CSE 1030 3.0 Introduction to Computer Science II
A Sample of Test 2

1

A constructor of a subclass usually starts with a special statement that communicates with its parent class.

(a) What is the general syntax of this statement?

```
super(<zero or more arguments>);
```

(b) What is its purpose; that is, why do we use it?

It allows the subclass to select the constructor to be used for instantiating the superclass and to pass parameters to it. In other words, it allows the subclass to initialize the state of its parent.

(c) Can this statement appear anywhere in the subclass or only in special locations? In the latter case, specify these locations.

No, it must be the very first statement of the body of a constructor.

(d) What happens if we do not use this statement at all in the subclass?

If absent, the compiler automatically inserts super(), that is, one that selects the default constructor of the superclass.

2

Consider the class `A` shown below.

```
public class A
{
   private int x;
   private double y;

   public A(int x, double y)
   {
      this.setX(x);
      this.setY(y);
   }
   public void setX(int x)
   {
      this.x = x;
   }
   public void setY(double y)
   {
      this.y = y;
   }
}
```

We want to create the following subclass.

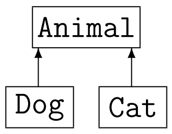
```
public class B extends A
{
   private String s;

   public B(String s, int x, double y)
   {
   }
}
```

Write the body of the constructor.

```
super(x, y);
this.s = s;
```

3

Consider the following UML class diagram in which the `Animal` class is abstract.



(a) Why is the class `Animal` abstract?

Because one should not be able to construct Animal objects (one cannot create instances of an abstract class).

(b) The class `Animal` contains a constructor. Explain why is there a constructor even though the class is abstract.

To initialize the attributes common to Dog and Cat (these attributes are declared in Animal).

(c) Give an example of a non-abstract method that may be found in `Animal`.

getAge

4

Classes `A` and `B` have some common features (attributes and methods). We are given two options:

1. Make `A` and `B` implement an interface `F`.
2. Make `A` and `B` extend an abstract class `S`.

We seek to understand the differences between these two options as far as the common features are concerned. Note that we are not after syntax rules or language-level differences; we want to make a choice based on software engineering principles.

(a) Which option handles the common features better? You need to justify your answer (i.e. explain why one option is better than the other) and also explain the meaning of "better" (i.e. the benefits of choosing it).

Option 2 is better because common features can be stored in the abstract class `S` *and doing so avoids code duplication, which, in turn, avoids inconsistencies. The interface* `F`*, on the other hand, can only hold method headers and final attributes so it does very little to eliminate redundancy.*

(b) Suppose that option 1 was chosen. Write down the implementation of `F` in the space below. Assume that the common features consist of two attributes (an `int` and a `String`), and the void method `absorb` that takes a `double` parameter.

```
public interface F
{
   public void absorb(double x);
}
```

5

Implement the following method:

```
/**
   Given an iterator over a collection of integers, return a list
   containing every other integer; that is, the first, third, fifth, etc.

   @param iterator an iterator over a collection of integers.
   @return a list of every other integer.
*/
public List<Integer> getEveryOther(Iterator<Integer> iterator)
```

```
public List<Integer> getEveryOther(Iterator<Integer> iterator)
{
   List<Integer> list = new ArrayList<Integer>();
   boolean odd = true;
   while(iterator.hasNext())
   {
      if (odd)
      {
         list.add(iterator.next());
      }
      else
      {
         iterator.next();
      }
      odd = !odd;
   }
   return list;
}
```

6

(a) Consider the classes `A` and `B`:

```
public class A
{
   public int m() throws XXXX
   {
      ...
   }
}

public class B extends A
{
   public int m() throws YYYY
   {
      ...
   }
}
```

The literals `XXXX` and `YYYY` can be replaced with either `AException` or `BException` (assume that these exception classes exist and that `BException` extends `AException`). Determine which of the following combinations is allowed and justify your answer:

1. `XXXX = AException` and `YYYY = BException`
2. `XXXX = BException` and `YYYY = AException`

Combination 1 is allowed. Class `B` *extends class* `A` *and overrides method* `m`*. Therefore, method* `m` *of class* `B` *should obey the contract of method* `m` *of class* `A`*. Since a* `BException` *is-an* `AException`*, this is the case for combination 1.*

(b) Consider the class `A`:

```
public class A
{
   public int m() throws XXXX
   {
      ...
      ... throw new YYYY();
      ...
   }
}
```

Note that one of the statements in method `m()` includes throwing an instance of exception `YYYY`. The literals `XXXX` and `YYYY` can be replaced with either `AException` or `BException` (assume that these exception classes exist, that they are *checked* exceptions, and that `BException` extends `AException`). Determine which of the following combinations is allowed and justify your answer:

1. `XXXX = AException` and `YYYY = BException`
2. `XXXX = BException` and `YYYY = AException`

Combination 1 is allowed. In this case, method `m` *throws a* `BException`*, which is-an* `AException`*. Hence, it satisfies the contract specified by the method header.*

7

The `Student` class has the following API:

public class **Student**

`public` **Student**`(int ID, String name)`

Construct a student with the given ID and name.

Parameters:
`ID` – the ID of this student.
`name` – the name of this student.

`public int` **getID**`()`

Return the ID of this student.

Returns:
the ID of this student.

The `GraduateStudent` class has the following API:

public class **GraduateStudent** extends Student

`public` **GraduateStudent**`(int ID, String name, String supervisor)`

Construct a graduate student with the given ID, name, and supervisor.

Parameters:
`ID` – the ID of this graduate student.
`name` – the name of this graduate student.
`supervisor` – the name of the supervisor of this graduate student.

`public boolean` **isEligible**`()`

Determine the eligibility of this graduate student.

Returns:
true if the ID is not zero and the supervisor is neither null nor empty,
false otherwise.

Write down a complete implementation of the `GraduateStudent` class. Do not include any javadoc documentation.

```
public class GraduateStudent extends Student
{
   private String supervisor;

   public GraduateStudent(int ID, String name, String supervisor)
   {
      super(ID, name);
      this.supervisor = supervisor;
   }

   public boolean isEligible()
   {
      return this.getID() != 0 &&
         this.supervisor != null &&
         !this.supervisor.equals("");
   }
}
```

8

How will your answer to the previous question change if the `supervisor` attribute of `GraduateStudent` (and the corresponding parameter of its constructor) were represented as `StringBuffer` instead of `String`? Justify your answer by drawing a memory diagram. Explore both normal aggregation and composition.

Read Section 4.2.4.