Classroom exercise 2

4 June 2004

The classroom exercise intends to help you self-evaluate your knowledge and skills and lets us gain knowledge about the performance of our students. The setup resembles the situation you will encounter during the final exam. The assistants will be happy to clarify any problems with the formulation of the tasks, but will not solve the tasks for you. This exercise will be corrected and graded by your assistant; the grade will not have any influence on the final exam.

Duration: 1h30 (without break)

Please solve this exercise alone. No material allowed. Don’t forget to write your name and first name on top of each page.

Inheritance (4 points)

Example:

Question: Give the definition of “covariance”.

Answer: Ability to change the argument types when redefining a feature if the types conform to each other.

To do:
1. Give the definition of “polymorphism”. (2 points)
2. Give the definition of “dynamic binding”. (2 points)

Contracts and inheritance (4 points)

Example:

Question: Explain the rules applying to inherited preconditions.

Answer: A routine redeclaration (redefinition or effecting) may keep or weaken the preconditions. It may only use require else (for “or-ing” the preconditions); the keyword require is not valid any more.

To do:
3. Explain the rules applying to inherited postconditions. (2 points)
4. Explain the rules applying to inherited class invariants. (2 points)
Design principles (8 points)

Example:

Question: What is the Command-Query separation principle?

Answer: A command (procedure) does something but does not return a result. A query (function or attribute) returns a result but does not change the state; they should not produce abstract side effects.

To do:

5. What is the Option-Operand separation principle? (2 points)

Example:

Question: Give a typical example of the Command-Query separation principle.

Answer:

```plaintext
class LIST [G]
...
feature -- Basic operation
    search (an_item:G): INTEGER is
        -- Search for `an_item’ and return the index in the list
        -- if found; zero otherwise.
        do
            ...
        end

The feature search is a command; it should not return a result. A proper implementation of the class LIST [G] would be to make the index available as an independent query found_index
```

To do:

6. Give a typical example of the Option-Operand separation principle. (4 points)
Example:
Question: What is the advantage of a design applying the Command-Query separation principle?

Answer: Forbidding side-effects in functions (queries) allow keeping the software in line with its mathematical foundation, an ADT. Maintaining a clear distinction between commands and queries ensures that talking about “functions” in software does not betray the meaning of this term in ordinary mathematics. This principle may be expressed informally as “asking a question should not change the answer”.

To do:
7. What is the advantage of a design applying the Option-Operand separation principle? (2 points)

Obsolete classes and features (6 points)

Example:
Question: What is an obsolete class?

Answer: An obsolete class is a class with an obsolete clause.

To do:
8. What is an obsolete feature? (2 points)

Example:
Question: Give an example of an obsolete class.

Answer: class ARRAY_LIST [G]

obsolete "[

Use MULTI_ARRAY_LIST instead (same semantics, but new name ensures more consistent terminology).
Caution: do not confuse with ARRAYED_LIST (lists implemented by one array each).

]"

inherit MULTI_ARRAY_LIST [G]
end

To do:
9. Give an example of an obsolete feature. (2 points)
10. What is an obsolete feature useful for? (2 points)
Design by Contract (24 points)

Consider the following class BOOK:

```plaintext
class BOOK
create
make
feature {NONE} -- Initialization
make is
  -- Initialize book.
  do
  ...
end
feature -- Status report
  borrowed: BOOLEAN
  -- Is book currently borrowed (i.e. not in the library)?
feature -- Basic operation
  borrow is
    require
      not_borrowed: not borrowed
    do
      borrowed := True
    ensure
      borrowed: borrowed
    end
  return is
    require
      borrowed: borrowed
    do
      borrowed := False
    ensure
      not_borrowed: not borrowed
    end
end
```

To do:

11. Add contracts (preconditions, postconditions, class invariants) to the following class LIBRARY. (Note: There is exactly one assertion missing per dotted line; you may write directly on the exercise sheet.) (24 points)

```plaintext
class LIBRARY
inherit ANY
redefine
default_create
end
feature {NONE} -- Initialization
default_create is
  -- Create ‘books’.
  do
    create books.make
  end
feature -- Access
```
books: LINKED_LIST [BOOK]

-- Books available in the library

feature -- Element change
extend (a_book: BOOK) is
  -- Extend 'books' with 'a_book'.
  require
  ...........................................................................................................................................................
  .................................................................
  do
    books.extend (a_book)
  ensure
  ...........................................................................................................................................................
  .................................................................
end

remove (a_book: BOOK) is
  -- Remove 'a_book' from 'books'.
  require
  ...........................................................................................................................................................
  .................................................................
  do
    books.start
    books.search (a_book)
    books.remove
  ensure
  ...........................................................................................................................................................
  .................................................................
end

feature -- Output
display_books is
  -- Display title of all 'books' available in the library.
  do
    if books.is_empty then
      io.put_string ("No book available at the moment")
    else
      from books.start until books.after loop
        io.put_string (books.item.title)
        books.forth
    end
  end

feature -- Basic operation
borrow_all is
  -- Borrow all 'books' available in the library.
  do
    from books.start until books.after loop
      books.item.borrow
      books.forth
  end

end

invariant
  ...........................................................................................................................................................
  .................................................................
end
Inheritance (24 points)

Consider the following inheritance hierarchy:

and the corresponding class texts:

```kotlin
class BOOK
create
make
feature -- Initialization
    make is
        -- Initialize book.
        do
            ...
        end
end
feature -- Output
    print_book is
        -- Print message.
        do
            io.put_string(“This is a book.%N”)
        end
end

class TEXTBOOK
inherit BOOK
rename print_book as print_textbook
redefine print_textbook
create
make
feature -- Output
    print_textbook is
        -- Print message.
        do
            io.put_string(“This is a textbook.%N”)
        end
end

class COMICS
```
Example:

**Question1:** Is the following code valid? Explain why or why not.

```plaintext
b: BOOK
create b.make
b.print_book
```

**Answer1:** Yes, because `b` is of type `BOOK` and class `BOOK` has a feature `print_book`.

**Question2:** The code presented in question 1 is valid. What message is printed when executing this code?

**Answer2:** This is a book.

To do:

12. Is the following code valid? Explain why or why not. (2 points)

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

13. Is the following code valid? Explain why or why not. (2 points)

```plaintext
b: BOOK
create {TEXTBOOK} b.make
b.print_textbook
```

14. One of the code samples presented in question 12 or 13 is valid. What message is printed when executing this code? (2 points)

15. Is the following code valid? Explain why or why not. (2 points)

```plaintext
t: TEXTBOOK
create t.make
t.print_book
```
16. Is the following code valid? Explain why or why not. (2 points)
   
   \[ \begin{align*} 
   t : & \text{TEXTBOOK} \\
   \text{create} t.\text{make} & \\
   t.\text{print}_\text{textbook} & 
   \end{align*} \]

17. One of the code samples presented in question 15 or 16 is valid. What message is printed when executing this code? (2 points)

18. Is the following code valid? Explain why or why not. (2 points)
   
   \[ \begin{align*} 
   b : & \text{BOOK} \\
   t : & \text{TEXTBOOK} \\
   \text{create} t.\text{make} & \\
   b := & t \\
   b.\text{print}_\text{book} & 
   \end{align*} \]

19. Is the following code valid? Explain why or why not. (2 points)
   
   \[ \begin{align*} 
   b : & \text{BOOK} \\
   t : & \text{TEXTBOOK} \\
   \text{create} t.\text{make} & \\
   b := & t \\
   b.\text{print}_\text{textbook} & 
   \end{align*} \]

20. One of the code samples presented in question 18 or 19 is valid. What message is printed when executing this code? (2 points)

21. Is the following code valid? Explain why or why not. (2 points)
   
   \[ \begin{align*} 
   b : & \text{BOOK} \\
   c : & \text{COMICS} \\
   \text{create} \{ \text{COMICS} \} & b.\text{make} \\
   c := & b \\
   c.\text{print}_\text{book} & 
   \end{align*} \]

22. Is the following code valid? Explain why or why not. (2 points)
   
   \[ \begin{align*} 
   b : & \text{BOOK} \\
   c : & \text{COMICS} \\
   \text{create} \{ \text{COMICS} \} & b.\text{make} \\
   c := & b \\
   c.\text{print}_\text{comics} & 
   \end{align*} \]

23. One of the code samples presented in question 21 or 22 is valid. What message is printed when executing this code? (2 points)

**Using agents (30 points)**

Consider the following class diagram:
and the corresponding class texts:

defered class
  STRATEGY
feature -- Basic operations
  do_something is
    -- Do something.
    deferred
  end
end
class STRATEGY_A
  inherit STRATEGY
feature -- Basic operations
  do_something is
    -- Do something.
    do
      io.put_string("Strategy A\n")
    end
end
class STRATEGY_B
  inherit STRATEGY
feature -- Basic operations
  do_something is
    -- Do something.
    do
      io.put_string("Strategy B\n")
    end
end
class CONTEXT
create
  make
feature {NONE} -- Initialization
  make (a_strategy: like strategy) is
    -- Set 'strategy' to 'a_strategy'.
    require
      a_strategy_not_void: a_strategy /= Void
    do
      strategy := a_strategy
    ensure
      strategy_set: strategy = a_strategy
  end
feature -- Basic operations
    do_something is
        -- Do something. (Call algorithm corresponding to `strategy'.)
        do
            strategy.do_something
        end
    end

feature -- Access
    strategy: STRATEGY
        -- Strategy to be applied
    feature -- Element change
        set_strategy (a_strategy: like strategy) is
            -- Set `strategy' to `a_strategy'.
            require
                a_strategy_not_void: a_strategy /= Void
            do
                strategy := a_strategy
            ensure
                strategy_set: strategy = a_strategy
            end
        invariant
            strategy_not_void: strategy /= Void
        end
    end
class
    APPLICATION
    create
        make
    feature {NONE} -- Initialization
        make is
            -- Do something using different strategies.
            local
                a_context: CONTEXT
            do
                create a_context.make (create {STRATEGY_A})
                a_context.do_something
                a_context.set_strategy (create {STRATEGY_B})
                a_context.do_something
            end
    end

It corresponds to a typical Eiffel implementation of the “Strategy” pattern. The application (corresponding to class APPLICATION) creates a context (corresponding to class CONTEXT) with a certain strategy (corresponding to the class STRATEGY and its descendants STRATEGY_A and STRATEGY_B).
The goal of this exercise is to use agents instead of inheritance to implement the Strategy pattern.

To do:

24. Rewrite the class CONTEXT to have an attribute strategy_procedure of type
    PROCEDURE [ANY, TUPLE] (instead of an attribute strategy of type STRATEGY).
    (Don’t forget to update the implementation of feature do_something.) (10 points)

25. Rewrite the class APPLICATION and possibly the other classes in order to use this
    new implementation of class CONTEXT (i.e. using agents instead of a STRATEGY
    object). (20 points)