Software Developer Course Assessment

Quantitative Assessment Practice

Course Name: Advanced Programming (Java)

Current Week: 6th june 2024

Submission date:12th june 2024

Introduction:

The purpose of this assessment is to help us understand how the class is doing in terms of the review material that we have covered during the previous couple of weeks. The **only** purpose of this assessment is for us to improve our approach to review and ensure that what we're currently doing is an effective strategy. Completion of this assessment is **mandatory - if you don't submit a solution, it will be marked as incomplete. You must complete a minimum of 80% of your assigned QAPs per course – otherwise you will be marked as incomplete for that course no matter how good your other grades are.** If you do submit a solution, it will be marked as complete, as you will receive full marks no matter what your actual performance was – again this is a participation grade.

Again, the goal here is to help you all in the best way that we can, so please do be honest when answering the questions related to how long it took, which resources you used, etc. And please ensure that you do your **own** work – don't just copy off a friend to get it done, earnestly do your best with it. If you can't get it completely working, give us what you have. While it will be graded, the grade will not count against you, it's just a way for us to see where everybody is, and to know which concepts, if any, we, as a class, may be struggling with.

Deadline: You will have until the end of the day on , 12th june (11:59pm) to submit your assessment solutions. Please ensure you answer all the questions outlined in the instructions portion of this document as well in your submission.

Marking: In this program core evaluation is marked with one of three possible marks: *Incomplete, Pass, Pass Outstanding.* For QAPs, though, where incomplete marks are more important for our own information as well as for the information of the student, we wanted to increase the resolution of our grading system. Therefore, QAPs are marked on a scale of 1-5. The details of this marking system are summarized in the table below.

Grade	**Meaning**
1	*Incomplete.* Student shows severe lack of understanding of the material – solution is heavily incomplete, non-functional, or completely off base of what the assignment was asking for.
2	*Partially Complete.* Students show some understanding of the material. Solution may be non-functional or partially functional, but the approach is correct, albeit with some major bugs or missing features.
3	*Mostly Complete.* Student demonstrates understanding of the major ideas of the assignment. Solution is mostly working, albeit with a few small bugs or significant edge cases which were not considered. Shows a good understanding of the correct approach, and is either nearly a feature-complete solution, or is a feature-complete solution with some bugs.
4	*Complete (Equivalent to: Pass.)* Student shows complete understanding of assigned work and implemented all necessary features. Any bugs that are present are insignificant (for example aesthetic bugs when testing the functionality of code) and do not impact the core functionality in a significant way. All necessary objectives for the assignment are completed, and the student has delivered something roughly equivalent to the canonical solution in terms of features and approach.
5	*Complete with Distinction (Equivalent to: Pass Outstanding)* The student demonstrates a clear mastery of the subject matter tested by the QAP. The solution goes above and beyond in some way, makes improvements on the canonical solution, or otherwise demonstrates the student's mastery of the subject matter in some way. A solution in this category would consider all reasonable edge cases and implement more than the necessary functionality required by the assignment.

Instructions:

You are allowed to complete the assessment problems below in whatever way you can but please answer the following questions/points as part of your submission:

1. How many hours did it take you to complete this assessment? (Please keep try to keep track of how many hours you have spent working on each individual part of this assessment as best you can - an estimation is fine; we just want a rough idea.)

2. What online resources you have used? (My lectures, YouTube, Stack overflow etc.)

3. Did you need to ask any of your friends in solving the problems. (If yes, please mention name of the friend. They must be amongst your class fellows.)

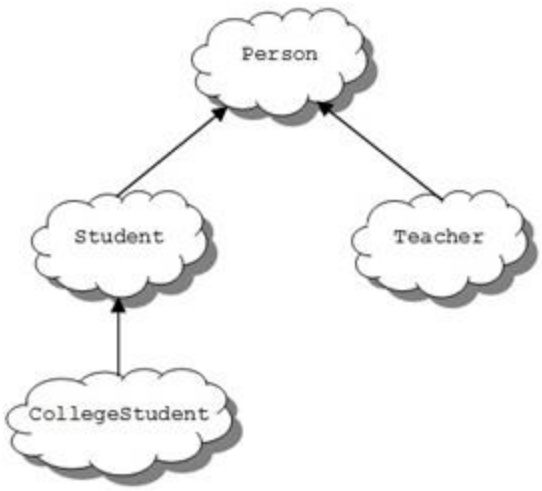
4. Did you need to ask questions to any of your instructors? If so, how many questions did you ask (or how many help sessions did you require)?

5. Rate (subjectively) the difficulty of each question from your own perspective, and whether you feel confident that you can solve a similar but different problem requiring some of the same techniques in the future now that you've completed this one.

Problem#1:

A HighSchool application has two classes: the Person superclass and the Student subclass. Using inheritance, in this lab you will create two new classes, Teacher and CollegeStudent. A Teacher will be like Person but will have additional properties such as *salary* (the amount the teacher earns) and *subject* (e.g. "Computer Science", "Chemistry", "English", "Other"). The CollegeStudent class will extend the Student class by adding a *year*(current level in college) nd *major* (e.g. "Electrical Engineering", "Communications", "Undeclared").

The inheritance hierarchy would appear as follows:



Listed below is the Person base class to be used as a starting point for the Teacher class:

```
class Person {

protected String myName ;   // name of the person
protected int myAge;        // person's age
protected String myGender;  // "M" for male, "F" for female

public Person(String name, int age, String gender)  {

myName = name; myAge = age ; myGender = gender;   }

public String toString()  {
return myName + ", age: " + myAge + ",
gender: " +myGender;
}
}
```

The Student class is derived from the Person class and used as a starting point for the CollegeStudent class:

```
class Student extends Person {
protected String myIdNum;    // Student Id Number
protected double myGPA;      // grade point average

public Student(String name, int age, String gender, String idNum, double gpa)  {
// use the super class' constructor
super(name, age, gender);

// initialize what's new to Student

myIdNum = idNum;

myGPA = gpa;

}    }
```

Tasks:

1. Add methods to "set" and "get" the instance variables in the Person class. These would consist of: getName, getAge, getGender, setName, setAge, and setGender.

2. Add methods to "set" and "get" the instance variables in the Student class. These would consist of: getIdNum, getGPA, setIdNum, and setGPA.

3. Write a Teacher class that extends the parent class Person.

    a. Add instance variables to the class for *subject* (e.g. "Computer Science", "Chemistry",, "English", "Other") and*salary* (the teachers annual salary). *Subject* should be of type String and *salary* of type double. Choose appropriate names for the instance variables.

b. Write a constructor for the Teacher class. The constructor will use five parameters to initialize myName, myAge, myGender, *subject*, and *salary*. Use the super reference to use the constructor in the Person superclass to initialize the inherited values.

c. Write "setter" and "getter" methods for all of the class variables. For the Teacher class they would be: getSubject, getSalary, setSubject, and setSalary.

d. Write the toString() method for the Teacher class. Use a super reference to do the things already done by the superclass.

4. Write a CollegeStudent subclass that extends the Student class.

a. Add instance variables to the class for *major* (e.g. "Electrical Engineering", "Communications", "Undeclared") and *year* (e.g. FROSH = 1, SOPH = 2, …). *Major* should be of type String and *year* of type int. Choose appropriate names for the instance variables.

b. Write a constructor for the CollegeStudent class. The constructor will use seven parameters to initialize myName, myAge, myGender, myIdNum, myGPA, *year*, and *major*. Use the super reference to use the constructor in the Student superclass to initialize the inherited values.

c. Write "setter" and "getter" methods for all of the class variables. For the CollegeStudent class they would be: getYear, getMajor, setYear, and setMajor.

d. Write the toString() method for the CollegeStudent class. Use a super reference to do the things already done by the superclass.

5. Write a testing class with a main() that constructs all of the classes (Person, Student, Teacher, and CollegeStudent) and calls their toString() method. Sample usage would be:

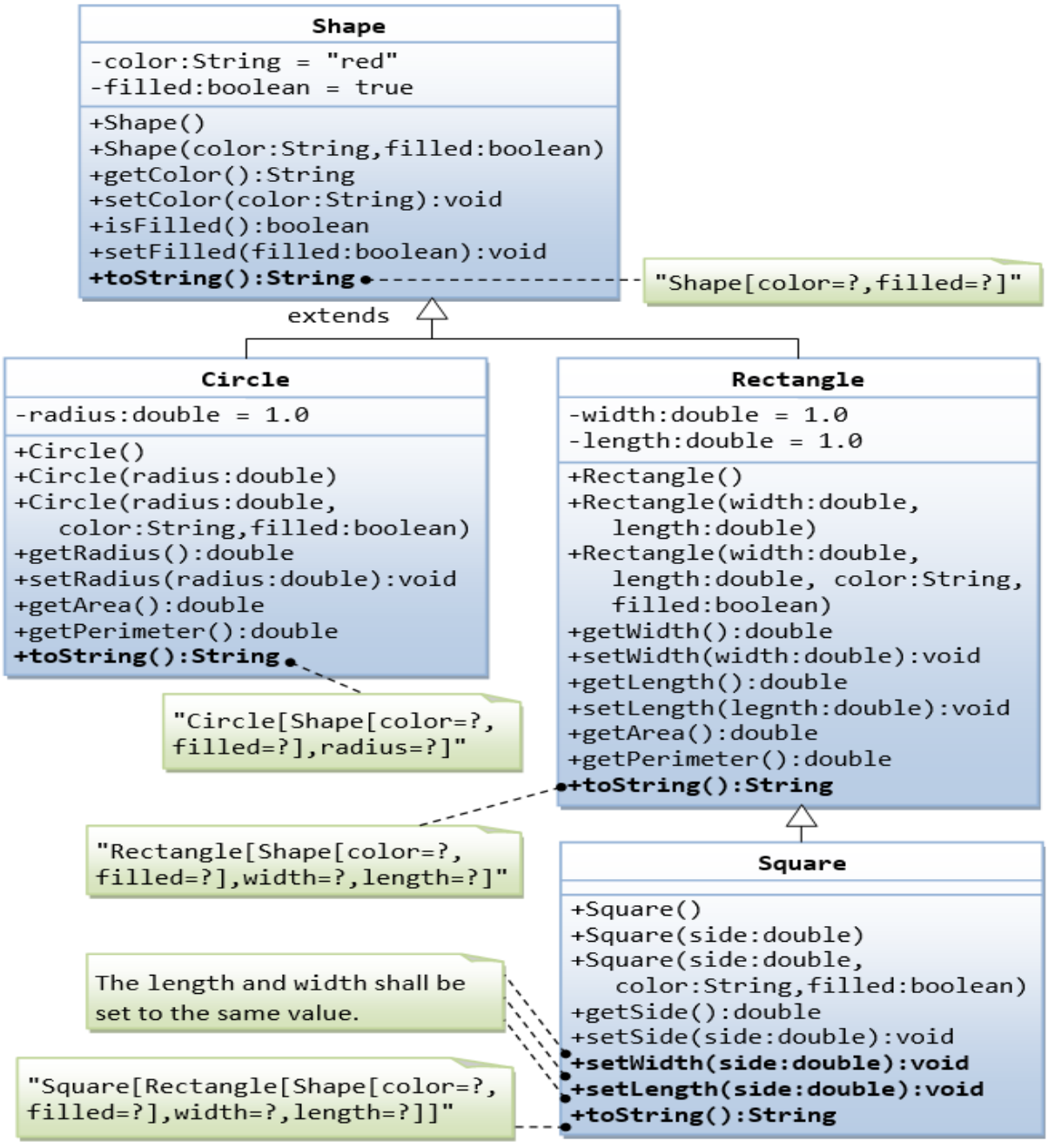
```
Person bob = new Person("Coach Bob", 27, "M");
System.out.println(bob);

Student lynne = new Student("Lynne Brooke", 16, "F", "HS95129″, 3.5);
System.out.println(lynne);

Teacher mrJava = new Teacher("Duke Java", 34, "M", "Computer Science", 50000);|
System.out.println(mrJava);

CollegeStudent ima = new CollegeStudent("Ima Frosh", 18, "F", "UCB123″, 4.0, 1, "English");

System.out.println(ima);
```

Deliverables:

Complete and working-class files with proper comments.

1. *Person.java*
2. *Student.java*
3. *Teacher.java*
4. *CollegeStudent.java*
5. *Demo.java*
6. *Screenshot of the running code's output*

Problem#2: Superclass Shape and its subclasses Circle, Rectangle and Square



1. Write a superclass called Shape (as shown in the class diagram), which contains:

   a) Two instance variables color (String) and filled (boolean).

   b) Two constructors: a no-arg (no-argument) constructor that initializes the color to "green" and filled to true, and a constructor that initializes the color and filled to the given values.

   c) Getter and setter for all the instance variables. By convention, the getter for a boolean variable xxx is called isXXX() (instead of getXxx() for all the other types).

d) A toString() method that returns "A Shape with color of xxx and filled/Not filled".

(Optional: Write a test program to test all the methods defined in Shape.)

2. Write two subclasses of Shape called Circle and Rectangle, as shown in the class diagram.

The **Circle** class contains:

a) An instance variable radius (double).

b) Three constructors as shown. The no-arg constructor initializes the radius to 1.0.

c) Getter and setter for the instance variable radius.

d) Methods getArea() and getPerimeter().

e) Override the toString() method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

The **Rectangle** class contains:

a) Two instance variables width (double) and length (double).

b) Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.

c) Getter and setter for all the instance variables.

d) Methods getArea() and getPerimeter().

e) Override the toString() method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

3. Write a class called **Square**, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram). Hint:

```
public Square(double side) {
   super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the toString() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.
- Do you need to override the getArea() and getPerimeter()? Try them out.
- Override the setLength() and setWidth() to change both the width and length, so as to maintain the square geometry.

4. Write a testing class with a main() that constructs all of the classes and calls their toString() method also their area and perimeter methods. (Example Code)

```
Shape S = new Shape("Red", true);
Circle C = new Circle(5.0);
Circle C1 = new Circle(5.0,"blue",true);
```

```
Rectangle R = new Rectangle(2.0,3.0);
Rectangle R1 = new Rectangle(2.0,3.0,"green",false);
Square Sq = new Square(4.0);
Square Sq1 = new Square(4.0,"yellow",true);
S.toString();
C.toString();
//Print here (C.getArea() and C.getPerimeter())
C1.toString();
//Print here(C1.getArea() and C1.getPerimeter())
R.toString();
//Print(R.getArea() and R.getPerimeter())
R1.toString();
//Print(R1.getArea() and R1.getPerimeter())
Sq.toString();
Print(sq.getArea() and sq.getPerimeter())
Sq1.toString();
Print(sq1.getArea() and sq1.getPerimeter())
```

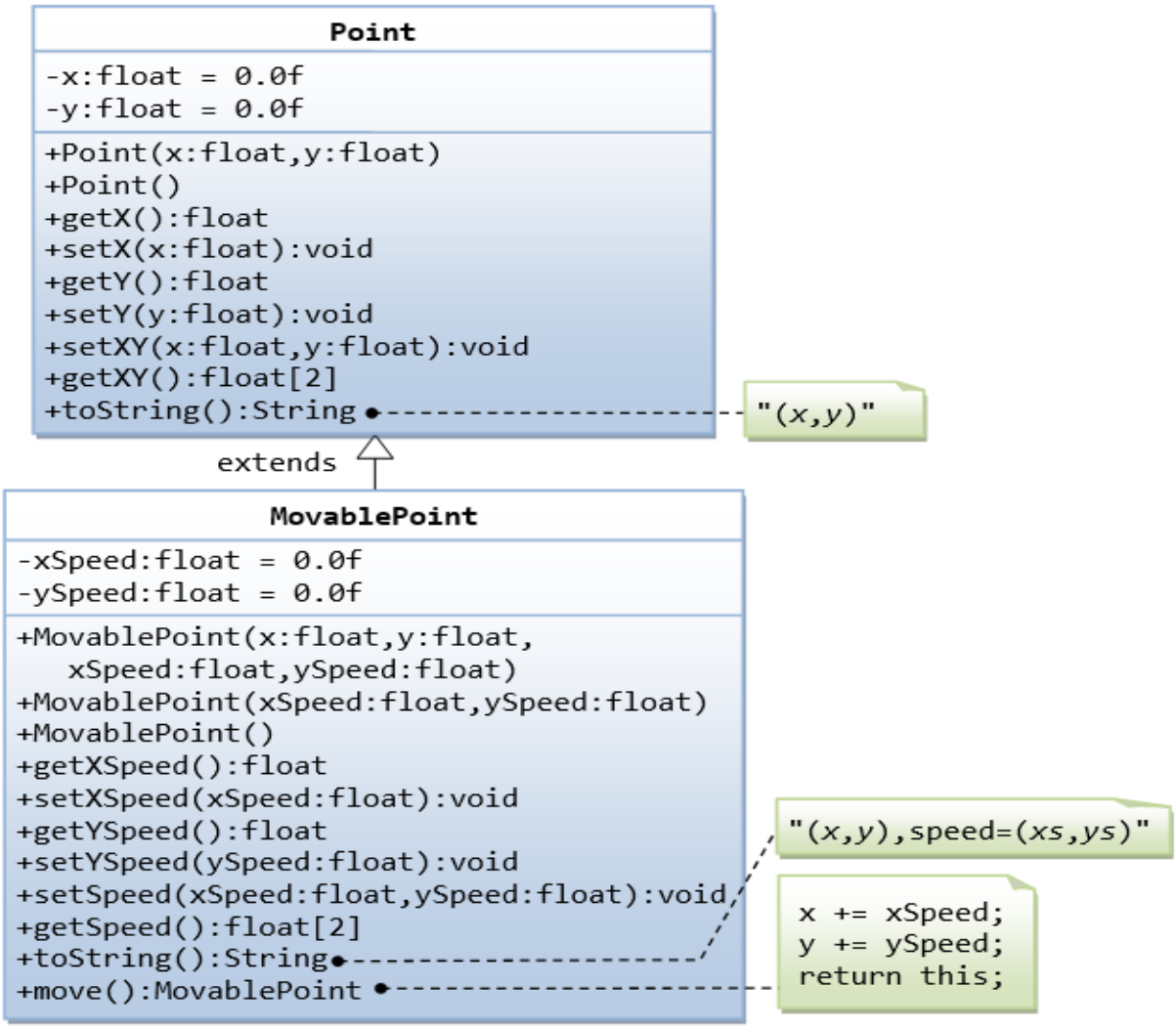
Deliverables:

Complete and working-class files with proper comments.

1. *Shape.java*
2. *Circle.java*
3. *Rectangle.java*
4. *Square.java*
5. *Demo.java*
6. *Screenshot of the running code's output*

Problem#3:

Write the classes as shown in the following class diagram. Also write a demo class with the main method, to show the working of the application. Mark all the overridden methods with annotation @Override.



Hints

1. You cannot assign floating-point literal say 1.1 (which is a double) to a float variable, you need to add a suffix f, e.g. 0.0f, 1.1f.
2. The instance variables x and y are private in Point and cannot be accessed directly in the subclass MovablePoint. You need to access via the public getters and setters. For example, you cannot write x += xSpeed, you need to write setX(getX() + xSpeed).

Deliverables:

Complete and working-class files with proper comments.

1. *Point.java*
2. *MovablePoint.java*
3. *Demo.java*
4. *Screenshot of the running code's output*