**PROJECT Bank account simulator program 100 points**

**Objective** To write a program that performs various bank transactions.

***PROJECT DESCRIPTION***

Bank of IIT has contacted you to write, compile and execute a complete program that creates bank account information and executes various transaction details for their clients.

Your program will prompt users for options such as creating an initial balance, entering deposits or withdrawals. Also your program will allow for the printing of account information including interest at various interest rates.

Use loops, user defined methods, conditional and relational logic and the basics of OOP to accomplish the objectives of this progam.

Error trapping will be part of your grade so don’t forget to include some basic error trapping logic! Comment your code thoroughly as well for maximum points.

***Project Details***

For this program you will create *two* separate Java files within your package, namely **AccountHolder** and **AccountHolderTest**.

The **AccountHolder** file must include the following class *field* members and data *methods* to allow for transaction processing.

|  |  |
| --- | --- |
| \*Field Name | Field modifier/type |
| annualInterestRate | static / double |
| balance | double |

\*assume all class level variables are declared private

|  |  |  |  |
| --- | --- | --- | --- |
| \*Method Name | Method (Instance or Static) | Argument | Return Type |
| AccountHolder | Constructor | double | none |
| deposit | Instance | double | void |
| withdrawal | Instance | double | void |
| monthlyInterest | Instance | void | void |
| modifyMonthlyInterest | Static | double | void |
| toString | Instance | void | String |

\*assume all methods are declared public

*Of course if you would like to add any extra fields or methods in your class(es) feel free*

*to do so.*

Coding detail for this file’s methods must include the following:

1. Allow the constructor to accept an argument representing an initial balance for the Account holder. Set your balance member equal to the value passed via the class constructor. Balances cannot start off negative! Include an error message if this is the situation.
2. Define in your **monthlyInterest** method body an assignment statement to update the account holders’ **balance** to be effected as follows:

**balance += balance \* (annualInterestRate / 12.0);**

1. Define in your **modifyMonthlyInterest** method body an assignment statement to update the **annualInterestRate** amount with some argument value (ex. **rateUpdate**) that gets passed thru the method as follows:

**annualInterestRate = rateUpdate;**

Be sure that the updated rate passed thru the method’s parameter is a valid rate

that is greater than or equal to 0 *and* less then or equal to 1.0.

1. For your **deposit** & **withdrawal** methods either have your method body either increase or decrease the holder’s current balance accordingly.

Some added rules to follow here:

1. Do not allow the withdraw to decrease the holder’s balance below $100.

Inform the user of of this if this is the situation.

1. If a withdrawal allows the account balance to drop below $500, a one time transaction fee of $50 will be deducted from the current account holder’s balance.
2. For your **toString** method, include the following statement

**return String.format("$%.2f", balance);**

1. Fully document your methods. Include comments on what your method is to perform, what parameters (i.e. data types) are to be passed in if any, and what will be returned by the method if anything.

The **AccountHolderTest** file must include the following transactional detail from your main method, *executed* in the **following order**

1. Prompt the user for an intial account balance (have the balance passed into the **AccountHolder** constructor).
2. Prompt the user to enter in a deposit amount.
3. Prompt the user for a withdrawal amount.
4. Show an example when a user enters a withdrawal that will drop their balance

below $100.

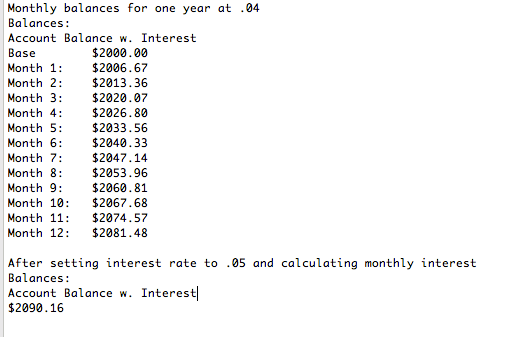
1. Allow the interest for the bank to be initially set at 4%.
2. Display in a columnar format, a report showing interest being added to the users account over a period of 12 months.

Have one column display the month number and one column to show the respected new calculated balance for each particular month. Label your column headings appropriately.

*Note*- you can easily allow for any new balances to print merely by passing your **AccountHolder** object via your output statement (by doing so you trigger your toString method you defined in your AccountHolder class automatically which returns the accounts current balance).

1. Update the interest amount once again to 5%.
2. Display a report of balances for a twelve month period with a current interest rate applied at 4% then show an updated balance with an updated interest rate of 5%.

A sample display showing new balances for a twelve month period and an updated balance at the end given a new interest rate is shown below. Note- included in the display is the word ‘Base’ to depict what the current balance is in the account as a starting point.



After you fully have tested your app, modify your program output to display the current date and time and your name at the end of your output. To do so include the following code:

String timeStamp = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(Calendar.*getInstance*().getTime());

System.***out***.println("Cur dt=" + timeStamp + "\nProgrammed by Sammy Student\n");

Make sure to change to your name as Programmer.

You will need to import in the following to allow for the methods to be acceptable.

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

Note you can always go to your menu and choose **Source>Organize Imports** to automatically include the necessary import statements for your program application!!!

***Please include this code to print the date and time plus your name for all your labs herein including your final project as well****.*

What to turn in?

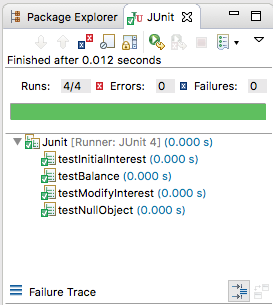
1. First include into a word doc as well the following runtime snapshot(s) for credit. Always label each snapshot depicting what the snapshot entails.
2. Show all your input prompts for the intial balance, a deposit and a withdrawal.
3. Show your message response when the user inputs a withdrawal below $100.
4. Show the 12 month report display when the rate is set at 4%.
5. Display the final account balance after a new rate of 5% has been entered in.
6. Include also Junit tests for 4 outcomes, see snapshot below. 3 mandatory tests must be included namely a test on Balance, a test on ModifyInterest, a test for Initial Interest. Include also an additional test of your choice. **Snapshot** results of all four of your successful test runs for credit as well.

Mandatory tests are to include the following at minimum:

-Test on a Balance: ensure the account balance is positive.

-Test on Initial Interest: ensure interest is initially set to 0.

-Test on ModifyInterest: ensure interest rate is within parameters defined by lab spec.



1. Lastly copy all your java source files (including your junit test source code) into your Word doc. Include the java file name always at the the top of your source code area.
2. Submit your finished word doc (name your filename appropriately, ex. **sstudent1Lab510**) into Blackboard.
3. Submit also a zip file of your entire java project files seperately as well into Blackboard.

Finally, remember to fully document your program! Include at the top most part of your source code the following:

-your name

-the date

-your particular source file name ex. AccountHolder.java

-lab number

Include also a *brief*, yet solid description of your program in your main source file.

**Points also will be deducted if there is any absence of proper documentation.**