Mock Exam 1

ETH Zurich

November 7,8 2011

Name: _____

Group: _____

1 Terminology (10 points)

Goal

This task will test your understanding of the object-oriented programming concepts presented so far in the lecture. This is a multiple-choice test.

Todo

Place a check-mark in the box if the statement is true. There may be multiple true statements per question; 0.5 points are awarded for checking a true statement or leaving a false statement un-checked, 0 points are awarded otherwise.

1. A command...
   - ☐ a. call is an instruction.
   - ☐ b. may modify an object.
   - ☐ c. may appear in the precondition and the postcondition of another command but not in the precondition or the postcondition of a query.
   - ☐ d. may appear in the class invariant.
2. The syntax of a program...
   - ☐ a. is the set of properties of its potential executions.
   - ☐ b. can be derived from the set of its objects.
   - ☐ c. is the structure and the form of its text.
   - ☐ d. may be violated at run-time.
3. A class...
   - ☐ a. is the description of a set of possible run-time objects to which the same features are applicable.
   - ☐ b. can only exist at runtime.
   - ☐ c. cannot be declared as expanded; only objects can be expanded.
   - ☐ d. may have more than one creation procedure.

4. Immediately before a successful execution of a creation instruction with target x of type C...
   - ☐ a. $x = Void$ must hold.
   - ☐ b. $x /= Void$ must hold.
   - ☐ c. the postcondition of the creation procedure may not hold.
   - ☐ d. the precondition of the creation procedure must hold.
5. Void references...
   - ☐ a. cannot be the target of a successful call.
   - ☐ b. are not default values for any type.
   - ☐ c. indicate expanded objects.
   - ☐ d. can be used to terminate linked structures (e.g. linked lists).

2 Design by Contract (10 Points)

2.1 Task

Your task is to fill in the contracts (preconditions, postconditions, class invariants, loop variants and invariants) of the class *CAR* according to the specification given in the comments. You are not allowed to change the class interface or the given implementation. Note that the number of dotted lines does not indicate the number of missing contracts.

```
class
    CAR

create
    make

feature {NONE} -- Creation

    make
            -- Creates a default car.
        require
            ...

            ...

            ...
        do
            create {LINKED_LIST [CAR_DOOR]} doors.make
        ensure
            ...

            ...

            ...
        end

feature {ANY} -- Access

    is_convertible : BOOLEAN
```

```
32      -- Is the car a convertible (cabriolet)? Default: no.

34    doors: LIST [CAR_DOOR]
        -- The doors of the car. Number of doors must be 0, 2 or 4. Default: 0.
36
    color: COLOR
38      -- The color of the car. 'Void' if not specified. Default: 'Void'.

40 feature {ANY} -- Element change

42    set_convertible (a_is_convertible: BOOLEAN)
        require
44
            .....
46
            .....
48
            .....
50      do
            is_convertible := a_is_convertible
52      ensure
54          .....
56          .....
58          .....
        end
60
    set_doors (a_doors: ARRAY [CAR_DOOR])
62      require
64          .....
66          .....
68          .....
        local
70        door_index: INTEGER
        do
72        doors.wipe_out
          if a_doors /= Void then
74          from
                door_index := 1
76          invariant
78              .....
80              .....
82              .....
            until
84              door_index > a_doors.count
            loop
```

```
86                doors.extend (a_doors [door_index])
                  door_index := door_index + 1
88            variant
90                ..........
92                ..........
94                ..........
              end
96          end
        ensure
98
            ..........
100
            ..........
102
            ..........
104     end

106   set_color (a_color: COLOR)
        require
108
            ..........
110
            ..........
112
            ..........
114     do
            color := a_color
116     ensure
118         ..........
120         ..........
122         ..........
        end
124
invariant
126
    ..........
128
    ..........
130
    ..........
132
end
```

3 Inheritance: A Persistence Framework (12 Points)

Read the background information, look at the class diagram and code, and then answer task 1 and task 2.

3.1 Background Information

The following classes represent a simplified persistence framework. A persistence framework offers services to store and retrieve objects. A serialization manager is used to store objects using a certain medium (memory or file) and a certain format, like binary. Figure 1 shows the corresponding class diagram. Listings 1, 2, 3, 4, and 5 show a few lines of code from some of these classes.

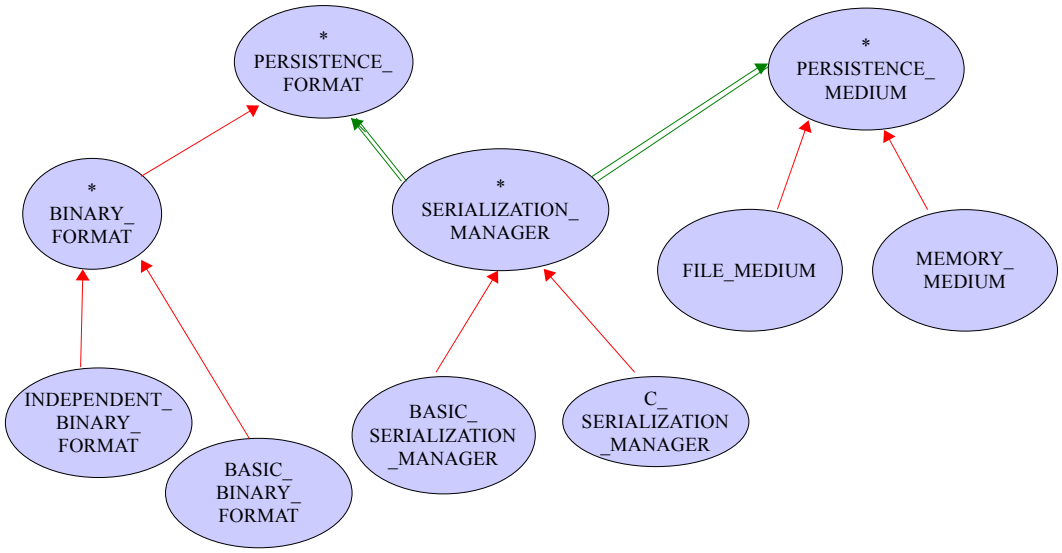

Figure 1: Class diagram for the persistence framework

Listing 1: class *SERIALIZATION_MANAGER*

```
deferred class
    SERIALIZATION_MANAGER

feature -- Access

    format: PERSISTENCE_FORMAT
            -- The format used for serialization.

    medium: PERSISTENCE_MEDIUM
            -- The medium used for serialization.

    retrieved_item: ANY
            -- Object retrieved.

feature -- Creation

    make
                -- Provide format and medium for the current serializer.
        deferred
        ensure
            format_set: format /= Void
            medium_set: medium /= Void
        end

feature -- Basic operations
```

```
    store (a_object: ANY)
                -- Serialize 'a_object' using the format and medium set for current object.
        require
            object_exists : a_object /= Void
        deferred
        end

    retrieve
                -- Retrieve an object using the medium and format set for current serializer.
                -- Set the retrieved object in 'retrieved_item'
        deferred
        end
end
```

Listing 2: class *BASIC_SERIALIZATION_MANAGER*

```
class
    BASIC_SERIALIZATION_MANAGER

inherit
    SERIALIZATION_MANAGER

create make

feature -- Creation

    make
            -- Provide format and medium for the current serializer.
        do
            print ("Creating a basic serialization manager.")
            -- Other necessary initialization.
        end

feature -- Basic operations

    store (a_object: ANY)
                -- Serialize 'an_object' using the format and medium set for current object.
        do
            print ("Serializing an object.")
        end

    retrieve
                -- Retrieve an object using the medium and format set for current serializer.
                -- Set the retrieved object in 'retrieved_item'
        do
            print ("Deserializing an object.")
        end
end
```

Listing 3: class *PERSISTENCE_MEDIUM*

```
deferred class
    PERSISTENCE_MEDIUM

feature -- Access
```

```
    name: STRING
            -- Current persistence medium name.

feature -- Basic operations

    write (a_object: ANY)
            -- Write 'a_object' on the current medium.
        require
            object_exists : an_object /= Void
        deferred
        end
end
```

Listing 4: class *FILE_MEDIUM*

```
class
    FILE_MEDIUM

inherit
    PERSISTENCE_MEDIUM

create
    make

feature -- Initialization

    make
            -- Create a file medium.
        do
            print ("Creating a file.")
        end

feature -- Basic operations

    write (a_object: ANY)
            -- Write 'a_object' on the current medium.
        do
            print ("Writing a file.")
        end
end
```

Listing 5: class *PERSISTENCE_FORMAT*

```
deferred class
    PERSISTENCE_FORMAT

feature -- Access

    header: STRING
                -- Meta-information about the serialization format.

    body: STRING
                -- Main serialization content.

feature -- Status setting
```

```
    set_header (a_header: STRING)
                -- Set header for serialization.
        require
            a_header_not_void: a_header /= Void
        do
            header := a_header
        ensure
            header_set: header = a_header
        end

    set_body (a_body: STRING)
                -- Set body for serialization.
        require
            a_body_not_void: a_body /= Void
        do
            body := a_body
        ensure
            body_set: body = a_body
        end
end
```

3.2 Task 1

Put checkmarks in the checkboxes corresponding to the correct answers. There is at least one correct answer per question. Multiple correct answers per question are possible. The number of points for each correctly marked statement may vary. For every incorrectly marked statement you will be taken away 1 point. If the sum of your points is negative, you will receive 0 points.

1. **Suppose you want the framework to provide support for XML stored in a text file. Which two of the following solutions seem the most appropriate to you?**
   - **a.** Add one new class, namely *XML_FORMAT*, and make it inherit from *PERSISTENCE_FORMAT*. ☐
   - **b.** Add the necessary code to handle the XML format to class *PERSISTENCE_FORMAT*. In addition, add a new class named *XML_SERIALIZATION_MANAGER* and make it inherit from *SERIALIZATION_MANAGER*. ☐
   - **c.** Add three new classes, namely *XML_FORMAT*, *TEXTUAL_FORMAT*, and *XML_SERIALIZATION_MANAGER*. The first of them, *XML_FORMAT*, will inherit from the second, *TEXTUAL_FORMAT*. In addition, *TEXTUAL_FORMAT* will inherit from *PERSISTENCE_FORMAT* and *XML_SERIALIZATION_MANAGER* will inherit from *SERIALIZATION_MANAGER*. ☐
   - **d.** Add one new class, *TEXTUAL_FORMAT*, including the necessary code to serialize data in XML format, and make it inherit from *PERSISTENCE_FORMAT*. ☐
   - **e.** Add two new classes, *XML_FORMAT* and *XML_SERIALIZATION_MANAGER*. Make *XML_FORMAT* inherit from *PERSISTENCE_FORMAT*, and make *XML_SERIALIZATION_MANAGER* inherit from *SERIALIZATION_MANAGER*. ☐
   - **f.** Add two new classes, *XML_FORMAT* and *XML_SERIALIZATION_MANAGER*. Then add to class *SERIALIZATION_MANAGER* two attributes having types *XML_FORMAT* and *XML_SERIALIZATION_MANAGER*. ☐

3.3 Task 2

For each code fragment below, state if it compiles or not. If it does NOT compile, explain why it doesn't compile. If it does compile, write down what is printed at the console. Assume assertion

2. **Suppose you have to write the code for feature *store* in a new class *ADVANCED_SERIALIZATION_MANAGER* that inherits from *BASIC_SERIALIZATION_MANAGER*. What do you have to do to be able to reuse the existing implementation of feature *store* in *BASIC_SERIALIZATION_MANAGER*, and adding some code to it? The new code should be placed after the reused code.**

   a. In *ADVANCED_SERIALIZATION_MANAGER*, use the keyword **redefine** after the clause **inherit** *BASIC_SERIALIZATION_MANAGER*, and specify the new implementation in the body of feature *store*. ☐

   b. In *BASIC_SERIALIZATION_MANAGER*, specify the new implementation in the body of feature *store*. Nothing else is necessary because feature *store* is not implemented in class *SERIALIZATION_MANAGER*. ☐

   c. In *ADVANCED_SERIALIZATION_MANAGER*, use the keyword **undefine** after the clause **inherit** *BASIC_SERIALIZATION_MANAGER*, and specify the new implementation in the body of feature *store*. ☐

   d. In *BASIC_SERIALIZATION_MANAGER*, use the keyword **redefine** after the clause **inherit** *BASIC_SERIALIZATION_MANAGER*, and specify the new implementation in the body of feature *store*. In addition, use the keyword **Precursor** to reuse the implementation from *SERIALIZATION_MANAGER*. ☐

   e. In *ADVANCED_SERIALIZATION_MANAGER*, use the keyword **redefine** after the clause **inherit** *BASIC_SERIALIZATION_MANAGER*, and specify the new implementation in the body of feature *store*. In addition, use the keyword **Precursor** to reuse the implementation from *BASIC_SERIALIZATION_MANAGER*. ☐

   f. In *ADVANCED_SERIALIZATION_MANAGER*, use the keyword **undefine** after the clause **inherit** *BASIC_SERIALIZATION_MANAGER*, and specify the new implementation in the body of feature *store*. In addition, use the keyword **Precursor** to reuse the implementation from *BASIC_SERIALIZATION_MANAGER*. ☐

checking is off.

1.
```
manager_1: SERIALIZATION_MANAGER
manager_2: BASIC_SERIALIZATION_MANAGER
an_object: STRING
...
create manager_1.make
create manager_2.make
create an_object.make_from_string ("test")
manager_1 := manager_2
manager_1.store (an_object)

.....

.....
```

2.
```
manager_1: SERIALIZATION_MANAGER
an_object: STRING
...
create {BASIC_SERIALIZATION_MANAGER}manager_1.make
create an_object.make_from_string ("test")
manager_1.store (an_object)

.....

.....
```

3.
```
manager_1: SERIALIZATION_MANAGER
manager_2: BASIC_SERIALIZATION_MANAGER
an_object: STRING
...
create manager_2.make
create an_object.make_from_string ("test")
manager_1 := manager_2
manager_1.store (an_object)

.....

.....
```

4.
```
manager_1: SERIALIZATION_MANAGER
manager_2: BASIC_SERIALIZATION_MANAGER
an_object: STRING
...
create manager_2.make
create an_object.make_from_string ("test")
manager_2 := manager_1
manager_2.store (an_object)

.....

.....
```

4 Inversion of Linked List (10 Points)

The classes *SINGLE_LINKED_LIST* [*G*] and *SINGLE_CELL* [*G*] implement a single linked list. The first cell of the list is stored in the attribute *first* of the class *SINGLE_LINKED_LIST* [*G*]. Attribute *next* of class *SINGLE_CELL* [*G*] delivers the next cell . Calling *next* on the last cell will return a *Void* reference.

Implement the feature *invert* of class *SINGLE_LINKED_LIST* [*G*], so that it inverts the order of the elements in the list. For example, inverting the list [6, 2, 8, 5] results in [5, 8, 2, 6]. **Do not** create new objects of type *SINGLE_CELL* [*G*] and also **do not** introduce any new feature in class *SINGLE_LINKED_LIST* [*G*] and *SINGLE_CELL* [*G*].

```
class
  SINGLE_LINKED_LIST [G]

feature -- Access

  first : SINGLE_CELL [G]
      -- Head element of the list, 'Void' if the list is empty

feature -- Basic operations

  invert
      -- Invert the order of the elements of the list .
      -- E.g. the list [6, 2, 8, 5] should become [5, 8, 2, 6].
    local
      .....

      .....
```

```
18
        .....
20
    do
22      .....
24      .....
26      .....
28      .....
30      .....
32      .....
34      .....
36      .....
38      .....
40      .....
    end
42 end
```

```
class
2   SINGLE_CELL [G]

4 feature -- Access

6   next: SINGLE_CELL [G]
        -- Reference to the next generic list cell of a list
8
  feature -- Element change
10
    set_next (an_element: SINGLE_CELL [G])
12      -- Set 'next' to 'an_element'.
    ensure
14      next_set: next = an_element

16 end
```