

Midterm Test

COSC 3311 3.0 Software Design

Section M, Winter 2007

Family Name: _____

Given Name(s): _____

Student Number: |__| |__| |__| |__| |__| |__| |__| |__| |__| |

Question	Out of	Mark
Q1	25	
Q2	20	
Q3	21	
Q4	18	
Q5	16	
Total	100	
Letter grade		

1. [25 marks] Consider the following root class:

```
class ROOT_CLASS
create make
feature -- Initialization

    make is do
        -- Creates and populates linked lists a and b
    end

    a,b: LINKED_LIST[PERSON]

    method1 is
    do
        -- Implementation goes here
    ensure
        a = b
        equal (a,b)
        deep_equal (a,b)
    end

    method2 is
    do
        -- Implementation goes here
    ensure
        a = old a
        equal (a, old a)
        deep_equal (a, old a)
    end

    method3 is
    do
        -- Implementation goes here
    ensure
        b = old b
        equal (b, old b.twin)
        deep_equal (b, old b.deep_twin)
    end

end -- class ROOT_CLASS
```

Class PERSON is given in the next page (only some features are shown):

```
class PERSON

feature
  age:INTEGER
  name:STRING

  set_age(a:INTEGER)  is
    do
      age := a
    end
end
```

For each one of the features method1, method2, and method3, do the following:

If you believe that any of the postconditions is redundant, indicate which and why. If you believe that all postconditions are necessary, provide three example implementations of the feature. Each implementation must cause a different postcondition to be false, while the other two are true. Following are some features of LINKED_LIST[G] you might need:

```
start          -- Make the first item in the list the current item.
replace (v: G) -- Replace current item by v.
remove         -- Remove current item.
```

For method1 and method 2, the first postcondition implies the other two. For method 3, all postconditions are necessary. Following are three implementations. Each one violates exactly one of the postconditions.

```
-- Violates only post1
b := b.twin
b.start -- These two lines are not necessary for full marks
b.back  -- In practice, one needs them for the postcondition to fail

-- Violates only post2
b.start
b.replace (b.item.twin)
b.back -- Not necessary for full marks

-- Violates only post3
b.start
b.item.set_age(3)
```

2. [20 marks] Consider that you are developing a computer game that shows a number of different screens to the user. Each screen contains some graphics and, if enabled, a GUI that allows the user to select various options from different menus. The following set of classes implements part of this system. Class GAME is the root class. The implementation of features whose name starts with draw are not shown.

```
class GAME
  create make

  feature

    make is
      local
        s1,s2,s3: SCREEN
      do
        create screens.make
        create {INITIAL_SCREEN} s1
        create {MIDDLE_SCREEN} s2
        create {FINAL_SCREEN} s3
        screens.extend(s1)
        screens.extend(s2)
        screens.extend(s3)
      play
    end

    screens: LINKED_LIST[SCREEN]

    play is
      do
        -- Present the screens to the user in order
      end
    end
end
```

```
deferred class
  SCREEN
  feature
    render is deferred end
end
```

```
class INITIAL_SCREEN
inherit SCREEN
feature
  gui_enabled: BOOLEAN
  render is
    do
      draw_graphics_1
      if gui_enabled then
        draw_menu_1
        draw_options_1
      end
    end
end
end
```

```
class MIDDLE_SCREEN
inherit SCREEN
feature
  enabled: BOOLEAN
  render is
    do
      draw_graphics_2
      if enabled then
        draw_menu_2
        draw_options_2
      end
    end
end
end
```

```
class FINAL_SCREEN
inherit SCREEN
feature
  gui: BOOLEAN
  render is
    do
      draw_graphics_3
      if gui then
        draw_menu_3
        draw_options_3
      end
    end
end
end
```

[10 marks] Identify a problem with the design shown above. Explain clearly why it is a problem.

[10 marks] Present an alternative design that alleviates the problem.

Duplication in the code of render. If the way screens are rendered changes, all subclasses of SCREEN have to be updated. Source of bugs. Violation of Single Choice Principle.

Solution: Make render a template method in SCREEN.

3. [21 marks] Consider the following three implementations for the play feature of class GAME of the previous question. One of these implementations will not compile. Indicate which one, and explain why. For the two implementations that compile correctly, indicate which one is a better design and why.

```
play is
  local
    i: INITIAL_SCREEN
    m: MIDDLE_SCREEN
    f: FINAL_SCREEN
  do
    screens.start
    i := screens.item
    i.render
    screens.forth
    m := screens.item
    m.render
    screens.forth
    f := screens.item
    f.render
  end
```

Will not compile. The first assignment causes a compilation error since it assigns a superclass reference to a subclass reference.

```
play is
  local
    i: INITIAL_SCREEN
    m: MIDDLE_SCREEN
    f: FINAL_SCREEN
  do
    screens.start
    i ?= screens.item
    i.render
    screens.forth
    m ?= screens.item
    m.render
    screens.forth
    f ?= screens.item
    f.render
  end
```

Compiles. Not very good design. Simulates polymorphism. Will break if there is a different set of screens in the linked list.


```
play is
do
  from
    screens.start
  until
    screens.after
  loop
    screens.item.render
    screens.forth
  end
end
```

Compiles. Best design, uses polymorphism correctly.

4. [18 marks] Consider the following three classes. In the university they refer to, it is possible for teachers to enroll in courses. The enrollment procedure is different for teachers since they do not have to satisfy prerequisites. Teaching assistants may enroll either as students or teachers depending on which course they are interested in.

```
deferred class UNIVERSITY_PERSON
feature
  enroll is deferred end
end
```

```
class STUDENT
inherit UNIVERSITY_PERSON

feature
  enroll is
    do
      -- Student enrollment procedure
    end
end
```

```
class TEACHER
inherit UNIVERSITY_PERSON

feature
  enroll is
    do
      -- Teacher enrollment procedure
    end
end
```

Class TEACHING_ASSISTANT inherits from both TEACHER and STUDENT. Due to the multiple inheritance, class TEACHING_ASSISTANT may have to use various adaptation mechanisms to resolve problems such as name clashes etc. The boxes in the next two pages present a number of possibilities (only the inheritance clause is shown). You can assume that when a redefinition clause exists, the class does contain an appropriate redefinition.

Inside each box, indicate whether the class will compile correctly. If not, explain why. For the versions that compile, indicate which one is the better design.

```
inherit
  TEACHER
  STUDENT
```

No, two definitions of enroll.

```
inherit
  TEACHER
    redefine enroll end
  STUDENT
```

No, two definitions of enroll.

```
inherit
  TEACHER
    redefine enroll end
  STUDENT
    redefine enroll end
```

Yes, good design.

```
inherit
  TEACHER
  STUDENT
  undefine enroll end
```

Yes, not great design, TAs enroll like teachers.

```
inherit
  TEACHER
    rename enroll as teacher_enroll end
  STUDENT
    rename enroll as student_enroll end
```

No, one needs to be selected.

```
inherit
  TEACHER
    select enroll end
  STUDENT
    rename enroll as student_enroll end
```

Yes, not great design, university persons that happen to be teaching assistants will enroll like teachers.

5. [16 marks] Consider a class called CAR that contains the following features:

```
gas: INTEGER
speed: INTEGER
accelerate is
  do
    -- Implementation goes here
  ensure
    gas = old gas - 1
    speed = old speed + 1
  end
```

Class CAR contains many other features not shown here. Class SPORTS_CAR inherits from CAR. However, sports cars use 2 units of gas every time the accelerate feature is called. Describe how you would modify the design to accommodate this fact. Assume that due to other features not shown here, the inheritance link between SPORTS_CAR and CAR needs to be preserved.

Change the first postcondition of accelerate to

```
gas = old gas - gas_usage
```

Add the following feature in class CAR

```
gas_usage: INTEGER is
  do
    Result := 1
  end
```

Redefine the feature in class SPORTS_CAR as follows

```
gas_usage: INTEGER is
  do
    Result := 2
  end
```

