Assignment 7: Inheritance

ETH Zurich

Hand-out: 12 December 2005
Due: 20 December 2005

1 Dynamic exercise

Goal
Understand the effects of dynamic binding.

Summary
Consider the 2 classes shown in Figure 1. Figure 2 shows how the vertices of a polygon are represented.

Assume:
\( p: POLYGON; r: RECTANGLE; x: REAL \)

Permitted:
\( x := p.\text{perimeter} \)
\( x := r.\text{perimeter} \)
\( x := r.\text{diagonal} \)
\( p := r \) (see Figure 3)

NOT permitted:
\( x := p.\text{diagonal} \) (even just after \( p := r \))
\( r := p \)
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

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

Figure 1: Classes POLYGON and RECTANGLE
Description

Consider the following inheritance hierarchy:

Figure 4: Hierarchy of classes BOOK, TEXTBOOK, and COMICS

and the corresponding class texts from Figure 5, Figure 6, and Figure 7.
class BOOK
create
make

feature -- Initialization

  make is
    -- Initialize book.
    do
    end

feature -- Output

  print_book is
    -- Print message.
    do
      io.put_string("This is a book.
    end

end

Figure 5: Class BOOK

class TEXTBOOK
inherit BOOK
  rename
    print_book as print_textbook
  redefine
    print_textbook
end
create
make

feature -- Output

  print_textbook is
    -- Print message.
    do
      io.put_string("This is a textbook.
    end

end

Figure 6: Class TEXTBOOK
class COMICS

inherit BOOK
    rename print_book as print_comics
    redefine print_comics
end

create make

feature -- Output

    print_comics is
        -- Print message.
        do
            Precursor {BOOK}
            io. put_string ('This is a comics:%n')
        end

end

Figure 7: Class COMICS

Examples

Question 1: Is the following code valid? Explain why or why not.

b: BOOK
create b.make
b. print_book

Answer 1: Yes, because b is of type BOOK and class BOOK has a feature print_book.

Question 2: The code presented in question 1 is valid. What message is printed when executing this code?

Answer 2: "This is a book."

To do

1. Is the following code valid? Explain why or why not.

b: BOOK
create {TEXTBOOK}b.make
b. print_book

2. Is the following code valid? Explain why or why not.

b: BOOK
create {TEXTBOOK}b.make
b. print_textbook
3. One of the code samples presented in question 1 or 2 is valid. What message is printed when executing this code?

4. Is the following code valid? Explain why or why not.
   
   ```
   b: BOOK
   t: TEXTBOOK
   create t.make
   b := t
   b.print_book
   ```

5. Is the following code valid? Explain why or why not.
   
   ```
   b: BOOK
   t: TEXTBOOK
   create t.make
   b := t
   b.print_textbook
   ```

6. One of the code samples presented in question 4 or 5 is valid. What message is printed when executing this code?

7. Is the following code valid? Explain why or why not.
   
   ```
   b: BOOK
   c: COMICS
   create {COMICS} b.make
   c ?= b
   if c /= Void then c.print_book end
   ```

8. Is the following code valid? Explain why or why not.
   
   ```
   b: BOOK
   c: COMICS
   create {COMICS} b.make
   c ?= b
   if c /= Void then c.print_comics end
   ```

9. One of the code samples presented in question 7 or 8 is valid. What message is printed when executing this code?

**To hand in**

Hand in your answers to questions 1 to 9.

## 2 Inherited Fraction

**Goal**

- Inherit from a class.
- Use infix/prefix notation.
Description

*NUMERIC* is a deferred class in the EiffelBase library that exports the following features:

- *one*
- *zero*
- *divisible*
- *exponentiable*
- *infix “+”*
- *infix “−”*
- *infix “/”*
- *infix “∗”*
- *prefix “+”*
- *prefix “−”*

Your task is to implement class *FRACTION (numerator denominator)* inheriting from *NUMERIC*. The test class shown in Figure 8 should work with your implementation without any changes. You can download the source of this class from [http://se.inf.ethz.ch/teaching/ws2005/0001/exercises/fraction_test.e](http://se.inf.ethz.ch/teaching/ws2005/0001/exercises/fraction_test.e).

To do

1. Create a new project with root class *FRACTION_TEST*.
2. Copy and paste the class above in the root class.
3. Create a new class, inherit from *NUMERIC* and implement the missing features.

Hint

- To reduce a fraction, you can use a Greatest Common Divisor (GCD) algorithm, for example the Euclidian algorithm. Try to find this one on the web and adapt it to Eiffel.
- In Eiffel, integer division is done with //, integer remainder (modulo) with \.\.
- Have a closer look at class *FRACTION_TEST* for guidance on how to implement *FRACTION*.

Remarks

Do not forget contracts. This example has a very obvious invariant.
class FRACTION_TEST
create
    make
feature -- Initialization
    a, b, c: FRACTION
    make is
        -- Test the class FRACTION.
        do
            create a.make (1, 2)
            create b.make (3, 4)
            io.put_string ("Calculating with fractions:" + "%N\%N")
            io.put_string ("a : " + a.out)
            io.put_string ("b : " + b.out)
            c := a + b
            io.put_string ("a + b : " + c.out)
            c := a - b
            io.put_string ("a - b : " + c.out)
            c := a * b
            io.put_string ("a * b : " + c.out)
            c := a / b
            io.put_string ("a / b : " + c.out)
        end
end

Figure 8: Class FRACTION_TEST
class
   PLANE

feature -- Status

   below: STRING
      -- What is below the plane

   altitude: INTEGER
      -- Altitude of the plane

feature -- Basic operations

   land is
      -- Land
      require
         earth_below: below.is_equal("earth")
      do
         ...
      ensure
         zero_altitude: altitude = 0
   end

end

Figure 9: Class PLANE

To hand in
Hand in the full source of your class FRACTION. Make sure to upload your learning logs! We appreciate your cooperation!

3 Landing... on your feet

Goal
Understand what happens to contracts with inheritance.

Description
When a routine is redefined in a subclass, its precondition can be kept or weakened, and its postcondition can be kept or strengthened. Hence, in a redefined routine, any precondition is introduced by the keywords require else and any postcondition by the keywords ensure then. Resulting assertions are: original precondition or new_pre and original_postcondition and new_post. Class invariants are accumulated: every class inherits all the invariant clauses of its parents and these clauses are conceptually ”and”-ed.

Assume you have a class PLANE whose code is partly shown in Figure 9. Planes have sensors which detect the altitude at which they are and if they are above earth or water. As the contract of routine land shows, planes can land only under certain circumstances.
Hydroplanes are a special kind of plane. They can land on and take off from both earth and water. If they have landed on water, it means that they have deployed the flotation. You must implement class `HYDROPLANE` as a subclass of `PLANE` and redefine feature `land` (and update its contracts accordingly) so that it reflects the different landing ability and mechanism of hydroplanes.

**To do**

Write class `HYDROPLANE` as a subclass of `PLANE`, redefine the contracts of feature `land` (leaving its body blank) and add any features that you need in the contracts. Make the contracts as complete as possible.

**To hand in**

Hand in the source code for class `HYDROPLANE` and any necessary explanations.