Software Developer Course Assessment

Quantitative Assessment Practice

Course Name: Advanced Programming (Java)
Date: 27th September 2024
Submission date: 8th October 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.**

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.

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.)

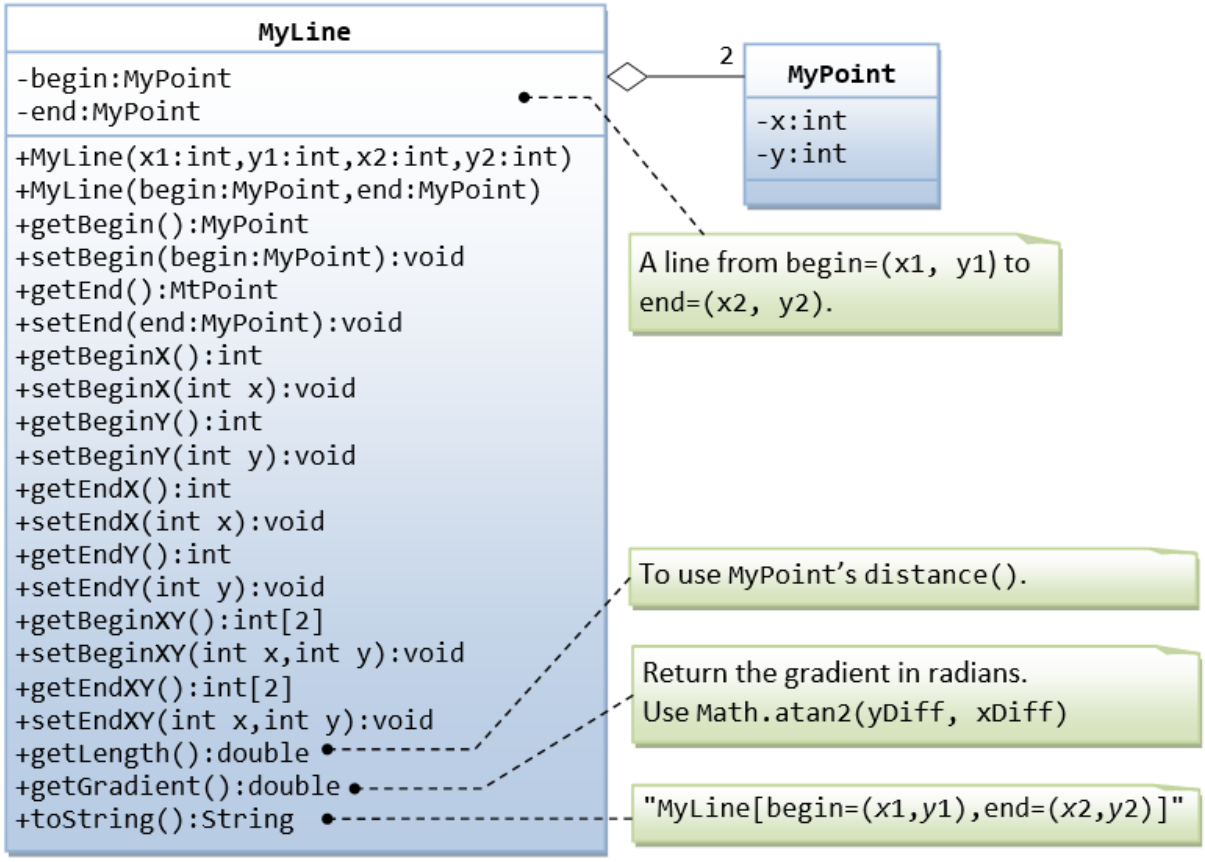
4. Did you need to ask questions to any of your instructors? If so, how many questions did you ask (or how many help sessions did you require)?

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:

The MyLine and MyPoint Classes

A class called MyLine, which models a line with a begin point at (x1, y1) and an end point at (x2, y2), is designed as shown in the class diagram. The MyLine class uses two MyPoint instances (written in the earlier exercise) as its begin and end points. Write the MyLine class. Also write a test driver to test all the public methods in the MyLine class.



Deliverable#1:

Complete and working-class files with proper comments.

1. *MyPoint.java*
2. *MyLine.java*
3. *TesMyLine.java*
4. *Screenshots of the running code's output*

Problem#2

The MyRectangle and MyPoint Classes

Design a MyRectangle class which is composed of two MyPoint instances as its top-left and bottom-right corners. Draw the class diagrams, write the codes, and write the test drivers.

Deliverable#2:

Complete and working-class files with proper comments.

5. MyRectangle.java
6. TestMyRectangle.java
7. Screenshots of the running code's output

Problem#3:

Introduction

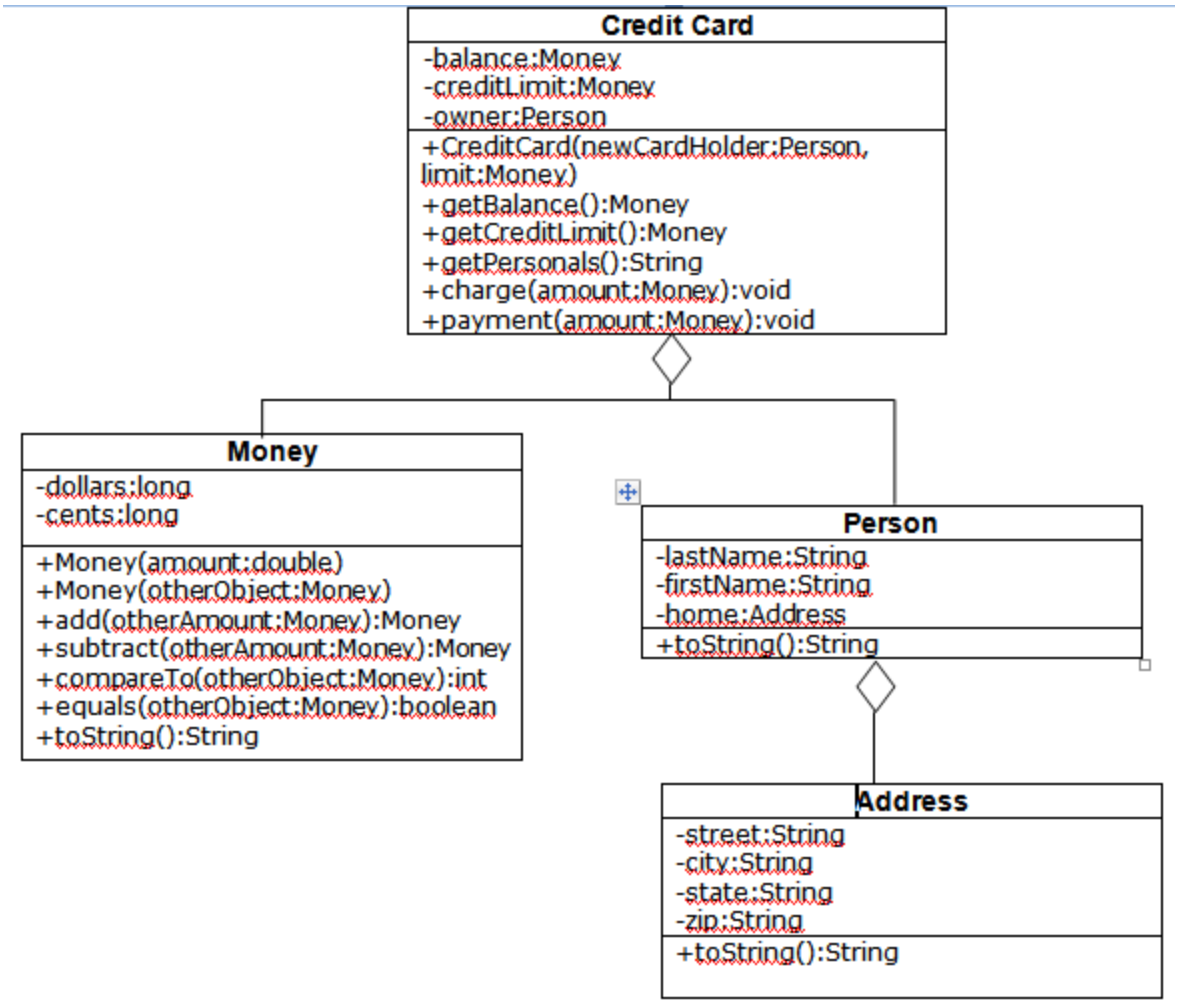
This time, the object that we are choosing is more complicated. It is made up of other objects. This is called **aggregation**. A credit card is an object that is very common, but not simple. Attributes of the credit card include information about the owner, as well as a balance and credit limit. These things would be our instance fields. A credit card allows you to make payments and charges. These would be methods. As we have seen before, there would also be other methods associated with this object in order to construct the object and access its fields.

Examine the UML diagram that follows. Notice that the instance fields in the CreditCard class are other types of objects, a Person object or a Money object. We can say that the CreditCard "has a" Person, which means aggregation, and the Person object "has a" Address object as one of its instance fields. This aggregation structure can create a very complicated object. I will try to keep this lab reasonably simple.

To start with, we will write the Class Money. We will investigate overloading methods by writing another constructor method. The constructor that you will be writing is a copy constructor. This means it should create a new object, but with the same values in the instance variables as the object that is being copied.

Next, we will write equals and toString methods. These are very common methods that are needed when you write a class to model an object. You will also see a compareTo method that is also a common method for objects. Search online to look for Equals & CompareTo Methods.

After we have finished the Money class, we will write a CreditCard class. This class contains Money objects, so you will use the methods that you have written to complete the Money class. The CreditCard class will explore passing objects and the possible security problems associated with it. We will use the copy constructor we wrote for the Money class to create new objects with the same information to return to the user through the accessor methods.



Task #1 Overloading by Writing a Copy Constructor

Overload the constructor. The constructor that you will write will be a copy constructor. It should use the parameter money object to make a duplicate money object, by copying the value of each instance variable from the parameter object to the instance variable of the new object.

Task #2 Passing and Returning Objects

1. Create a CreditCard class according to the UML Diagram on the back. It should have data fields that include an owner of type Person, a balance of type Money, and a creditLimit of type Money.
2. It should have a constructor that has two parameters, a Person to initialize the owner and a Money value to initialize the creditLimit. The balance can be initialized to a Money value of zero. Remember you are passing in objects (pass by reference), so you have passed in the address to an object. If you want your

CreditCard to have its own creditLimit and balance, you should create a new object of each using the copy constructor in the Money class.

3. It should have accessor methods to get the balance and the available credit. Since these are objects (pass by reference), we don't want to create an insecure credit card by passing out addresses to components in our credit card, so we must return a new object with the same values. Again, use the copy constructor

to create a new object of type money that can be returned.

4. It should have an accessor method to get the information about the owner, but in the form of a String that can be printed out. This can be done by calling the toString method for the owner (who is a Person).
5. It should have a method that will charge to the credit card by adding the amount of Money in the parameter to the balance if it will not exceed the credit limit. If the credit limit will be exceeded, the amount should not be added, and an error message can be printed to the console.
6. It should have a method that will make a payment on the credit card by subtracting the amount of Money in the parameter from the balance.
7. Compile, debug, and test it out completely by running CreditCardDemo.java.

Address.java

```
//The code must be inserted ahead of the call to the content
// Defines an address using a street, city, state, and zipcode
```

Person.java

```
// Defines a person by name and address
```

CreditCardDemo.java

```
// Demonstrates the CreditCard class
public class CreditCardDemo
{
public static void main (String [] args)
{
final Money LIMIT = new Money(1000);
final Money FIRST_AMOUNT = new Money(200);
final Money SECOND_AMOUNT = new Money(10.02);
final Money THIRD_AMOUNT = new Money(25);
final Money FOURTH_AMOUNT = new Money(990);
Person owner = new Person(“Christie”, “Diane”,
new Address(“237J Harvey Hall”,”Menomonie”,
“WI” , “54751”));
CreditCard visa = new CreditCard(owner, LIMIT);
System.out.println(visa.getPersonals());
System.out.println(“Balance: ” + visa.getBalance());
System.out.println(“Credit Limit : ” + visa.getCreditLimit());
System.out.println();
System.out.println(“Attempt to charge ” + FIRST_AMOUNT);

visa.charge(FIRST_AMOUNT);
System.out.println(“Balance : ” + visa.getBalance());
System.out.println(“Attempt to charge ” + SECOND_AMOUNT);
visa.charge(SECOND_AMOUNT);
System.out.println(“Balance : ” + visa.getBalance());
System.out.println(“Attempt to pay ” + THIRD_AMOUNT);
visa.charge(THIRD_AMOUNT);
System.out.println(“Balance : ” + visa.getBalance());
System.out.println(“Attempt to charge ” + FOURTH_AMOUNT);
visa.charge(FOURTH_AMOUNT);
System.out.println(“Balance : ” + visa.getBalance());
}
}
```

You should get the output:

```
Diane Christie, 237J Harvey Hall, Menomonie, WI
54751
Balance: $0.00
Credit Limit: $1000.00
Attempt to charge $200.00
Charge: $200.00
Balance: $200.00
Attempt to charge $10.02
Charge: $10.02
Balance: $210.02
Attempt to pay $25.00
Payment: $25.00
Balance: $185.02
Attempt to charge $990.00
Exceeds credit limit
Balance: $185.02
```

Deliverable#3:

Complete and working-class files with proper comments.

1. *CreditCard.java*
2. *Money.java*
3. *Address.java*
4. *Person.java*
5. *CreditCardDemo.java*
6. *Screenshot of the running code's output*

Submission:

Please create a public github repository and upload all the java files, screen shots and feedback questions. Submit the link to that repository.