Exercise 4: Inheritance

Hand-out: 7 May 2004
Due: 14 May 2004

Master Solution

Summary: Inheritance

Principle: Describe a new class not from scratch but as extension or specialization of one existing class — or several in the case of multiple inheritance (which you will see in the next lecture).

- Redefinition: A class may change an inherited feature.
- Polymorphism: An entity may have different forms at run-time.
- Dynamic binding: Effect of a feature call $x.f$ depends on run-time form of $x$.

Here is a typical example:

```
p: POLYGON
r: RECTANGLE       -- Assume RECTANGLE inherits from POLYGON.
t: TRIANGLE         -- Assume TRIANGLE inherits from POLYGON.
x: INTEGER
...
if some_test then
  p := r
else
  p := t
end
x := p.perimeter    -- Assume class POLYGON has a feature perimeter
                    -- returning an INTEGER.
```

-- If `some_test` is true, `p.perimeter` uses the version of `perimeter` from `RECTANGLE`.
-- If `some_test` is false, `p.perimeter` uses the version of `perimeter` from `TRIANGLE`.
1. The stacks of Hanoï

(This exercise in an extract of *Object-Oriented Software Construction, 2nd edition*, by Bertrand Meyer; page 869.)

Assume a deferred class *STACK* with a procedure *put* to push an element onto the top, with a precondition involving the boolean-valued function *full*.

Now consider the famous problem of the Towers of Hanoï, where disks are stacked on piles – the towers – with the rule that a disk may only be put on a larger disk.

**To do**

Is it appropriate to define the class *HANOI_STACK*, representing such piles, as an heir to *STACK*? If so, how should the class be written? If not, can *HANOI_STACK* still make use of *STACK*? Write the class in full for the various possible solutions; discuss the pros and cons of each, state which one you prefer, and explain the rationale for your choice.

**Hint**

The boolean query *full* could also be called *extendible*; as you study the exercise you will note that the choice of name may affect the appeal of various possible solutions.

**To hand in**

Hand in the text of class *HANOI_STACK* and explanations.

**Solution**

The precondition of the *extend* feature of *STACK* is simply the Boolean flag *extendible*. This makes it impossible to specify that only certain (smaller) discs may be put onto the top of the *HANOI_STACK*, as preconditions in the inheritance clause may only be weakened in an inheritance relation.

The solution is to use a client / supplier relation to reuse the *STACK*: The *HANOI_STACK* passes its calls to an instance of class *STACK*.

```plaintext
class HANOI_STACK

create

    make_empty, make_full

feature -- Initialization

    make_empty is
        -- Create an empty stack for the towers of Hanoï.
```
do
  create \{LINKED\_STACK [INTEGER]\} stack
ensure
  is\_empty: is\_empty
end

make\_full is
  -- Create a full stack with discs from size 1 to `Maximum\_size'.
local
  i: INTEGER
do
  make\_empty from
  i := Maximum\_size
until
  i = 0
loop
  stack.put (i)
  i := i - 1
ensure
  is\_full: is\_full
all\_discs\_present: count = Maximum\_size
end

feature -- Access

  Maximum\_size: INTEGER is 10
  -- Maximum size of the discs

item: INTEGER is
  -- Size of the top disc on the stack
require
  not\_empty: count > 0
do
  Result := stack\_item
ensure
  value\_range: item >= 1 and item <= Maximum\_size
end

feature -- Measurement

count: INTEGER is
  -- Number of discs on the stack
do
  Result := stack\_count
ensure
count_not_negative: Result >= 0
count_not_larger_than_maximum: Result <= Maximum_size
end

feature -- Status report

is_empty: BOOLEAN is
-- Is the stack empty?
do
Result := stack.is_empty
ensure
definition: Result = stack.is_empty
end

is_full: BOOLEAN is
-- Is the stack full?
do
Result := (item = 1)
ensure
definition: Result = (item = 1)
end

feature -- Element change

put (a_disc: INTEGER) is
-- Put a disc of size `a_disc' on the top of the stack.
require
a_disc_not_too_small: a_disc >= 1
a_disc_not_too_large: not is_empty implies a_disc < item
do
stack.put (a_disc)
ensure
count_increased: count = old count + 1
disc_put: item = a_disc
end

feature -- Removal

remove is
-- Remove the top disc from the stack.
require
not_empty: not is_empty
do
stack.remove
ensure
count_decreased: \( \text{count} = \text{old count} - 1 \)

end

feature -- Conversion

\[
\text{linear\_representation}: \text{LINEAR [INTEGER]} \text{ is}
\]
\[
\text{-- A linear representation of the stack}
\]
do
\[
\text{Result} := \text{stack.linear\_representation}
\]
ensure
\[
\text{linear\_representation\_not\_void}: \text{Result} \neq \text{Void}
\]
end

feature \{NONE\} -- Implementation

\[
\text{stack}: \text{STACK [INTEGER]}
\]
\[
\text{-- Stack for the towers of Hanoi (to which calls are forwarded)}
\]

invariant

\[
\text{empty\_has\_no\_discs}: \text{is\_empty} = (\text{count} = 0)
\]
\[
\text{full\_has\_smallest\_disc}: \text{is\_full} = (\text{item} = 1)
\]

end

2. Where do the iterators belong?

(This exercise is an extract of \textit{Object-Oriented Software Construction, 2\textsuperscript{nd} edition}, by Bertrand Meyer; page 870.)

Would it be a good idea to have iterator features (\textit{while\_do} and the like) included in classes describing the data structures on which they iterate, such as \textit{LIST}? Consider the following points:

- The ease of applying iterations to arbitrary \textit{action} and \textit{test} routines, chosen by the application.
- Extendibility: the possibility of adding new iteration schemes to the library.
- More generally, respect of object-oriented principles, in particular the idea that operations do not exist by themselves but only in the relation to certain data abstractions.

\textbf{To hand in}

Hand in the (detailed) answer to the above question.
Solution

First, it is important to understand what an iterator is. In the book Design Patterns by Gamma et al., the Iterator pattern is described as “a way to access the elements of an aggregate object sequentially without exposing its underlying implementation”. Typically, it means equipping a class with features like start, forth, after, and item. Here the question is to know whether it is better to add these features to the traversable container itself (we call this an “internal iterator”) or to an external class, say ITERATOR (“external iterator”). There are pros and cons to both approaches, as explained below:

**Internal iterators:**
- **Advantages:**
  - Respect the principle of object technology according to which operations (here, traversal operations) do not exist without an underlying data structure.
  - Make it easy for clients to traverse a given container because it is only a call to a feature of the class. No need to create an iterator object on the container and use it afterwards.
  - No problem of consistency when the container gets changed because the iteration features are part of the class itself.
- **Disadvantages:**
  - Do not allow more than one iterator on a given container at the same time.
  - Do not allow adding new iteration schemes very easily: one needs to change the container class, which is not always desirable if many clients are already using it.

**External iterators:**
- **Advantages:**
  - Allow multiple simultaneous traversal of the same container.
  - Allow adding new iteration schemes easily: one just needs to write another ITERATOR class (or another feature in an ITERATOR class).
- **Disadvantages:**
  - Problem of consistency between the iterator and the structure it iterates over: what happens if an element is removed or added to the container? External iterators are called “robust” when the iterators are always up-to-date even when the associated container changes.
  - Iteration is not a proper ADT; it is an operation on an ADT (the one of traversable containers). Thus having iterators as external classes is not really in line with the spirit of object technology and the fact that classes should be based on ADTs.
  - A bit more complicated to use for clients: they cannot apply iteration features directly on their container objects; they need to create an iterator object with the given container and then call iteration features on that iterator.
3. About dialog

To do
Write a class \texttt{ABOUT\_DIALOG}. The dialog layout should be as follows:

![Dialog layout diagram]

The dialog title should be “My fancy about dialog”.
You can choose whatever image you want.
The message should be:

My fancy about dialog

Copyright (C) 2004 <your name here>
All rights reserved

ETH Zentrum
CH-8092 Zurich
Switzerland
Electronic mail: <your email address here>

Hint
Make your class \texttt{ABOUT\_DIALOG} a descendant of the class \texttt{EV\_DIALOG} from EiffelVision2.

To hand in
Hand in the text of class \texttt{ABOUT\_DIALOG} and send the Eiffel file “fancy_message_dialog.e” and the “.ace” file (in your project directory) to your assistant after making sure that your project compiles. (If it does not compile, please explain your problems to your assistant.)
**Solution**

Here is a possible solution. Many variants would be correct:

```plaintext
indexing

    description: "Dialog giving information about the application"

class

    ABOUT_DIALOG

inherit

    EV_DIALOG

redefine

    initialize

end

create

    default_create

feature {NONE} -- Initialization

    initialize is

        -- Initialize GUI components and build the dialog.

        local

            an_icon_pixmap: EV_PIXMAP

        do

            Precursor {EV_DIALOG}

            create about_pixmap

            create text_label.make_with_text (about_dialog_text)

            create ok_button.make_with_text_and_action ("OK", agent close_dialog)

            set_title (window_title)

            set_size (340, 360)

            create an_icon_pixmap

            an_icon_pixmap.set_with_named_file (icon_file_name)

            set_icon_pixmap (an_icon_pixmap)

            build

        end

    build is

        -- Build dialog.

        local

            hbox: EV_HORIZONTAL_BOX

```

---

**SUMMER 2004**

**NUMBER OF EXERCISES: 13**

**SOLUTION:**

Here is a possible solution. Many variants would be correct:

```plaintext
indexing

    description: "Dialog giving information about the application"

class

    ABOUT_DIALOG

inherit

    EV_DIALOG

redefine

    initialize

end

create

    default_create

feature {NONE} -- Initialization

    initialize is

        -- Initialize GUI components and build the dialog.

        local

            an_icon_pixmap: EV_PIXMAP

        do

            Precursor {EV_DIALOG}

            create about_pixmap

            create text_label.make_with_text (about_dialog_text)

            create ok_button.make_with_text_and_action ("OK", agent close_dialog)

            set_title (window_title)

            set_size (340, 360)

            create an_icon_pixmap

            an_icon_pixmap.set_with_named_file (icon_file_name)

            set_icon_pixmap (an_icon_pixmap)

            build

        end

    build is

        -- Build dialog.

        local

            hbox: EV_HORIZONTAL_BOX

```
\begin{verbatim}
vbox, vb: EV_VERTICAL_BOX
c: EV_CELL
h_separator: EV_HORIZONTAL_SEPARATOR

\textbf{do}

create white_color.make_with_8_bit_rgb (255,255,255)
create vbox
create c
c.set_background_color (white_color)
c.set_minimum_height (10)
vbox.extend (c)
vbox.disable_item_expand (c)

create hbox
create c
c.set_background_color (white_color)
c.set_minimum_width (10)
hbox.extend (c)
hbox.disable_item_expand (c)
about_pixmap.set_with_named_file (about_file_name)
hbox.extend (about_pixmap)
create c
c.set_background_color (white_color)
c.set_minimum_width (10)
hbox.extend (c)
hbox.disable_item_expand (c)

text_label.align_text_left

text_label.set_background_color (white_color)
hbox.extend (text_label)
hbox.disable_item_expand (text_label)
create c
c.set_background_color (white_color)
c.set_minimum_width (10)
hbox.extend (c)
hbox.disable_item_expand (c)
vbox.extend (hbox)

create c
c.set_background_color (white_color)
c.set_minimum_height (10)
vbox.extend (c)
vbox.disable_item_expand (c)

create h_separator
vbox.extend (h_separator)
vbox.disable_item_expand (h_separator)
\end{verbatim}
create vb
create c
c.set_minimum_height (10)
vb.extend (c)
vb.disable_item_expand (c)

create hbox
hbox.extend (create {EV_CELL})
ok_button.align_text_center
ok_button.set_minimum_width (80)
ok_button.set_minimum_height (25)
hbox.extend (ok_button)
hbox.disable_item_expand (ok_button)
create c
c.set_minimum_width (10)
hbox.extend (c)
hbox.disable_item_expand (c)
vb.extend (hbox)
vb.disable_item_expand (hbox)

create c
c.set_minimum_height (10)
vb.extend (c)
vb.disable_item_expand (c)

vbox.extend (vb)
vbox.disable_item_expand (vb)
end

feature {NONE} -- Event handling

close_dialog is
    -- Close dialog.
    do
        destroy
    end

feature {NONE} -- Implementation (GUI components)

about_pixmap: EV_PIXMAP
    -- Pixmap on the left side of the dialog

text_label: EV_LABEL
    -- Label with dialog message
Here is the corresponding dialog:
My fancy about dialog

Copyright (C) 2004 Karine Arnout
All rights reserved

ETH Zentrum
CH-8092 Zürich
Switzerland
Electronic mail: Karine.Arnout@inf.ethz.ch

OK