

## **CSE-271: Object-Oriented Programming**

### **Exercise #6**

Max Points: 20

**Name:** Jacob Igel



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

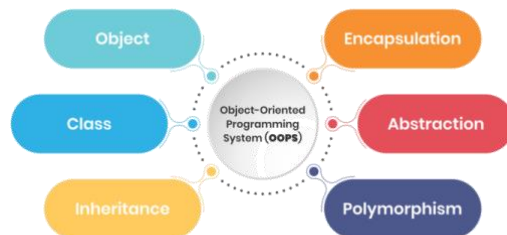


We already configured Eclipse's source formatter and Checkstyle plug-in as part of Lab #1. If your Eclipse is not configured (because you are using a different computer) then use the instructions from Lab #1 to configure Eclipse.

### 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?

The methods in the parent must not be **static** or defined to be **final**.

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

| Abstract class                              | Interface   |
|---|---|
| Can have abstract and non-abstract methods  | Can only have abstract methods                    |
| You can implement the interface             | You cannot implement an abstract class            |
| Can be extended using the keyword 'extends' | Can be implemented using the keyword 'implements' |

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

A copy constructor is a constructor that is used to make copies of other objects. You can use a shallow copy which resues the objects references or a deep copy that creates new objects and copies values.

```
Public class Student implements Cloneable, Comparable<Student> {  
    protected String name;  
    protected int[] scores;  
  
    public Student(Student src) {  
        this.name = src.name; //shallow copy  
        this.scores = Arrays.copyOf(src.scores, src.scores.length); // deep copy  
    }  
}
```

4. Given the adjacent Student class, show the source code for a shallow-copy constructor for the Student class.

```
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.

```
public Student(Student src) {  
    this.name = src.name; //shallow copy  
    this.scores = Arrays.copyOf(src.scores,  
src.scores.length); // deep copy  
}
```

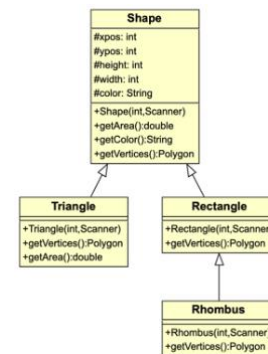
```
class Student {  
    private int[] scores;  
    private String name;  
}
```

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

The `clone` method makes an exact copy of the structure of the object while the copy-constructor will copy the complete structure as well as the data itself.

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.

```
/**  
 * Make a copy of an abstract shape assuming it  
 * implements the Cloneable interface.  
 * @param src The source shape to be duplicate.  
 * @return A copy of the source shape.  
 */  
Shape duplicate(Shape src) {  
    return src.clone();  
}  
  
Public Shape(Shape src) {  
    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 `Cloneable` 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. <b>Do not modify this class.</b> |

**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.  |
| <code>equals</code>             | Two <code>Undergraduate</code> objects are equal only if name and all scores are equal.  |
| <code>compareTo</code>          | Two <code>Undergraduate</code> objects are to be compared first based on name and (if name is equal) then based on average scores. |
| <code>clone</code>              | 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_2                      | 25.3 %   | 109                  | 321                 | 430                |
| src                         | 25.3 %   | 109                  | 321                 | 430                |
| (default package)           | 25.3 %   | 109                  | 321                 | 430                |
| UndergraduateTest.java      | 19.5 %   | 55                   | 227                 | 282                |
| UndergraduateTest           | 19.5 %   | 55                   | 227                 | 282                |
| testCompareTo()             | 0.0 %    | 0                    | 117                 | 117                |
| testClone()                 | 0.0 %    | 0                    | 54                  | 54                 |
| testAddScore()              | 0.0 %    | 0                    | 52                  | 52                 |
| testLetterGrade()           | 85.2 %   | 23                   | 4                   | 27                 |
| createUnder(String, int...) | 100.0 %  | 29                   | 0                   | 29                 |
| Undergraduate.java          | 5.7 %    | 4                    | 66                  | 70                 |
| Student.java                | 64.1 %   | 50                   | 28                  | 78                 |

## 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