Programming Assignment 3 (IntelliJ)

Due Oct 19, 2019 by 11:59pm **Points** 50 **Submitting** a file upload **File Types** zip

This assignment must be done using you IDEA IntelliJ Community Edition: (**https://www.jetbrains.com/idea/download/** (https://www.jetbrains.com/idea/download/)). You will need to install and configure the software before starting. Instructions for how to do this, and tutorials for the software, can be found on the IntelliJ website (**https://www.jetbrains.com/idea/documentation/** (https://www.jetbrains.com/idea/documentation/)).

Programming Assignment 3 will consist of writing multiple short programs. You will write these programs from scratch, using the concepts covered so far.

Each problem below should be self contained within its own IntelliJ project. You will upload these projects to your GitHub site. Each project must contain the java source code, the project files created by IntelliJ, *and* the .class files generated when you compile your program. To include the .class files, you may need to modify your .gitignore file. All three problems should be in the *same* repository. To keep things simple, I suggest that you create a new repository on GitHub called **<last_name>_PA3** and push your projects there (instead of the same repo you used for Assignments 1 and 2).

You will submit your assignment as *both* a link to your GitHub page and an upload of your projects as a *.zip file. Your **projects should be named <last_name>_pN**, where N is the problem number the corresponds to each project. You must **name your zip file <last_name>_PA3.zip**. Failure to adhere to these naming convention may result in your assignment going ungraded. Due to Webcourse@UCF limitations, when you submit your assignment, submit the zip file. Add your GitHub URL as a note during the submission, or as a comment after the submission. If you do not know how to create a zip file, refer to Google (for windows, just look **on microsoft's website** (https://support.microsoft.com/en-us/help/4028088/windows-zip-and-unzip-files)).

If you forget to submit one or both of the required items on time, additional submissions after the deadline will be considered late for the purposes of calculating your grade.

An example structure of your GitHub repository might be:

```
hollander_PA3
  |--- hollander_p1
  |--- hollander_p2
  |--- hollander_p3
```

Grades for this program will be determined according to the rubic. If your project on GitHub does not contain the complete IntelliJ project structure (i.e. the **.iml** file, **.idea** folder, your **.java** files, and your

.class files), it will not be graded. This requirement is in place so that my graders can either pull your code directly from git or unzip it and run it without modification. Similarly, if your project does not run when we try and open it, it will not be graded. This is most likely to occur if you put all of your files in the same directory. By default, IntelliJ separates the source and binary files; so your GitHub repo should reflect this structure.

Problem 1

Problem 1 consists of multiple parts. You should finish one part and ensure that it works before moving to the next part. You will need to refactor (rewrite parts of) your code as you move between parts. Only your final program will be graded.

Part 1

The use of computers in education is referred to as *computer-assisted instruction (CAI)*. Write a program that will help an elementary school student learn multiplication. Use a `SecureRandom` object to produce two positive one-digit integers (you will need to look up how to do this). The program should then prompt the user with a question, such as

```
How much is 6 times 7?
```

The student then inputs the answer. Next, the program checks the student's answer. If it's correct, display the message `"Very good!"` and ask another multiplication question. If the answer is wrong, display the message `"No. Please try again.>again."` and let the student try the same question repeatedly until the student finally gets it right. A separate method should be used to generate each new question. This method should be called once when the application begins execution and each time the user answers the question correctly.

Part 2

Modify the program from Part 1 so that various comments are displayed for each answer as follows:

Possible responses to a correct answer:

```
Very good!
Excellent!
Nice work!
Keep up the good work!
```

Possible responses to an incorrect answer:

```
No. Please try again.
Wrong. Try once more.
Don't give up!
No. Keep trying.
```

Use random-number generation to choose a number from 1 to 4 that will be used to select one of the four appropriate responses to each correct or incorrect answer. Use a `switch` statement to issue the responses.

Part 3

Modify the program from Part 2 to use your question generation method to ask the student 10 *different* questions. Give students only one chance at answering each question. Count the number of correct and incorrect responses typed by the student. After the program has asked 10 questions, and the student types 10 answers, your program should calculate the percentage that are correct. If the percentage is lower than 75%, display "Please ask your teacher for extra help.", then reset the program so another student can try it. If the percentage is 75% or higher, display "Congratulations, you are ready to go to the next level!", then reset the program so another student can try it.

Part 4

Modify the program from Part 3 to allow the user to enter a difficulty level. At a difficulty level of 1, the program should use only single-digit numbers in the problems; at a difficulty level of 2, numbers as large as two digits, and so on. Allow for four levels of difficulty.

Part 5

Modify the program from Part 4 to allow the user to pick a type of arithmetic problem to study. An option of 1 means addition problems only, 2 means multiplication problems only, 3 means subtraction problems only, 4 means division problems only and 5 means a random mixture of all these types.

Problem 2

Create class **SavingsAccount**. Use a static variable **annualInterestRate** to store the annual interest rate for all account holders. Each object of the class contains a private instance variable **savingsBalance** indicating the amount the saver currently has on deposit. Provide method **calculateMonthlyInterest** to calculate the monthly interest by multiplying the **savingsBalance** by

annualInterestRate divided by 12—this interest should be added to **savingsBalance**. Provide a static method **modifyInterestRate** that sets the **annualInterestRate** to a new value.

Write a program to test class **SavingsAccount**. Instantiate two **SavingsAccount** objects, **saver1** and **saver2**, with balances of $2000.00 and $3000.00, respectively. Set **annualInterestRate** to 4%, then calculate the monthly interest for each of 12 months and print the new balances for both savers. Next, set the **annualInterestRate** to 5%, calculate the next month's interest and print the new balances for both savers.

Assignment 3 Rubric (1)

Criteria	Ratings		Pts
Problem 1: A SecureRandom object is used to generate random number	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program displays randomly generated questions and prompts student for answer	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program stores student response in a double precision floating-point variable	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program uses floating-point comparison to determine if the students answer is correct	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program displays the proper response message based on the correctness of the student's answer	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program is able to display multiple positive and negative responses, based on student correctness	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: The specific response displayed to students is determined randomly from within the proper set of responses	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program shall run the user through 10 questions per session	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program shall display the number of correct and incorrect responses at the end of a session	**1 pts Full Marks**	**0 pts No Marks**	1 pts

Criteria	Ratings		Pts
Problem 1: Program shall display a message indicating whether or not the student is ready to advance at the end of a session	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program shall ask the user if they wish to begin a new session once the current session has terminated	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program shall reset its state when a new session is started	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Program shall terminate when the user no longer wishes to start a new session	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: The first thing a student is prompted for when starting a session is a difficulty level of 1, 2, 3, or 4	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: The number of digits in each number of problem correspond to the difficulty level (1 = 1 digit, 2 = 2 digit, 3 = 3 digit, 4 = 4 digit)	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: The second thing a student is prompted for when starting a session is the type of problem to be studied: 1, 2, 3, 4, or 5	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: Problems presented to students during a session reflect the selected problem type (1 is addition only, 2 is multiplication only, 3 is subtraction only, 4 is division only, and 5 is a mix of addition, subtraction, multiplication, and division.)	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 1: A method is used to generate the questions	**2 pts Full Marks**	**0 pts No Marks**	2 pts

Criteria	Ratings		Pts
Problem 1: A method is used to generate the response	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Problem 1: A method is used to determine the difficulty level	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Problem 1: A method is used to determine the problem type	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Problem 1: IntelliJ project is stored on GitHub	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Problem 2: Class named SavingsAccount exists	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: annualInterestRate is a private class variable	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: savingsBalance is a private instance variable	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: calculateMonthlyInterest is a public instance method	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: calculateMonthlyInterest is correctly implemented	**2 pts Full Marks**	**0 pts No Marks**	2 pts

Criteria	Ratings		Pts
Problem 2: modifyInterestRate is a public class method	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: modifyInterestRate is correctly implemented	**2 pts Full Marks**	**0 pts No Marks**	2 pts
Problem 2: Test program instantiates two SavingsAccount objects	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: Test program sets the annualInterestRate to 4%	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: Test program calculates the monthly interest for each of the 12 months and prints the balances for both SavingsAccount instances.	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: Test program sets the annualInterestRate to 5%	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: Test program calculates the monthly interest for the next month's interest and prints the balances for both SavingsAccount instances.	**1 pts Full Marks**	**0 pts No Marks**	1 pts
Problem 2: IntelliJ project is stored on GitHub	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Problem 2: Test program is in a class called Application	**1 pts Full Marks**	**0 pts No Marks**	1 pts

Criteria	Ratings	Pts
Total Points: 50		