Object-Oriented Software Construction

Bertrand Meyer
Lecture 8: Inheritance
Typing vs. binding

▪ What do we know about the feature to be called?

▪ Static typing:
  At least one

▪ Dynamic binding:
  The right one

▪ Example:
  my_aircraft.lower_landing_gear
Example hierarchy

AIRCRAFT
  \* deferred
  + effected
  ++ redefined

PLANE

BOEING
  \* deferred

AIRBUS
  \* deferred

COPTER

lower_landing_gear

B_737

B_747

A_320

B_747_400

lower_landing_gear

lower_landing_gear+

lower_landing_gear++
(After Barry W. Boehm)
Multiple inheritance

- A class may have two or more parents.
- What not to use as an elementary example: \textit{TEACHING\_ASSISTANT} inherits from \textit{TEACHER} and \textit{STUDENT}.
The teaching assistant example

- This is in fact a case of repeated inheritance:
Common 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

* Denotes multiplicity.
Multiple inheritance: Nested windows

- "Graphical" features: `height`, `width`, `change_height`, `change_width`, `xpos`, `ypos`, `move`...
- "Hierarchical" features: `superwindow`, `subwindows`, `change_subwindow`, `add_subwindow`...

```plaintext
class WINDOW inherit
  RECTANGLE
  TREE [WINDOW]
feature
  ...
end
```
Multiple inheritance: Composite figures

Simple figures

A composite figure
Defining the notion of composite figure

* \( \text{FIGURE} \) \n  \- display*
  \- hide
  \- rotate
  \- move
  \- …

\( \text{LIST} [\text{FIGURE}] \) \n  \- count
  \- put
  \- remove
  \- …

\( \text{COMPOSITE FIGURE} \) \n  \- display^
Defining the notion of composite figure through multiple inheritance

* FIGURE

OPEN FIGURE

SEGMENT

POLYLINE

... perimeters

POLYGON

CLOSED FIGURE

perimeter*

TRAVERSE

RECTANGLE

perimeter+

TRIANGLE

ELLIPSE

CIRCLE

perimeter++

... perimeters

COMPOSITE FIGURE

LIST [FIGURE]

perimeter

perimeter+

perimeter++

... perimeters

perimeter+
A composite figure as a list
class COMPOSITE FIGURE inherit

FIGURE
redefine display, move, rotate, ... end

LIST [FIGURE]

feature

display is

do

from start
until after
loop
item.display
forth

end

... Similarly for move, rotate etc. ...

end
Complex figures

- Note: a simpler form of procedures *display*, *move* etc. can be obtained through the use of iterators.

- Exercise: Use agents for that purpose.
Name clashes under multiple inheritance

Diagram:

- Node A
- Node B
- Node C
- Arrows from A to C, from B to C
- Nodes labeled with "foo"
Resolving name clashes

rename foo as fog

rename foo as zoo
Resolving name clashes (cont’d)

class $C$ inherit

$A$

    rename $foo$ as $fog$
    end

$B$

    rename $foo$ as $zoo$
    end

feature

...
Results of renaming

\[ a1: A \]
\[ b1: B \]
\[ c1: C \]

... 
\[ c1.fog \]
\[ c1.zoo \]
\[ a1.foo \]
\[ b1.foo \]

Invalid:
\[ a1.fog, a1.zoo, b1.zoo, b1.fog, c1.foo \]
Another application of renaming

- Provide locally better adapted terminology.

- Example: *child (TREE); subwindow (WINDOW).*

feature
...
end


feature
...
end
The need for deferred classes

- In the scheme seen earlier:

  \[ f: FIGURE; \ c: CIRCLE; \ p: POLYGON \]
  ...
  \texttt{create \ c.make (...)}; \texttt{create \ p.make (...)}
  ...
  \texttt{if}\ \ldots\ \texttt{then}
  \hspace{1cm} f := c
  \texttt{else}
  \hspace{1cm} f := p
  \texttt{end}
  ...
  \texttt{f.move (...)}; \texttt{f.rotate (...)}; \texttt{f.display (...)}; \ldots

- How do we ensure that a call such as \texttt{f.move (...)} is valid even though there is no way to implement a general-purpose feature \texttt{move} for class \texttt{FIGURE}?
Deferred classes

defered class \textit{FIGURE} feature

\begin{align*}
\text{move} \ (v: \textsc{vector}) & \quad \text{is} \\
& \quad \text{deferred} \\
& \quad \text{end} \\
\text{rotate} \ (a: \textsc{angle}; \ p: \textsc{point}) & \quad \text{is} \\
& \quad \text{deferred} \\
& \quad \text{end} \\
\end{align*}

... \textit{display}, \textit{hide}, ...

end

Not permitted:

\textit{create} \ f ...
Deferred classes and features

- A feature is either deferred or effective.
- To effect a inherited feature (deferred in the parent) is to make it effective. No need for redefine clause.
- Like a feature, a class is either deferred or effective.
- A class is deferred if it has at least one deferred feature (possibly coming from an ancestor) that it does not effect. It is effective otherwise.
- A deferred class may not be instantiated.
- BUT: A deferred class may have assertions (in particular, a deferred routine may have a precondition and a postcondition, and the class may have a class invariant).
Deferred classes

- Compare with Ada-Modula 2-Java interface/body separation:
  - May contain both deferred and non-deferred elements.
  - More than one implementation is possible.
  - Formal semantic specification (assertions).
Table variants

* TABLE
  * SEQUENTIAL_TABLE
    + ARRAY_TABLE
    + LINKED_TABLE
    + FILE_TABLE
deferred class ''SEQUENTIAL_TABLE'' [G] inherit ''TABLE'' [G]

feature

has (x: G): BOOLEAN is

-- Does x appear in table?
do from start until after or else equal (item, x) loop forth end

Result := not after

end
forth is
  -- Move cursor to the next position.
require
  not after
deferred
ensure
  position = old position + 1
end

start is
  -- Move cursor to the first position.
deferred
ensure
  empty or else position = 1
end
position: INTEGER is
defered
end

... empty, found, after, ...

invariant

0 <= position

position <= size + 1

empty implies (after or before)

end
Descendant implementations

- **TABLE**
  - * has*
  - +

- **SEQUENTIAL_TABLE**
  - * has+
  - after* forth* item* start*

- **ARRAY_TABLE**
  - +
  - after* forth* item* start*

- **LINKED_TABLE**
  - +
  - after* forth* item* start*

- **FILE_TABLE**
  - +
  - after* forth* item* start*
Descendant implementations (cont’d)

- **SEQUENTIAL_TABLE**
  - after*
  - forth*
  - item*
  - start*
  - has+

- **ARRAY_TABLE**
  - after+
  - forth+
  - item+
  - start+

- **LINKED_TABLE**
  - after+
  - forth+
  - item+
  - start+

- **FILE_TABLE**
  - after+
  - forth+
  - item+
  - start+
## Implementation variants

<table>
<thead>
<tr>
<th></th>
<th>start</th>
<th>forth</th>
<th>after</th>
<th>item (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array</strong></td>
<td>i := 1</td>
<td>i := i + 1</td>
<td>i &gt; count</td>
<td>t[i]</td>
</tr>
<tr>
<td><strong>Linked list</strong></td>
<td>c := first_cell</td>
<td>c := c.right</td>
<td>c := Void</td>
<td>c.item</td>
</tr>
<tr>
<td><strong>File</strong></td>
<td>rewind</td>
<td>read</td>
<td>end_of_file</td>
<td>f↑</td>
</tr>
</tbody>
</table>
End of lecture 8