CSCI 1301: Introduction to Computing and Programming

Project 1: Credit Card Pay Off

Introduction

This project will allow you to apply your knowledge of variables, assignments, expressions, type casting, input, output, and basic algorithm design. The program can be completed using variable assignment statements as well as input and output statements. You are free, however, to use any other features of Java.

You will write a Java application to compute and display the number of months needed to pay off credit card debt. Your program will prompt the user for the following three things:

- the *principal*, which is the amount of money owed on the credit card;
- the *annual interest rate*;
- the *monthly payment*, the amount the user plans to pay the credit card company each month.

Based on these inputs, the program will compute the number of months required to pay the debt. It will also compute the total amount paid to the credit card company after all payments are made, the total the amount of interest paid, and the overpayment.

The number of months needed to pay off the debt can be calculated using the following formula.

$$\frac{ln(monthlyPayment) - ln\left(monthlyPayment - \frac{annualInterestRate}{1200.0} \times principal\right)}{ln\left(\frac{annualInterestRate}{1200.0} + 1.0\right)}$$

In the formula, *ln*(x) is the natural logarithm of a real number x > 0.

The total amount paid to the credit card company is the *ceiling* of the number of months needed to pay off the debt multiplied by the monthly payment. The total interest paid is simply the principal amount (which is provided by the user) subtracted from the total amount paid to the credit card company. The overpayment is the amount overpaid to the credit card company, which is an algorithm you'll have to figure out. Details about these computations are given in the Program Requirements section.

The following is a sample run, and the input (**5000.00**, **15.0**, **100.00**—shown in red for demonstration purposes only) and output of your submitted program should be formatted the same way when run in Eclipse's console or on a command line. Additional sample runs are included at the end of this document.

```
Principal:                    5000.00
Annual Interest Rate (%):     15.0
Monthly Payment:              100.00

Months Needed To Pay Off:     79
Total Amount Paid:            $7900.00
Total Interest Paid:          $2900.00
Overpayment:                  $4.43
```

Program Requirements

The instructions below must be followed in order for full credit to be awarded. Following the instructions is a vital part to this and all programming projects.

1. The name of the program file must be **`PayoffDebt.java`**.
2. The class name in the file must be **`PayoffDebt`** (in Java, the class name and **`.java`** file name must match).
3. The program must read inputs and display the results in **exactly** as indicated in the examples.
4. The annual interest rate should be entered as a percent as shown in the examples.
5. The Months Needed To Pay Off must be stored with two different variables. One variable should store the raw value as a **`double`** computed by the formula above. The other variable should be stored and written to output as a 32-bit integer, and it should contain the *ceiling* of the raw value (the smallest integer greater than or equal to it). For instance, the integer ceiling of 9.2 is 10.
6. All other numeric variables must be **`double`** data types.
7. You must figure out the algorithm to compute the overpayment based on the examples given at the end of this project. Hint: It uses both the floating point and integer values of The Months Needed To Pay Off. Write its algorithm in pseudocode in the comments of your project directly above the line(s) of source code that computes the overpayment.
8. The Total Amount Paid, Total Interest Paid, and Overpayment output must be formatted like units of currency. They must have a $ in front of their values, and their values must show only two digits after the decimal point. An example of formatting output can be done with the **`printf`** method, and examples of this are shown in the 8th edition of the course textbook. You can also round to two decimal places by using mathematical operations (divide, type casting, and multiply). Your answer must be within .01 of the answer to receive full credit.
9. You must include the Statement of Academic Honesty comment at the top of the program's source file. Copy and agree to the entirety of the text contained in the file **StatementOfAcademicHonesty.txt** on the Labs and Projects webpage, and fill in the class name of your **`.java`** file, your name, submission date, and the program's purpose. In the future, every Java source file of every project you submit **must** have a comment such as this at the top of every source file. Otherwise, points will be deducted from your project.

When writing your program, you may safely assume that all input to the program is valid. That is, you do not need to write code to handle erroneous or ill-formed data entered by the user.

Additional Requirements

These are things that make the graders lives easier, and ultimately, you will see in the real world as well. Remember the teaching staff does not want to touch your code after they gave you

requirements; they want to see the perfect results they asked for! Here is a checklist of things you can **lose points** for:

- (10 points) If the source file(s)/class(es) are named incorrectly (case matters!)
- (10 points) If your source file(s) have a package declaration at the top
- (10 points) If any source file you submit is missing your Statement of Academic Honesty at the top of the source file. All submitted source code files must contain your Statement of Academic Honesty at the top of each file.
- (10 points) If you have more than one instance of Scanner in your program. Your program should only have one instance of Scanner.
- (15 points) Inconsistent I/O (input/output) that does not match our instructions or examples exactly (unless otherwise stated in an assignment's instructions). Your program's I/O (order, wording, formatting, etc.) must match our examples and instructions.
- (100 points) If the source file(s) are not submitted before the specified deadline's late period ends (48 hours after the deadline) or if they do not compile.
- (25 points) Late penalties will be deducted as per the course syllabus.
- If your (10 points) comments or (10 points) variables are "lacking"
    - Here, "lacking" means that you or a TA can find **any** lines of code or variables that take more than 10 seconds to understand, and there is no comment, or the variable name does not make sense (variable names like **b**, **bb**, **bbb**, etc. **will almost never be acceptable)**
- (10 points) Indentation is not consistent throughout your source code
    - Refresh your memory of indentation patterns in chapter 2 in the course textbook
    - Be careful of a combination of tabs and spaces in your files (use one or the other)!

If any of the above do not make sense to you, talk to a TA, or ask on Piazza!

Hints

- Aligning the input prompts and the output values will require the use of escape sequences. It is recommended that you use tabs ("\t") rather than spaces.
- Java's JDK provides a class **`Math`** which is always available which contains a method called "**`ceil`**" which will return the ceiling of a given number passed as an argument. The returned value will be returned as a **`double`**. For example,

```
double a = Math.ceil(3.4576);      // Math.ceil returns 4.00
double b = Math.ceil(4);           // Math.ceil returns 4.00
```

- The JDK's Math class also contains a method called **`log`**, which returns the natural logarithm, *ln*, of its argument. Again, the returned value has type **`double`**. For example,

```
double c = Math.log(12.7);         //Math.log returns
                                   //the ln(12.7)≈2.54160199
```

Project Submission

- Submit the file **`PayoffDebt.java`** and only that file via eLC.

Project Grading

All projects are graded out of a possible 100 points. Programs can be submitted up to 48 hours late, but late programs will lose 25 points on the first two days late (0 to 48 hours after its deadline). Programs not submitted within 48 hours after the deadline will receive a grade of 0. Programs that do not compile will receive a grade of zero. You must make absolutely certain your program compiles before submitting, and you must thoroughly test your program with many different inputs to verify that it is working correctly. This project will be graded for correctness and adherence to all project instructions and requirements.

Examples

Your program should work correctly on the examples below. It should prompt the user for information (shown in red for demonstration purposes only – yours will not be red), and produce calculated values matching those shown below. All input and output should be formatted as shown when run in Eclipse's console or in the command line.

```
Principal:                    5000.00
Annual Interest Rate (%):     15.0
Monthly Payment:              100.00

Months Needed To Pay Off:     79
Total Amount Paid:            $7900.00
Total Interest Paid:          $2900.00
Overpayment:                  $4.43
```

```
Principal:                    10250.50
Annual Interest Rate (%):     17.25
Monthly Payment:              550.75

Months Needed To Pay Off:     22
Total Amount Paid:            $12116.50
Total Interest Paid:          $1866.00
Overpayment:                  $102.02
```

```
Principal:                    3557.27
Annual Interest Rate (%):     5.74
Monthly Payment:              275.99

Months Needed To Pay Off:     14
Total Amount Paid:            $3863.86
```

```
Total Interest Paid:          $306.59
Overpayment:                  $183.43
```

```
Principal:                    8556.74
Annual Interest Rate (%):     29.95
Monthly Payment:              316.78

Months Needed To Pay Off:     46
Total Amount Paid:            $14571.88
Total Interest Paid:          $6015.14
Overpayment:                  $162.25
```