DUE DATE: By the end of your lab session

CSE-271: Object-Oriented Programming

Exercise #6

Max Points: 20

Name: Raobot

For your own convenient reference – You should first save/rename this document using the naming convention `MUid_Exercise6.docx` (example: `raodm_Exercise6.docx`) prior to proceeding with this exercise.

Objectives: The objectives of this exercise are to:

1. Review the concepts of inheritance & interfaces
2. Review concepts of polymorphic method calls
3. Gain familiarity with JUnit based testing
4. Gain familiarity with code coverage

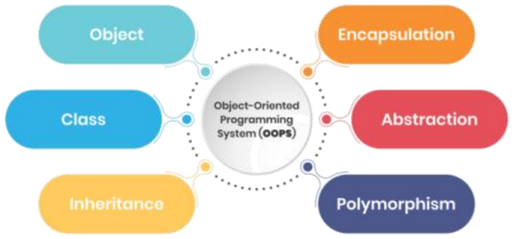
Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from Eclipse output into this document. You may discuss the questions or seek help from your neighbor, TA, and/or your instructor.

Part #0: One time setup of Eclipse (IDE) – Only if needed

We already configured `Eclipse`'s source formatter and Checkstyle plug-in as part of Lab #1. If your `Eclipse` is not configured (because you are using a different computer) then use the instructions from Lab #1 to configure `Eclipse`.

Part #1: Generic concepts of Object-oriented programming

Estimate time: < 30 minutes



Background: Object-oriented Programming (OOP) is a programming paradigm that is widely used and adopted by several mainstream programming languages, such as: C++, C#, JavaScript, Java, and Python. Hence, understanding the generic concepts underlying OOP is very important for your future careers, immaterial of the programming-language that you may be working with. Moreover, clearly and concisely explaining concepts is a very important skills for your future job-interviews and in your jobs. Hence, the exams also involve such questions, and in the labs, we will practice this style of questions.

Exercise: Briefly (2-to-3 sentences each) respond to the following questions regarding generic concepts of object-oriented programming (OOP).

1. In a Java class, what are the 2 types of methods cannot be overridden to accomplish polymorphism?

   In Java, the following two types of methods cannot be overridden:
   - In Java, `final` methods cannot be overridden
   - In Java, `static` methods cannot be overridden

2. Briefly state at three differences between an `abstract` class and an `interface`?

Abstract class	**Interface**
It is a parent of some child class.	It is not a parent class, but simply specifies certain characteristics of a class.
Can have (non-static, non-final) instance variables	Cannot have instance variables
Abstract classes can have some non-abstract methods.	All methods in an interface are abstract.

3. What is a copy constructor? Show an example of a copy-constructor.

   A copy constructor is used to make a duplicate or copy of a given object. Copy constructors may make deep-copies or shallow copies as illustrated by the following example:

```
class Student {
    private int[] scores;
    private String name;

    /** Copy constructor.
      *
      * @param src The source object to be copied.
      */
    public Student(Student src) {
        // name is Immutable, prefer shallow copy
        this.name = src.name;
        // scores are Mutable, prefer deep copy
        this.scores = Arrays.copyOf(src.scores, src.scores.length);
    }
}
```

4. Given the adjacent `Student` class, show the source code for a <u>shallow-copy constructor</u> for the `Student` class.

```
/** Copy constructor.
  * @param src The source object to be copied.
  */
public Student(Student src) {
     this.name = src.name;
     this.scores = src.scores;
}
```

```
class Student {
    private int[] scores;
    private String name;
}
```

5. Given the adjacent `Student` class, show the source code for a deep-copy constructor for the `Student` class.

```java
/** Copy constructor.
 *
 * @param src The source object to be copied.
*/
public Student(Student src) {
    // name is Immutable, prefer shallow copy
    this.name = src.name;
    // scores are Mutable, prefer deep copy
    this.scores = Arrays.copyOf(src.scores,
src.scores.length);
}
```

```java
class Student {
    private int[] scores;
    private String name;
}
```

6. What is the difference between using a copy-constructor versus the `clone` method?

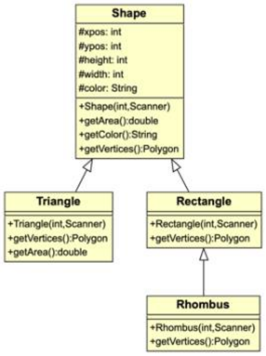
In order to use the copy-constructor we have to know the child class that we are creating a copy – i.e., it is not a polymorphic call. For example:

```java
Rectangle rec = new Rectangle(src);
```

On the other hand, the `clone` method provides a polymorphic approach to making a copy of an object – i.e., we don't have to know the final derived class.

7. Given the adjacent UML (that shows all the subclasses of `Shape`), rewrite the following `duplicate` method to use copy-constructors instead of the `clone` method.

```java
/**
 * Make a copy of an abstract shape assuming it
 * implements the Clonable interface.
 * @param src The source shape to be duplicate.
 * @return A copy of the source shape.
 */
Shape duplicate(Shape src) {
    return src.clone();
    if (src instanceof Rhombus) {
        return new Rhombus((Rhombus) src);
    } else if (src instanceof Rectangle) {
        return new Rectangle((Rectangle) src);
    } else {
        return new Triangle((Triangle) src);
    }
}
```



Part #2: Object-oriented Programming (OOP) in Java

Estimated time: 30 minutes

Background: Different programming languages provide slightly different ways to accomplish OOP. Java's approach requires implementing several standard methods and interfaces for each class, specifically:

- Subclasses are encouraged to override the `equals` method
- It is recommended to implement the `Comparable` interface and implement the `compareTo` method

- It is convenient if Clonable interface is implemented by overriding the clone method.

Setup: First, you need to setup a Java project in Eclipse, download the starter code. In this part of the exercise, you are given the following set of classes:

File Name	***Description***
Student.java	An abstract student class. **Do not modify this class**.

Exercise: In this part of the exercise, you are required to develop a subclass of Student called Undergraduate. Hence, first briefly review the Student class. Next, develop the Undergraduate class (that extends Student) with the following methods:

Method	**Description**
Constructor to initialize name.	Simply calls constructor in parent class.
Copy constructor	Must perform deep copy.
equals	Two Undergraduate objects are equal only if name and all scores are equal.
compareTo	Two Undergraduate objects are to be compared first based on name and (if name is equal) then based on average scores.
clone	Clones an object

Testing: In the next part of this exercise, you will be developing JUnit tests to test the operations of these methods.

Part #3: Unit testing and code coverage

Estimated time: 30 minutes

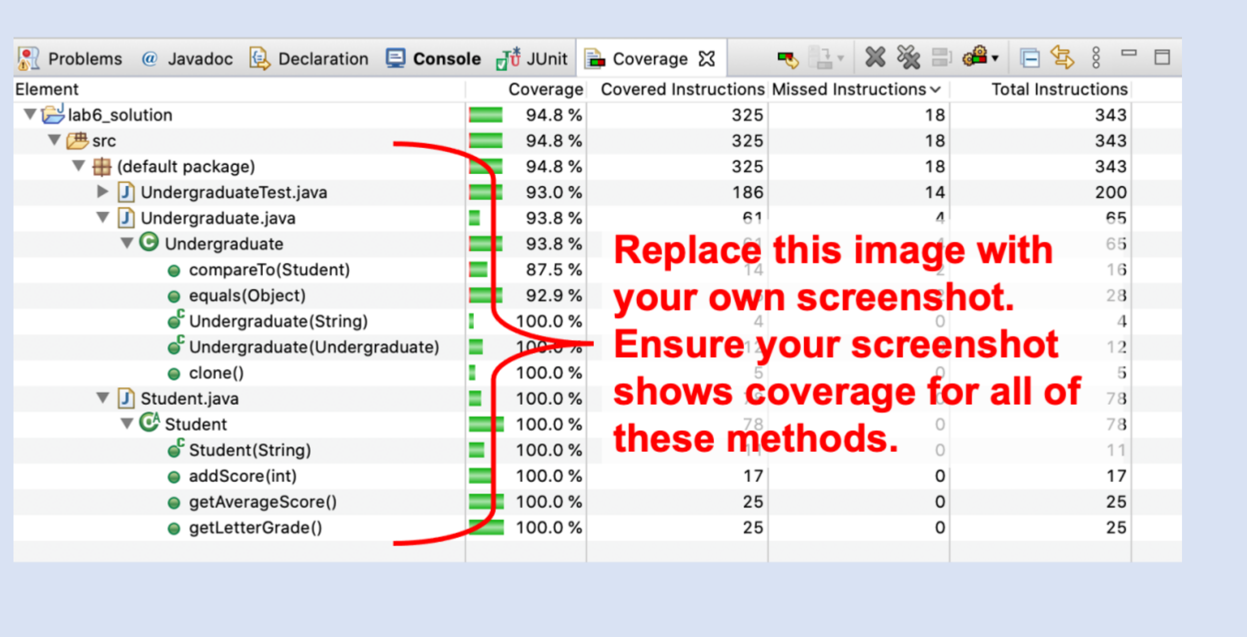
Background: Software testing is a very important aspect of software development. Testing plays a vital role in ensuring the overall quality and reliability of a software system. Hence, different types of software testing approaches are widely used, including: Unit testing, functional testing, regression testing, performance testing, etc.

The most common form of testing that is performed by programmers is unit testing. Unit testing typically involves testing the functionality of a subset of methods or a class. There are several different frameworks for unit testing. The JUnit framework is one of the most widely used frameworks for unit testing Java classes. Hence, it is important to have familiarity with using JUnit for unit testing.

Exercise: In this part of the exercise, you are expected to develop a `JUnit` testing class named `UndergraduateTest.java` to test the `Undergraduate` and consequently the `Student` classes developed in the previous section. In addition, you are expected to ensure that you have at least 80% test coverage on both of these two classes.

There are videos on Canvas demonstrating the use of `JUnit` via Eclipse along with using code coverage tools in Eclipse. It may be useful to quickly review these videos as part of this exercise.

It is recommended you create 4 separate tests to test the operation of `clone`, `compareTo`, `getLetterGrade`, and addScore (with invalid score) methods. Once you have successfully developed your unit test and verified code coverage, place a screenshot showing at least 80% coverage in the space below. Ensure your screenshot shows coverage for each method in `Undergraduate` and `Student` class as shown in the example screenshot below:



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
lab6_solution	94.8 %	325	18	343
src	94.8 %	325	18	343
(default package)	94.8 %	325	18	343
UndergraduateTest.java	93.0 %	186	14	200
Undergraduate.java	93.8 %	61	4	65
Undergraduate	93.8 %	61	4	65
compareTo(Student)	87.5 %	14	2	16
equals(Object)	92.9 %	[illegible]	2	28
Undergraduate(String)	100.0 %	4	0	4
Undergraduate(Undergraduate)	100.0 %	12	[illegible]	12
clone()	100.0 %	5	0	5
Student.java	100.0 %	[illegible]	[illegible]	78
Student	100.0 %	78	0	78
Student(String)	100.0 %	11	0	11
addScore(int)	100.0 %	17	0	17
getAverageScore()	100.0 %	25	0	25
getLetterGrade()	100.0 %	25	0	25

Replace this image with your own screenshot. Ensure your screenshot shows coverage for all of these methods.

Part #4: Submit to Canvas via CODE plug-in

Estimated time: < 5 minutes

Exercise: You will be submitting the following files via the Canvas CODE plug-in:

1. This MS-Word document saved as a PDF file – **Only submit PDF file**.
2. The Java source files: `Undergraduate.java` and `UndergraduateTest.java`, that you modified in this exercise.

Ensure you actually complete the submission on Canvas by verifying your submission