**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 program.

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 |

|  |  |  |  |
| --- | --- | --- | --- |
| \*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 |

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

An added rule to follow here:

Disallow a withdrawal attempts to drive account balance to drop below $50. Deliver a message to the console stating that the balance must hold to at least $50.

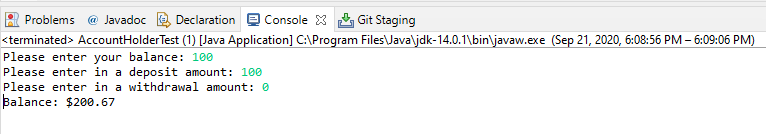
For your **AccountHolderTest** file, include any local variables in main to work the application. Include the following transactional detail from your main method for each run below, *executed* in the **following order.**

Your main() activities

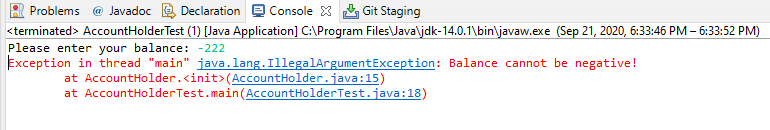
1. Allow the interest for the bank to be initially set at **4%**. This can be hard coded in.
2. Create an AccountHolder object and prompt the user for an initial account balance and have the initial balance passed into the **AccountHolder** constructor.
3. Prompt the user to enter in a deposit amount.
4. Prompt the user for a withdrawal amount.
5. Display an ending balance, including added monthly interest, to the account holder.

Snapshot your results above and paste it into Word for credit. Label files accordingly. Ex. Lab1\_yourInitials.docx. Submit all files to BlackBoard when complete.

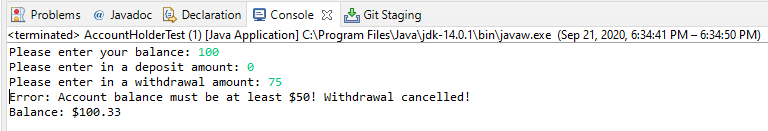
Case 1: Correct balance, withdrawal, and deposit amount



Case 2: Incorrect balance



Case 3: Incorrect withdrawal amount



/\*

This program will hold all the methods and attributes that an account should have

Author: Tony Acosta Hernandez

\*/

**public** **class** AccountHolder

{

**public** **static** **double** *annualInterestRate*;

**public** **double** balance;

**public** AccountHolder(**double** balance)

{

**if**(balance < 0)

{

**throw** **new** IllegalArgumentException("Balance cannot be negative!"); //Ends the program after throwing exception

}

**else**

**this**.balance = balance;

}

**public** **void** deposit(**double** depo)

{

**this**.balance += depo; //Adds deposit to our balance

}

**public** **void** withdrawal(**double** with)//balance cannot drop below $50

{

**if** (balance - with < 50) //Don't do anything if its less than 50

System.***out***.println("Error: Account balance must be at least $50! Withdrawal cancelled!");

**else**

**this**.balance -= with; //Need error trapping

}

**public** **void** MonthlyInterest()//Calculates our monthly interest

{

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

}

}

**import** java.util.\*;

/\*

This is our tester program for AccountHolder. We will createa an AccountHolder object and work with it

Author: Tony Acosta Hernandez

\*/

**public** **class** AccountHolderTest

{

**public** **static** **void** main(String[] args)

{

Scanner scan = **new** Scanner(System.***in***);

AccountHolder.*annualInterestRate* = .04;

//Ask for balance here

System.***out***.print("Please enter your balance: ");

**double** balance = scan.nextDouble();

//Constructor will handle negative balance

AccountHolder account = **new** AccountHolder(balance);

//Get a deposit amount

System.***out***.print("Please enter in a deposit amount: ");

**double** deposit = scan.nextDouble();

account.deposit(deposit);

//Get a withdrawal amount. withdrawal() will handle the added rule

System.***out***.print("Please enter in a withdrawal amount: ");

**double** withdrawal = scan.nextDouble();

account.withdrawal(withdrawal);

//Get and print out monthly balance after interest rate is applied

account.MonthlyInterest();

System.***out***.printf("Balance: $%.2f", account.balance);

scan.close();

}

}