Final exam – Object Oriented Programming (236703)

Spring 2013, Moed Alef

Lecturer: Dr. Keren Lenz

Teaching assistants: Eran Gilad, Eyal Moscovici, Amit Carmeli

Instructions:

The exam contains 5 equally graded questions. You must answer all of them.

1. In case you don't know the answer to a question or a sub-question, you may write "I don't know" and receive 20% of the points.

2. Unless specified otherwise, on questions in which you are required to provide the output of given code, the code might not compile or run to completion. On such cases, you should describe the reason for the compilation or run-time error.

3. Make your answers precise. Adding irrelevant information (even if correct) might cause a deduction of points.

4. Make sure your handwriting is readable. No cursive please.

5. Start every answer on a new page.

6. No additional material is allowed.

7. The exam contains 9 pages. Make sure your form includes all of them.

8. The exam's duration is 3 hours.

Good luck!
Question 1: C++ Object Model

The C++ Standard committee has decided to follow the success of C#, Java and Squeak, and add an additional level to the language's object model. Namely, each class should have a meta-class that has a representation at run-time.

a. How many levels does each of the 4 languages described above have in its object model?

b. The opponents of the addition have claimed it contradicts one of the language's fundamental principles. Which principle is that? Explain.

c. The opponents have also claimed C++ has an additional "half-level" beyond the formal number anyway, and that "half-level" allows some of the capabilities provided by a full-blown additional level. What is that "half-level"? What kind of classes enjoys its capabilities?

d. The supporters of the addition have presented a winning argument: adding a level will provide support for the Mark phase in a Mark & Sweep garbage collection (in which all reachable objects are first marked, and then non-reachable ones are freed). What feature of the additional level does the argument above rely on?

**Hint:** Think why the language as it is now does not support a garbage collection implementation that is both safe (namely does not free reachable objects) and complete (does free all unreachable objects). You may assume the garbage collection implementation has a way to obtain all object pointed by local and global variables (known as "roots"), and that every pointer either points on a real object or NULL (no uninitialized or dangling pointers).

e. In order to secure their win, the supporters of the addition have provided yet another argument: the additional level will make virtual destructors redundant, even without a garbage collection system. Explain:

   a. When and why is a virtual destructor required?
   b. How will an additional level eliminate that requirement?
Question 2: C++ Multiple Inheritance

Consider the following C++ code:

```cpp
struct A {
    A() { cout << "A() ";
        f(); }

    virtual void f() { g(); }
    virtual void g() { cout << "A::g" << endl; }
};

struct B : virtual A {
    B() : A() { cout << "B() ";
        h1();
        f(); }

    virtual void g() { cout << "B::g" << endl; }

    void h1() {
        B* b = dynamic_cast<B*>(this);
        if (NULL == b)
            cout << "(not a B) ";
        else
            cout << "(also a B) ";
    }
};

struct C : virtual A {
    C() : A() { cout << "C() ";
        h2();
        f(); }

    void h2() {
        B* b = dynamic_cast<B*>(this);
        if (NULL == b)
            cout << "(not a B) ";
        else
            cout << "(also a B) ";
    }
};

struct D : B, C {
    D() : B(), C() { cout << "D() ";
        h1();
        h2(); }
};

a. What will be the output of the following commands?

D* d = new D();
d->f();
```
b. Will your answer to the previous sub-question change if class D had been defined as following (the rest of the code unchanged)?

```
struct D : C, B
```

If so, explain the differences (an exact output is not required). If not, explain why.

**The following sub-questions do not rely on sub-question b.**

c. Add the following function g to class C:

```
virtual void g() { cout << "C::g" << endl; }
```

Would this addition change your answer to sub-question a? If so, specify the output differences. If not, explain why.

d. Following the previous sub-question, now add the following function g to class D:

```
virtual void g() { cout << "D::g" << endl; }
```

Would this addition change your answer to sub-question a? If so, specify the output differences. If not, explain why.

e. Following the previous sub-question, now add the following function f to class D:

```
virtual void f(int x) { g(); }
```

Would this addition change your answer to sub-question a? If so, specify the output differences. If not, explain why.
Question 3: Java's Object Class

A student in the Object Oriented Programming course would like to implement a class named MyClass in Java. The class should support the equals, clone and hashCode methods, defined in Object as follows:

```java
protected Object clone() throws CloneNotSupportedException;
public int hashCode();
public Object equals();
```

The student has defined the class as follows:

```java
public class MyClass { ... }
```

The student has defined the clone method as follows:

```java
@Override
public MyClass clone() throws CloneNotSupportedException {
    return ______________;
}
```

a. Does the method signature comply with the rules of conformance?

b. Complete the method's body so that the code will compile but throw a CloneNotSupportedException at run-time. Explain.

c. The student has defined the signatures of the equals and hashCode methods as follows:

1. ```java
   @Override
   public boolean equals(MyClass c)
   ```

2. ```java
   @Override
   protected int hashCode()
   ```

For each one of the methods, does its signature comply with the rules of conformance? Explain.
d. The student would also like MyClass to implement the interface Comparable:

```java
interface Comparable<T> {
    int compareTo(T t);
}
```

The compareTo method should return 0 in case the compared objects are equal, a positive number in case the object given as argument is smaller, and a negative number in case the argument is larger. Consider the equals method's contract (given below) and answer:

Given `x.equals(z) == false` and `x.equals(y) == true`,

1. What is guaranteed for `x.compareTo(z)` and `x.compareTo(y)`?
2. Is the compareTo method transitive? Explain.
3. Is the compareTo method symmetric? Explain.
4. For what value of z will the expression `z.compareTo(x)` throw an exception for every x?

Reminder: the equals method determines whether two objects are equal, and returns true or false accordingly. Following is the method's contract, taken from its documentation:

Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is **reflexive**: for any non-null reference value x, `x.equals(x)` should return true.
- It is **symmetric**: for any non-null reference values x and y, `x.equals(y)` should return true if and only if `y.equals(x)` returns true.
- It is **transitive**: for any non-null reference values x, y, and z, if `x.equals(y)` returns true and `y.equals(z)` returns true, then `x.equals(z)` should return true.
- It is **consistent**: for any non-null reference values x and y, multiple invocations of `x.equals(y)` consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value x, `x.equals(null)` should return false.
Question 4: Squeak

Consider the following Squeak code:

Object subclass: #A
  instanceVariableNames: 'x'
  classVariableNames: ''
  poolDictionaries: ''
  category: ''

A subclass: #B
  instanceVariableNames: 'y'
  classVariableNames: ''
  poolDictionaries: ''
  category: ''

m
(A new) x.       "Line 1"
(A new) y.       "Line 2"
(B new) y.       "Line 3"
Transcript show: x.       "Line 4"
Transcript show: y.       "Line 5"

a. As you know, in Squeak all instance variables are protected. Specify which lines in
the method m could access the field on the following cases:

1. m is a method defined in A
2. m is a method defined in B
3. m is a method defined in A class (meta-classes have no special access
   rights)

For the following sub-questions, assume an object in Squeak is represented in
memory as an array of references to other objects.

b. For which types does that assumption hold?

c. Why are there types for which that assumption does not hold?

On the following sub-questions we shall add a language feature that will allow
indirect access to public instance variables of an object, relying on the assumption
above. Namely, the following code should run correctly:

Object subclass: #Student
  publicInstanceVariableNames: 'name'
  instanceVariableNames: '...' 
  classVariableNames: ''
  poolDictionaries: ''
  category: ''

s := Student new.
sn := s getPublicInstanceVariable: #name. "sn contains the student's
name"

d. Squeak has a method named basicAt: index. The method returns the value of an
instance variable stored in the given index, in the object's references array (index is a
natural number). Where would it be reasonable for basicAt: to be implemented?
e. In order to allow access to a public field, the field's name should be mapped to its index. We shall call that mapping instanceVarMap. Which class should store that mapping?

f. Implement the method getPublicInstanceVariable: aSymbol, which takes a name of a public instance variable and returns its variable (as shown in the code example above). You must make sure the field is indeed public.

**Instructions:**

1. Specify the class in which the method should be defined.
2. Specify any assumptions you have made in order to implement the method.
3. Implement the method.
**Question 5: C# Generic Extension Methods**

Consider the following code:

```csharp
public delegate O func<I, O>(I i);

public static class ExtensionOperation
{
    public static IEnumerable<T> foo<T, U, V>(IEnumerable<T> c,
                                                func<T, U> f,
                                                func<T, V> g)
    {
    ...
    }
}
```

a. Which classes does the method `foo` above extend?

Note that generic extension methods allow providing additional functionality to classes using composition instead of inheritance. The following sub-questions will discuss generic extension methods.

**Reminder:**

- Extension methods have a lower priority than regular class methods. Namely, they are used only in case the class does not provide any appropriate method.
- Extension methods must be defined in a static class. Namely, classes that inherit Object directly, and all of their methods are static.

b. Could using generic extension methods cause *coincidental* ambiguity? If so, provide an example. If not, explain why.

c. Could using generic extension methods cause *inherent* ambiguity? If so, provide an example. If not, explain why.

d. Does *mixins* provide any added value over generic extension methods? Provide code that demonstrates the difference between the two, or explain why there is none.

e. Does *traits* provide any added value over generic extension methods? Provide code that demonstrates the difference between the two, or explain why there is none.