Lecture 9: Inheritance

Agenda for today

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

Example: Inheritance hierarchy

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
invariant
  vertices_count >= 3
  vertices_count = vertices.count
end
```
Example: RECTANGLE by redefining POLYGON

```haskell
class RECTANGLE
inherit POLYGON
redefine perimeter
create
make
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

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

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

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
t: FIGURE`

Initialize:

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

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`:
  `{Pre,cp} do. {Post,w and INV}
- For every exported routine `r`:
  `{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)

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

A concrete example

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

\[\begin{array}{c}
\text{create } b \\
\text{w := b} \\
\text{w.display}
\end{array}\]

Using original version of redefined feature

```plaintext
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
- Genericy
  - Assignment attempt

Genericity vs. Inheritance
Programming in the large - Lecture 9

Genericity: \textit{LIST} [G]

\begin{verbatim}
class LIST [G] feature 
  item: \text{G} is ... 
  extend (x: \text{G}) is ...
end

figure_list: LIST [FIGURE] r: RECTANGLE x: SQUARE t: TRIANGLE p: POLYGON
figure_list.extend (p)
figure_list.extend (t)
figure_list.extend (s)
figure_list.i_th (i).display
\end{verbatim}

Example: Inheritance hierarchy

\begin{verbatim}
Genericity: Forcing a type - the problem

figure_list.store ("FILE_NAME") ...
-- Two years later:
  \text{figure\_list} := retrieved ("FILE\_NAME")
  \text{x} := figure_list.i_th (i) -- [1]
  print (x.diagonal) -- [2]

But:
- If \text{x} is declared of type \text{RECTANGLE}, [1] is invalid.
- If \text{x} is declared of type \text{FIGURE}, [2] is invalid.
\end{verbatim}

The solution: Assignment attempt

\begin{verbatim}
\text{x} \equiv \text{y} \with \text{x}: \text{A}
- If \text{y} is attached to an object whose type conforms to \text{A}, perform normal reference assignment.
- Otherwise, make \text{x} void.
\end{verbatim}

Forcing a type: The solution (using an assignment attempt)

\begin{verbatim}
f: FIGURE
r: RECTANGLE ...
  \text{figure\_list} := retrieved ("FILE\_NAME")
  \text{f} := figure_list.i_th (i)
  \text{r} \equiv \text{f}
  \text{if} \text{r} /= \text{Void} \text{then}
    \text{print} (r.diagonal)
  \text{else}
    \text{print} ("Too bad.")
  \text{end}
\end{verbatim}

End of lecture 9