CS 342 **Homework#1** **Fall 2023**

Maven and Java

Due Date: Sunday, Sept 10th, by 11:59PM

Description:

In this homework, you will run a Maven project in your development environment of choice. You will also create a new .java file with a few methods.

Implementation Details:

What you have:

Download the zip file from Blackboard and go to the HomeworkOne directory. This is the Maven project you will be working with. It is the beginning of a program that will have a user interface and calculate various savings strategies for the user.

Take a look in the src/main/java folder. You will find two files: **HomeworkOne.java** and **MyMoney.java**. **HomeworkOne.java** just contains the main method inside the HomeworkOne class.

MyMoney.java is a class that will encapsulate financial data from the user and be used, eventually, to calculate and display forecasts of various savings strategies for the user. It can read in double values from a file; which are used in some of the calculations. This class makes calls to a **SavingsFormulas** class where the actual methods for the financial calculations will be located. You should notice that these method calls are being made by the class name not a specific instance of that class. That means that every method in the **SavingsFormulas** class will be static; creating a so called "static" outer class. You should also notice that there is no **SavingsFormulas.java** file in your directory. *****You will be implementing that class; more on that in a bit.*****

If you take a look at src/main/resources, you will find the two text files, values.txt and values2.txt that this program currently uses to test the methods in the SavingsFormulas class. Each text file has one double value per line: values.txt contains dollar amounts saved in a year and values2.txt contains various interest rates for a number of years.

Now take a look in the src/test/java directory and look at the **SavingsMethodTest.java** file. This is a Junit5 test file. We will be doing unit tests with Junit5 soon but for now, you just need to know that this is where the test cases are. Notice that there is a test case for each **SavingsFormulas** method you will create.

CS 342 **Homework#1** **Fall 2023**

What you need to do:

1) In the src/main/java directory, create a new java file called SavingsFormulas.java. This file will have one class defined in it: **public class SavingsFormulas**
2) Inside of the SavingsFormulas class you need to implement the following methods:

public static double futureValueLumpSum(double cash, double interest, int years)
public static double futureValueLS_VariableInterest(double cash, double values[])
public static double compoundSavingsConstant(double cash, double interest, int years)
public static double compoundSavingsVariable(double values[], double interest)

These methods implement basic financial formulas for various types of compounding interest. You will notice that your data is read into arrays of doubles. There are better data structures in Java but since we have not covered them yet, we will stick with arrays which everyone should be comfortable with.

Formula to use for each method:

public static double futureValueLumpSum(double cash, double interest, int years)

This method takes a lump sum of money and figures out how much it will be worth in a certain number of years at a constant interest rate. The formula is

$$FV = PV(1 + i)^N$$

So if I had $10,000 and I wanted to know what it would be worth in 5 years, compounded yearly at 5% interest (corresponding method variable in parenthesis):

PV = 10,000 (**double cash**)
i = .05 (**double interest**)
N = 5 (**int years**)

The result would be $12762.81

public static double futureValueLS_VariableInterest(double cash, double values[])

This method takes a lump sum of money and figures out how much it will be worth in a certain number of years at a varying interest rate per year. The **values** Array is loaded with interest rates from values2.txt. Each index in the array corresponds to the rate for a specific year. For each year you have a unique interest rate i:

year1 = PV * (1+i)
year2 = year1 * (1+i)
year3 = year2 * (1+i)…..you get the idea.

So in the method you would be doing:

cash = cash(1 + values[0]);
cash = cash(1 + values[1]);
And so on for all the rates in **values[]**

Extra credit if you do this one recursively (you will need to write a private helper method)

public static double compoundSavingsConstant(double cash, double interest, double years)

This method calculates the future value of saving the same amount of money each year for a certain number of years at a constant interest rate.

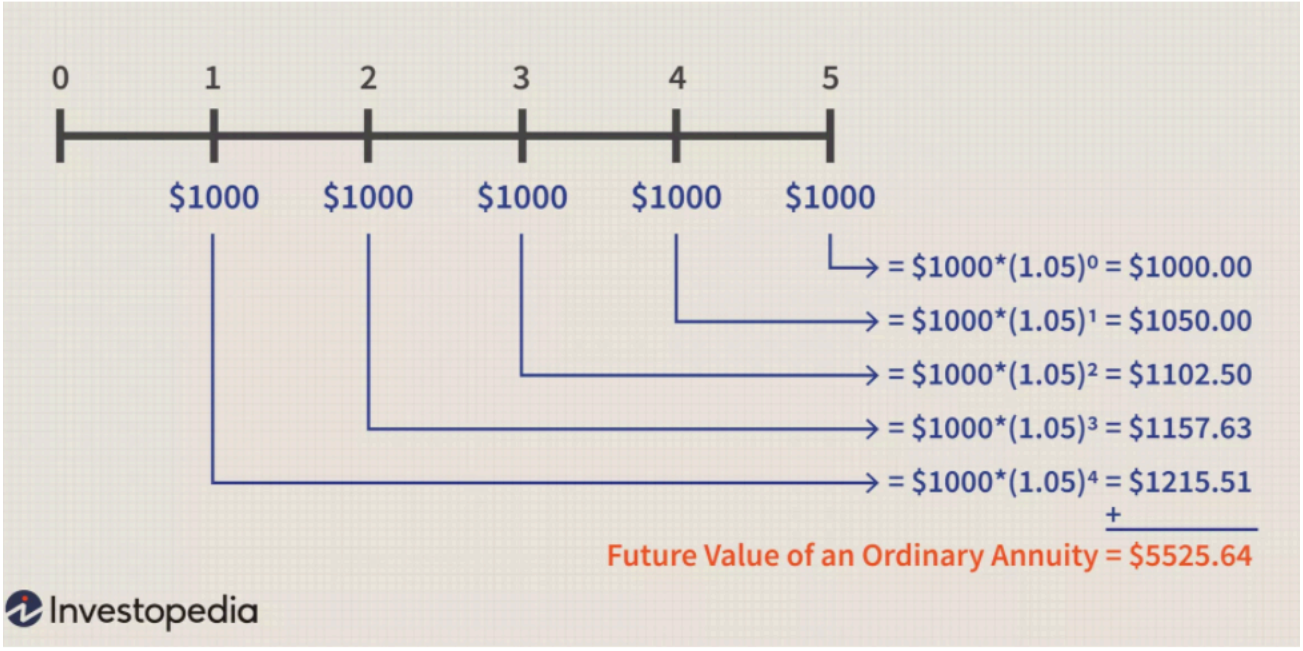
$$FV_A = Pmt\left[\frac{(1+i)^N - 1}{i}\right]$$

So if I invested $1000 at the end of each year for 5 years at 5% interest (corresponding method variable in parenthesis):

Pmt = 10,000 (**double cash**)
i = .05 (**double interest**)
N = 5 (**double years**)

The result would be $5525.63

The image below can help you understand what the formula is doing



https://www.investopedia.com/retirement/calculating-present-and-future-value-of-annuities/

public static double compoundSavingsVariable(double values[], double interest)

This method calculates the future value of investing a different amount of money at the end of each year for a certain number of years at a constant interest rate.

The **values** Array is loaded with the dollar amounts for each year from values.txt. Each index in the array corresponds to a dollar amount for a specific year.

year1 = values[0];
year2 = (year1 * interest) + values[1]……you get the idea

Extra credit if you do this one recursively (you will need to write a private helper method)

How to develop this project:

You will need to import this project, the HomeworkOne directory, as a Maven project in the IDE of your choice or just go to that directory from the command line. You can run the Maven commands:

clean: removes all class files and reports.
test-compile: compiles files in the src/test/java directory
compile: compiles files in the src/main/java directory
test: runs all test cases

Keep in mind that there is no code in main(). The idea here is to write a method and run the test cases to see if it is correct. You can not alter any of the code I have given you.

I suggest that you comment out the test cases for methods you have not implemented yet. For each method you write, you should run Maven clean, to clean your project, and then run Maven test. If the method is correct, you will pass the test/tests for that method (run with the values from the text files if applicable). If not, you will see the value your method returned and the value expected by the test case. You should be able to pass the test cases provided. DO NOT alter the test cases!!!

We will run your projects with different cases and different input files when we grade them.

Electronic Submission:

Put the Maven template folder with your files in a .zip and name it with your netid + MavenJava: for example, I would have a submission called mhalle5MavenJava.zip, and submit it to the link on Blackboard course website.

```
[INFO] -------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------
[INFO] Running SavingsMethodTest
Values in file values.txt
5000.0
5000.0
10000.0
18000.0
22000.0
Values in file values2.txt
0.07
0.05
0.045
0.09
0.07
0.065
[INFO] Tests run: 6, Failures: 0, Errors: 0,
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0,
[INFO]
[INFO] -------------------------------------
[INFO] BUILD SUCCESS
[INFO] -------------------------------------
[INFO] Total time:  2.297 s
[INFO] Finished at: 2020-09-05T18:41:10-05:0
```

WHAT YOU SHOULD SEE IF ALL METHODS ARE IMPLEMENTED PROPERLY

Assignment Details:

Late work on a homework is **NOT ACCEPTED.** Anything past the deadline will result in a zero.

We will test all homework on the command line using Maven 3.6.3. You may develop in any IDE you chose but make sure your homework can be run on the command line using Maven commands. Any homework that does not run will result in a zero. If you are unsure about using Maven, come see your TA or Professor.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml.

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing

your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml.