**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.)

*[It took me around 12 hours to complete. The first problem was done quickly, for the second problem I could not figure out for a long time where to start and what exactly to include. The third problem was the hardest, it took me a long time to figure out how to write Money and Credit Card classes]*

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

*[Used a lot of different websites to find a way to write the third problem easier and so I really could understand what I am writing]*

1. 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.)

*[I think I could ask someone for help and would done it quicker, but I understand better and will remember it for longer if I solve it alone]*

1. 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)?

*[No, I did not. I only asked for extension, because I was really sick and could not complete it]*

1. 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.

*[Without a third problem I will say that it is easy, and I will be able easily to complete it again. But I am not sure about third problem yet. I think I will struggle with it in the future again]*

# **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.

Graphical user interface, text, application

Description automatically generated

**Deliverable#1:**

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

1. *MyPoint.java*

package MyLine\_MyPoint;

public class MyPoint {

private int x;

private int y;

// Constructor

public MyPoint(int x, int y) {

this.x = x;

this.y = y;

}

// Getters and Setters

public int getX() {

return x;

}

public int getY() {

return y;

}

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

// Method to calculate distance to another MyPoint

public double distance(MyPoint other) {

return Math.hypot(other.x - this.x, other.y - this.y);

}

// To String/Overide

public String toString() {

return "MyPoint[" + "x=" + x + ", y=" + y + "]";

}

}

1. *MyLine.java*

package MyLine\_MyPoint;

public class MyLine {

private MyPoint begin;

private MyPoint end;

// Constructors

public MyLine(int x1, int y1, int x2, int y2) {

this.begin = new MyPoint(x1, y1);

this.end = new MyPoint(x2, y2);

}

public MyLine(MyPoint begin, MyPoint end) {

this.begin = begin;

this.end = end;

}

// Getters and Setters

public MyPoint getBegin() {

return begin;

}

public void setEnd(MyPoint end) {

this.end = end;

}

public MyPoint getEnd() {

return end;

}

public void setBegin(MyPoint begin) {

this.begin = begin;

}

public int getBeginX() {

return begin.getX();

}

public void setBeginX(int x) {

begin.setX(x);

}

public int getBeginY() {

return begin.getY();

}

public void setBeginY(int y) {

begin.setY(y);

}

public int getEndX() {

return end.getX();

}

public void setEndX(int x) {

end.setX(x);

}

public int getEndY() {

return end.getY();

}

public void setEndY(int y) {

end.setY(y);

}

public int[] getBeginXY() {

return new int[]{begin.getX(), begin.getY()};

}

public void setBeginXY(int x, int y) {

begin.setX(x);

begin.setY(y);

}

public int[] getEndXY() {

return new int[]{end.getX(), end.getY()};

}

public void setEndXY(int x, int y) {

end.setX(x);

end.setY(y);

}

// Get Length and Gradient

public double getLength() {

return begin.distance(end);

}

public double getGradient() {

return Math.atan2(end.getY() - begin.getY(), end.getX() - begin.getX());

}

// To String/Overide

public String toString() {

return "MyLine[begin=(" + begin.getX() + "," + begin.getY() + "), end=(" + end.getX() + "," + end.getY() + ")]";

}

}

1. *TesMyLine.java*

package MyLine\_MyPoint;

public class TestMyLine {

public static void main(String[] args) {

// Create points

MyPoint point1 = new MyPoint(1, 2);

MyPoint point2 = new MyPoint(4, 6);

// Create MyLine instance

MyLine line = new MyLine(point1, point2);

System.out.println();

// Test getBegin and getEnd

System.out.println("Begin Point: " + line.getBegin());

System.out.println("End Point: " + line.getEnd());

// Test getLength

System.out.println("Length of the line: " + line.getLength());

// Test getGradient

System.out.println("Gradient of the line: " + line.getGradient());

// toString

System.out.println(line.toString());

// Test set and get methods

line.setBeginXY(2, 3);

line.setEndXY(5, 7);

System.out.println();

System.out.println("After modifying:");

System.out.println("Begin Point: " + line.getBegin());

System.out.println("End Point: " + line.getEnd());

System.out.println("Length of the line: " + line.getLength());

System.out.println("Gradient of the line: " + line.getGradient());

// toString

System.out.println(line.toString());

System.out.println();

}

}

1. *Screenshots of the running code’s output*

*Begin Point: MyPoint[x=1, y=2]*

*End Point: MyPoint[x=4, y=6]*

*Length of the line: 5.0*

*Gradient of the line: 0.9272952180016122*

*MyLine[begin=(1,2), end=(4,6)]*

*After modifying:*

*Begin Point: MyPoint[x=2, y=3]*

*End Point: MyPoint[x=5, y=7]*

*Length of the line: 5.0*

*Gradient of the line: 0.9272952180016122*

*MyLine[begin=(2,3), end=(5,7)]*

# **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.

1. MyRectangle.java

package MyRectangle\_MyPoint;

public class MyRectangle {

private MyPoint topLeft;

private MyPoint bottomRight;

public MyRectangle(MyPoint topLeft, MyPoint bottomRight) {

this.topLeft = topLeft;

this.bottomRight = bottomRight;

}

// To calculate Area and Perimeter get width and height

public int getWidth() {

return bottomRight.getX() - topLeft.getX();

}

public int getHeight() {

return bottomRight.getY() - topLeft.getY();

}

// Calculate and get Area and Perimeter

public int getArea() {

return getWidth() \* getHeight();

}

public int getPerimeter() {

return 2 \* (getWidth() + getHeight());

}

// To String Overide

public String toString() {

return "Rectangle [Top Left: " + topLeft + ", Bottom Right: " + bottomRight + "]";

}

}

1. TestMyRectangle.java

package MyRectangle\_MyPoint;

public class TestMyRectangle {

public static void main(String[] args) {

// Create points for the rectangle

MyPoint topLeft = new MyPoint(1, 4);

MyPoint bottomRight = new MyPoint(5, 1);

// Create a rectangle using the points

MyRectangle rectangle = new MyRectangle(topLeft, bottomRight);

// Display the rectangle details

System.out.println(rectangle);

System.out.println("Width: " + rectangle.getWidth());

System.out.println("Height: " + rectangle.getHeight());

System.out.println("Area: " + rectangle.getArea());

System.out.println("Perimeter: " + rectangle.getPerimeter());

}

}

1. Screenshots of the running code’s output

Rectangle [Top Left: (1, 4), Bottom Right: (5, 1)]

Width: 4

Height: -3

Area: -12

Perimeter: 2

**Problem#3:**

**Introduction**

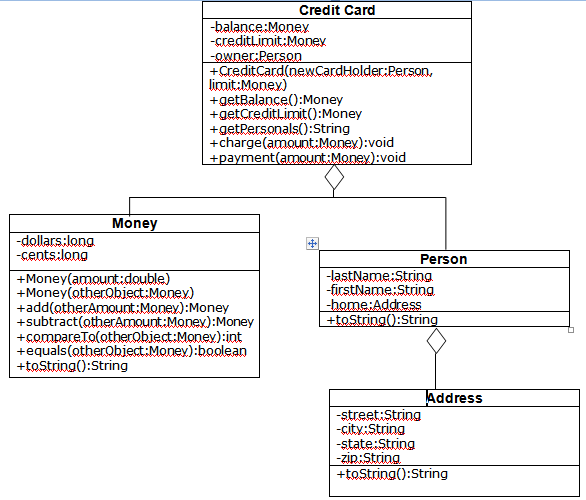
This time, the object that we are choosing is more complicated. It is made up of other objects. This is called **aggregation**. A credit card is an object that is very common, but not simple. Attributes of the credit card include information about the owner, as well as a balance and credit limit. These 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.payment(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*

package CreditCardProblem;

public class CreditCard {

private Money balance;

private Money creditLimit;

private Person owner;

public CreditCard(Person newCardHolder, Money limit) {

this.balance = new Money(0.0); // Initial balance is 0

this.creditLimit = new Money(limit); // Use copy constructor for credit limit

this.owner = newCardHolder; // Assign owner

}

// Get information

public Money getBalance() {

return new Money(balance);

}

public Money getCreditLimit() {

return new Money(creditLimit);

}

public String getPersonals() {

return owner.toString();

}

public void charge(Money amount) {

if (balance.add(amount).compareTo(creditLimit) <= 0) {

balance = balance.add(amount);

System.out.println("Charge: " + amount);

} else {

System.out.println("Exceeds credit limit");

}

}

public void payment(Money amount) {

balance = balance.subtract(amount);

System.out.println("Payment: " + amount);

}

}

1. *Money.java*

package CreditCardProblem;

public class Money {

private long dollars;

private long cents;

// Constructors

public Money(double amount) {

this.dollars = (long) amount;

this.cents = Math.round((amount - this.dollars) \* 100);

}

public Money(Money other) {

this.dollars = other.dollars;

this.cents = other.cents;

}

// Add, Substract this.amount from or to other.Amount

public Money add(Money otherAmount) {

long totalCents = (this.dollars \* 100 + this.cents) + (otherAmount.dollars \* 100 + otherAmount.cents);

return new Money(totalCents / 100.0);

}

public Money subtract(Money otherAmount) {

long totalCents = (this.dollars \* 100 + this.cents) - (otherAmount.dollars \* 100 + otherAmount.cents);

return new Money(totalCents / 100.0);

}

public int compareTo(Money otherObject) {

long thisTotalCents = this.dollars \* 100 + this.cents;

long otherTotalCents = otherObject.dollars \* 100 + otherObject.cents;

return Long.compare(thisTotalCents, otherTotalCents);

}

public boolean equals(Money other) {

return this.dollars == other.dollars && this.cents == other.cents;

}

// To String/Overide

public String toString() {

return String.format("$%d.%02d", dollars, cents);

}

}

1. *Address.java*

package CreditCardProblem;

public class Address {

private String street;

private String city;

private String state;

private String zip;

// Constructor

public Address(String street, String city, String state, String zip) {

this.street = street;

this.city = city;

this.state = state;

this.zip = zip;

}

// To String/Overide

public String toString() {

return "Address: " + street + ", " + city + ", " + state + " " + zip;

}

}

1. *Person.java*

package CreditCardProblem;

public class Person {

private String firstName;

private String lastName;

private Address home;

// Constructor

public Person(String firstName, String lastName, Address home) {

this.firstName = firstName;

this.lastName = lastName;

this.home = home;

}

// To String/Overide

public String toString() {

return "Person: " + lastName + " " + firstName + ", " + home.toString();

}

}

1. *CreditCardDemo.java*

package CreditCardProblem;

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();

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.payment(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());

System.out.println();

}

}

1. *Screenshot of the running code’s output*

*Person: Diane Christie, Address: 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*

# Submission:

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