

# Software Developer Course Assessment

## Quantitative Assessment Practice 3

**Course Name:** Advanced Programming (Java)

**Current Week:** 5<sup>th</sup> November 2025

**Submission Deadline:** 12th November 2025

### Introduction:

This assessment is designed to gauge your understanding of the review material covered in recent weeks. Its primary goal is to improve our teaching strategies by identifying areas that may require additional attention.

Participation in this assessment is **mandatory**. You must complete **at least 80% of the assigned QAPs** for each course. Failure to do so will result in your course being marked as incomplete, regardless of your other grades.

If you **submit a solution**, it will be marked as **complete**, and you will receive full participation marks—**regardless of correctness**. We encourage you to submit your best attempt, even if incomplete. **Do not copy** others' work; the purpose is to identify learning gaps honestly.

❖ Final Submission Date: **12<sup>th</sup> November 2025 (End of Day)**

❖ Ensure all questions in the **Instructions** section are answered in your submission.

### Grading Criteria:

QAPs are marked on a 1 to 5 scale as follows:

Grade	Description
1	Incomplete: Severe lack of understanding; the solution is non-functional or unrelated.
2	Partially Complete: Shows basic understanding; partial or buggy implementation.
3	Mostly Complete: Demonstrates good grasp of core ideas; mostly functional with minor bugs.
4	Complete (Pass): Functional, all requirements met; minor, non-critical bugs only.
5	Complete with Distinction (Pass Outstanding): Exceeds expectations; handles edge cases, shows mastery.



## **Instructions (To be included in your submission):**

Please answer the following:

1. How many hours did it take you to complete this assessment?  
*(Please estimate per problem if possible)*
2. What online resources did you use?  
*(e.g., Lectures, YouTube, Stack Overflow, etc.)*
3. Did you get help from any classmates?  
*(If yes, provide names; they must be enrolled in your class)*
4. Did you ask for help from an instructor?  
*(Mention the number of questions/help sessions)*
5. Rate the difficulty of each problem and your confidence in solving similar problems in the future.

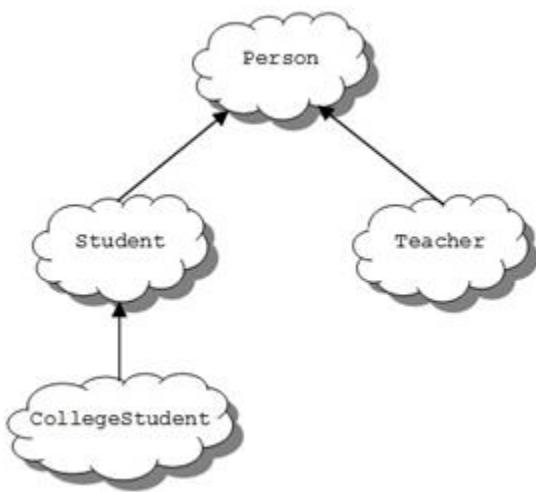
## **Submission Instructions:**

1. Create a public GitHub repository and share your work, Just include the link to the public repo in your submission
2. Include a Word document with:
  - o Screenshots of outputs
  - o Your responses to the questions in the Instructions section
3. Submit the GitHub repository link

## 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);

```

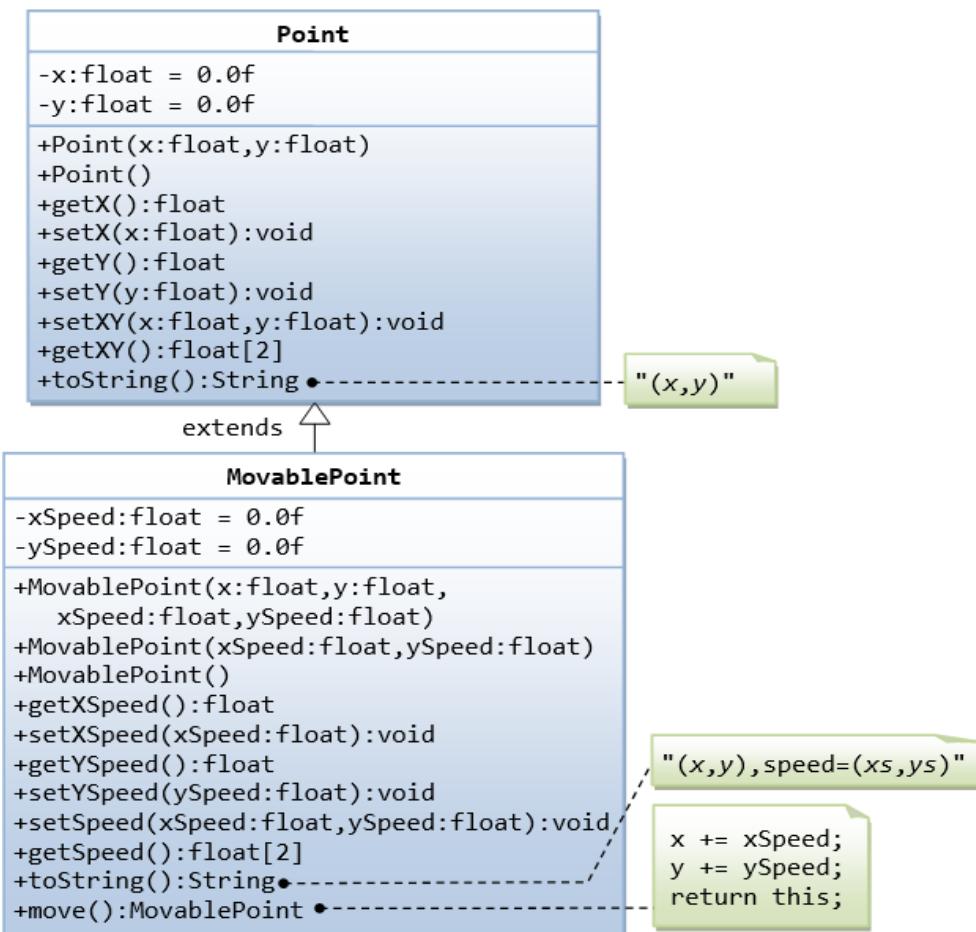
### **Deliverables:**

**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:**

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**

## Problem#3:

### Abstract Classes

Consider the following shapes; Ellipse, Circle, Triangle, EquilateralTriangle. Each shape should have a name, a method to compute its perimeter, and another method to compute its area. The name should be an instance variable of type String. Design your inheritance hierarchy with the common features in the **Abstract** superclass Shape. Notice that the area and perimeter are common to all Shapes, but we don't know how to compute the area or perimeter for a general shape.

The ellipse class has a major and minor axes a and b, respectively. The constructor should assign the largest value to a and smallest to b. The area and perimeters of an ellipse are:

$$\text{Perimeter} = P = \pi \sqrt{2(a^2 + b^2) - (a - b)^2/2} \quad [\text{Note that if } a = b = r, \text{ then } P = 2\pi r]$$
$$\text{Area} = A = \pi ab$$

The Triangle class has three instance variables side1, side2, and side3. The formula for the area and perimeter of a general Triangle with sides A, B, and C is given by.

$$AREA = \sqrt{S(S - A)(S - B)(S - C)}$$

$$PERIMETER = A + B + C$$

$$S = \frac{A + B + C}{2}$$

The condition for any three positive values to make sides of a Triangle is:

side1+side2>side3 and side2+side3>side1 and side3+side1>side2

You need to check this condition inside the constructor. If it is not satisfied, print an error message and terminate the program, otherwise make your Triangle object.

The three sides of the equilateral triangle are equal.

Make a Test class where you make objects from the different classes and store them in an array of type Shape. Then, make a loop and print the objects name, area, and perimeter through `toString` i.e. you need to override `toString` in the Shape class only.

**Deliverables:**

**Complete and working-class files with proper comments.**

1. `Shape.java`
2. `Circle.java`
3. `Ellipse.java`
4. `Triangle.java`
5. `EquilateralTriangle.java`
6. `Demo.java`
7. **Screenshot of the running code's output**

## **Problem#4:**

### **Interfaces**

Some OOP languages such as C++ allow a sub-class to inherit from more than one super class (multiple inheritance). While this has some advantages, it makes such languages complex. To avoid such complexities, Java does not allow for multiple inheritance. However, a lot of the advantages of multiple inheritance can be achieved using **Interfaces**.

An interface is like a class but with the following restrictions:

- All methods are implicitly **abstract** and **public**
- An interface cannot have instance variables. However, an Interface may have constants (final variables) and these are implicitly public and static. Also, they are inherited by any class that implements the interface.
- An Interface can extend another interface and it is implemented by a class using the **implements** keyword. In fact, a class may implement any number of interfaces.

Consider an interface `Scalable` with a method `scale` of type `void`. It takes the scaling factor as a parameter. Make the shape class defined above implement the `Scalable` interface. Note that since `Shape` is abstract, it does not have to implement `scale` method.

Make the appropriate subclasses override `scale` method by multiplying their instance variables by the scale factor.

Modify the above `Test` class so that you add a static method that receives an array of Type `Scalable`, and a scale factor. This method should visit all the elements of the

Scalable array and call the scale method with the scale factor passed to the static method. You should print your objects before and after scaling.

**Deliverables:**

**Complete and working-class files with proper comments.**

1. Shape.java
  2. Circle.java
  3. Ellipse.java
  4. Triangle.java
  5. EquilateralTriangle.java
  6. Scalable.java
  7. Demo.java
  8. Screenshot of the running code's output
-