}_B \{ \text{INV}_n \} \)
- Consider a call of the form \( a1.r \) where \( a1 \) is polymorphic:
  - No guarantee on the effect of \( \text{do}_A \) on an instance of \( B \)!

A concrete example

\[ \begin{align*}
  w & : \text{WINDOW} \\
  b & : \text{BUTTON} \\
  \text{create} & b \\
  w & := b \\
  w & . \text{display}
\end{align*} \]

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

Graph showing abstraction and specialization with type parameterization.
Genericity: LIST [G]

```plaintext
class LIST [G]
feature ...
  item: G is ...
  extend (x: G) is ...
end

figure_list: LIST [FIGURE]
r: RECTANGLE
s: SQUARE
i: TRIANGLE
p: POLYGON
figure_list.extend (p)
figure_list.extend (i)
figure_list.extend (s)
figure_list_i_th (i).display
```

Example: Inheritance hierarchy

```
deferred + effective ++ redefined
extent*
center*
rotate *
display*
```

Genericity: Forcing a type - the problem

```plaintext
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

\[ x \, ?= \, y \]

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)

```plaintext
f: FIGURE
r: RECTANGLE
...
figure_list := retrieved ("FILE_NAME")
f := figure_list.i_th(i)
r ?= f
if r /= Void then
    print (r.diagonal)
ext else
    print ("Too bad.")
end
```

End of lecture 9