CS 5000 – Summer 2025
Assignment #8, 50 Points

Objects and Classes – Chapter 9

A note for good coding practices: Starting with this assignment, you need to incorporate the following concepts/practices in your code as required by the problem statement. Do not expect them to be spelled out in the problem statements or subsequent assignments. This and subsequent assignments will be assessed for utilization of these concepts in the code.

- Apply the concept of encapsulation.
- Always declare variables as private.
- Provide setters and getters methods to allow access to class variables as per the problem statement requirements and the role/purpose of each variable in the solution.
- Always declare support methods as private.
- Allow the user to enter input values to facilitate code test-ability. DO not hard code inputs unless stated in the problem statement.
- Allow the user to re-run the code in the same session using a sentinel loop.
- Always sanitize/validate input values as soon as being entered to make sure they are "good" values (using assertions or if statements).
- Always have the main/test/driver program in a sperate file than the class to solve the problem at hand.

Develop a complete Java program for each of the following problems. Please name the programs as indicated, add proper program headers, and output labels as shown below. ***Please use only concepts and programming constructs/syntax we discuss to date.***

Program #1 (25 points): Design and implement a Java class, named ***`Account`***. The class defines the following data fields and methods:

1. Private `int` data field named `id` to store the account ID (default value is 0).
2. Private `double` data field named `balance` to store the account balance (default value is 0.0).
3. Private `double` data field named `annualInterestRate` to store the interest rate (default value is 0.0%). (Assume all accounts have the same interest rate. The annual interest rate is percentage such as 3.2%, thus you need to divide by 100 to get double value 0.032).
4. Private `Date` data field named `dateCreated` (an object of class Date) to store the date when the account was created.
5. Non-argument constructor method that creates a default account (with default values).
6. Constructor method that creates an account with specified ID and initial balance.
7. Get and Set methods for variables `id`, `balance`, and `annualInterestRate`.
8. Get method for variable `dateCreated`.
9. Method named `getMonthlyInterestRate()` that returns the monthly interest rate as double value (i.e., `annualInterestRate / 12`). The monthly interest rate is formatted as percentage (%) when displayed.
10. Method named `getMonthlyInterest()` that returns the earned monthly interest amount as double value (i.e., `balance * monthlyInterestRate)`. The monthly interest amount is formatted as currency ($) when displayed.
11. Method named `withdraw()` that withdraws a specific amount from the account.
12. Method named `deposit()` that deposits a specific amount to the account.

In a separate file, write a test program (named ***`TestAccount`***) to create an account object named `myObject` as follows:

- Account ID is 123456; Initial balance is $10,000, annual interest rate is 2.5%.
- Withdraw $3,500
- Deposit $500
- Print out the account balance
- Print out the earned monthly interest
- Print out the date the account was created

Now, add method `toString()` to class `Account` to allow the user to printout a meaningful description of an account object using all of its instance variables. For example, the following statement in the test program

```
System.out.print(myObject);
```

on object `myObject` would display the account information as follows. Make sure your code displays the outputs following the test data format.

```
Account ID:             123456
Account Balance:        $7,000.00
Annual Interest Rate:   2.5%
Monthly Interest:       $14.58
Date Opened:            Wed Jul 06 10:35:04 EDT 2022
```

Here is an example of `toString()` method for a student object:

```
//-------------------------------------------------------------------------
// Returns a string representation of student object using name and studentID.
//-------------------------------------------------------------------------
public String toString(){
 // name and studentID are instance variables in class Student
 return ("The student name is " + name + ", and the ID is " + studentID);
}
```

Now, modify the test program above to create 2 more account objects (say `myAccount` and `yourAccount`) with different initial balance values and different interest rates. **Allow the user to enter the object values**. Test all class methods on at least one object in a logical order and display meaningful information about the object after each method call. Use a sentinel loop to allow re-runs and re-create those objects with different values.

Handle account overdraft issue with proper error message.

Document your code and organize and space the outputs properly. Use escape characters and formatting objects ($ and %) as needed.

Note: To show the fraction part of the Interest rate value as `2.5%`, you need to set the fraction digits of the formatting object fmt as follows:

```
fmt.setMinimumFractionDigits(1);
```

Here is an example:

```
import java.text.NumberFormat;
. . .
Double rate = 0.05275;  //rate is 5.275%
NumberFormat fmt = NumberFormat.getPercentInstance();
fmt.setMaximumFractionDigits(1);
System.out.println(fmt.format(rate));
```

Program #2 (25 points): Design and implement a Java class, named ***`Rectangle`***. The class defines the following data fields and methods:

1. Private `double` data field named `width` to store the rectangle width (default value is 1.00).
2. Private `double` data field named `height` to store the rectangle height (default value is 1.00).
3. Non-argument constructor method that creates a default rectangle (with default values).
4. Constructor method that creates a rectangle with specified width and height values.
5. Get methods for the data fields `width` and `height`.
6. Method named `getArea()` that returns the area of the rectangle.
7. Method named `getPerimeter()` that returns the perimeter of the rectangle.
8. Method `toString(String objectName)` to printout a meaningful description of a rectangle object. Assuming rectangle ABC is15.00 units wide and 20.00 units high, calling `toString("ABC")` would display the following output:)

```
Rectangle ABC is 15.0 units wide and 20.0 units high.
```

In a separate file, write a test program (named ***TestRectangle***) to create 3 rectangle objects named `myRctangle`, `hisRectangle`, and `herRectangle` as follows:

- `myRectangle` is a default object. (i.e., uses default width and height)
- `hisRectangle` has width 5.0 and height 10.0 (these are just examples, read values from the user)
- `herRectangle` has width 5.75 and height 12.50 (these are just examples, read values from the user)

Handle negative inputs with proper errors messages and allow user to re-enter incorrect input value. Using proper methods, display the width, height, area, and perimeter for each object following these examples (format output values in 2 decimal places).

Test data:

```
myRectangle:
------------
Width:      1.00
Height:     1.00
Area:       1.00
Perimeter:  4.00

hisRectangle:
------------
Width:      5.00
Height:     10.00
Area:       50.00
Perimeter:  30.00

herRectangle:
------------
Width:      3.00
Height:     4.00
Area:       12.00
Perimeter:  14.00
```

Using method `toString`, display a meaningful description for each rectangle object. For example,

```
Rectangle myRectangle is 1.0 unit wide and 1.0 unit high.

Rectangle hisRectangle is 5.0 units wide and 10.0 units high.
```

Allow the user to enter the object values. Test all class methods on at least one object in a logical order and display meaningful information about the object after each method call. Use a sentinel loop to allow re-runs and re-create objects `hisRectangle` and `herRectangle` using different input values.

Document your code and organize and space the outputs properly. Use escape characters and formatting objects ($ and %) as needed. Make sure your code displays the outputs following the test data format.

Submission:

1. Before submitting your programs, make sure you review the assignment submission requirements and grading guidelines posted in D2L. The grading guidelines explain some of the common errors found in programming assignments.
2. The assignment due date is posted in D2L.
3. Please compile, run, and test your code right before you upload your java files to the assignment submission folder in D2L.
4. Please upload only the .java files (total 4 files).