Francisco Rois Siso

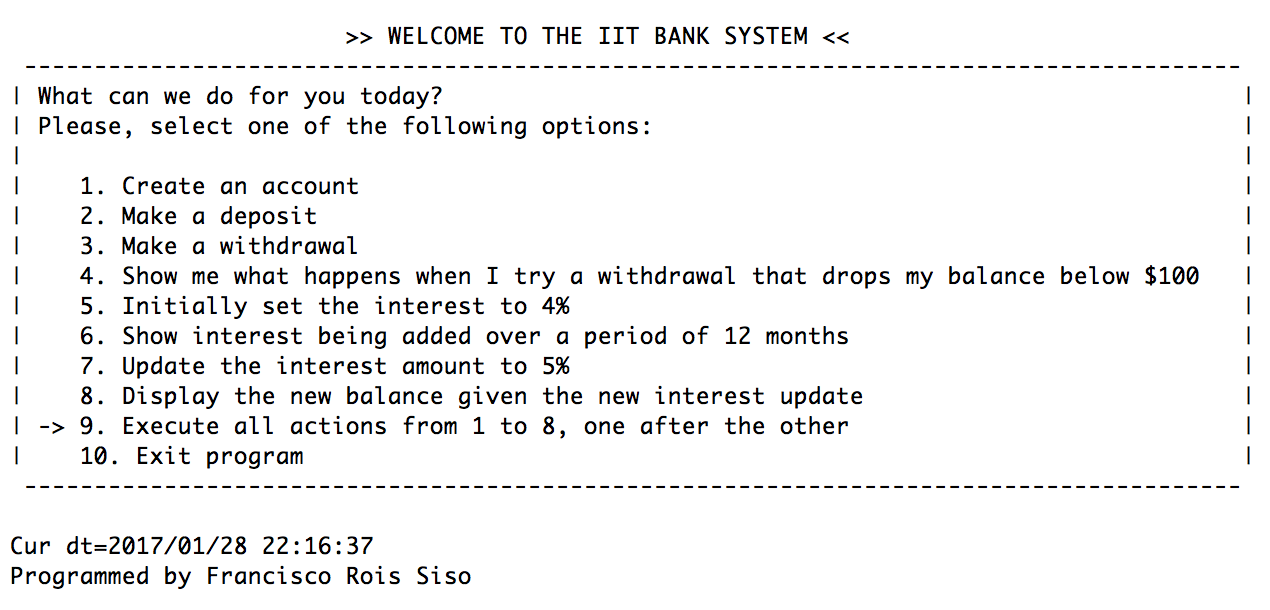
A20385948

01/28/2017

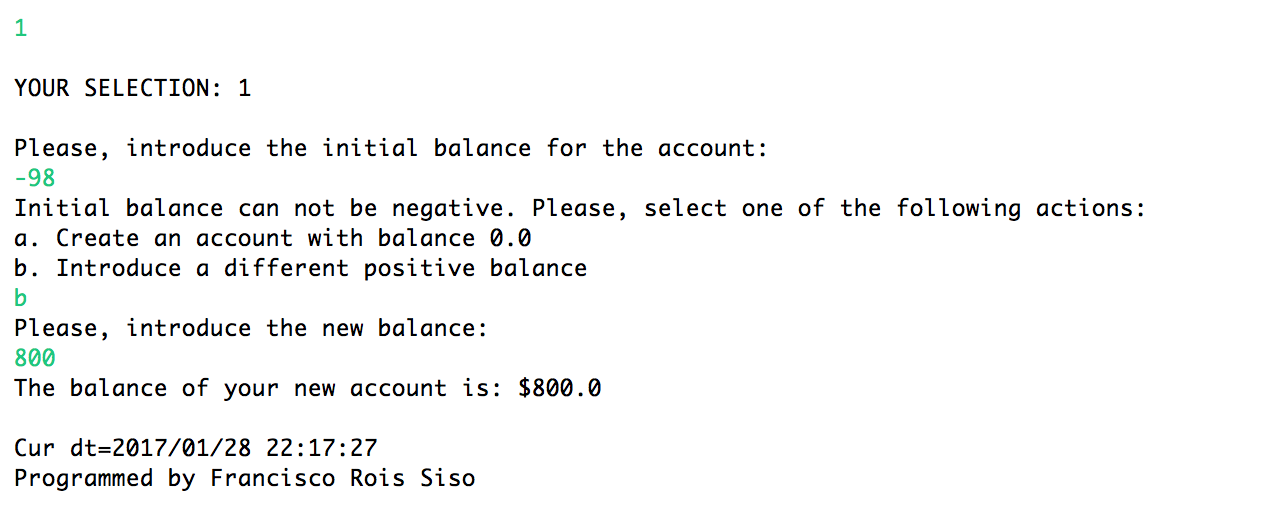
Lab1 - ITMD510 Object-Oriented Application Development

**Content**

1. Snapshots
2. AccountHolderTest.java (source code)
3. AccountHolder.java
4. **SNAPSHOTS**

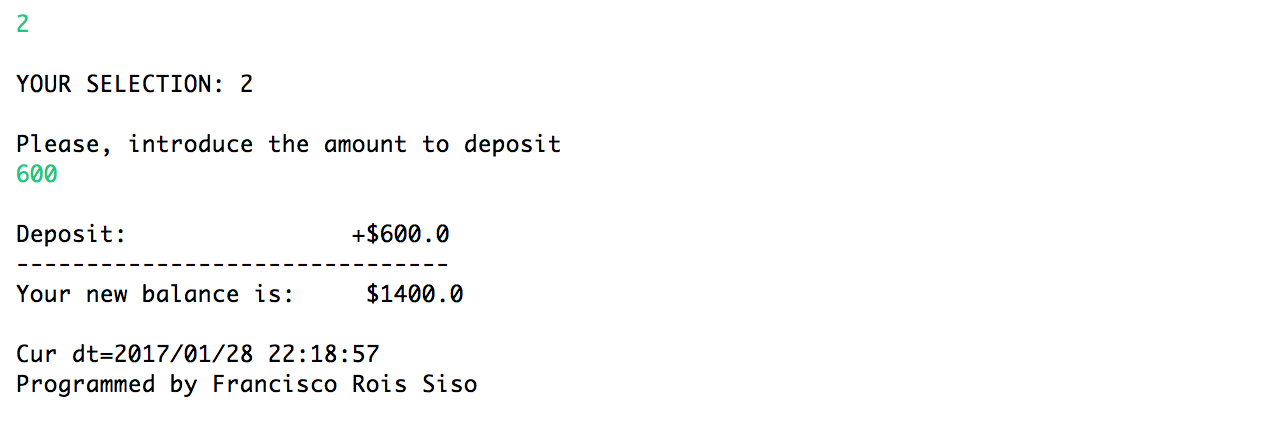
****

*Main Menu displayed at the beginning of the program and after every action is finished.*

****

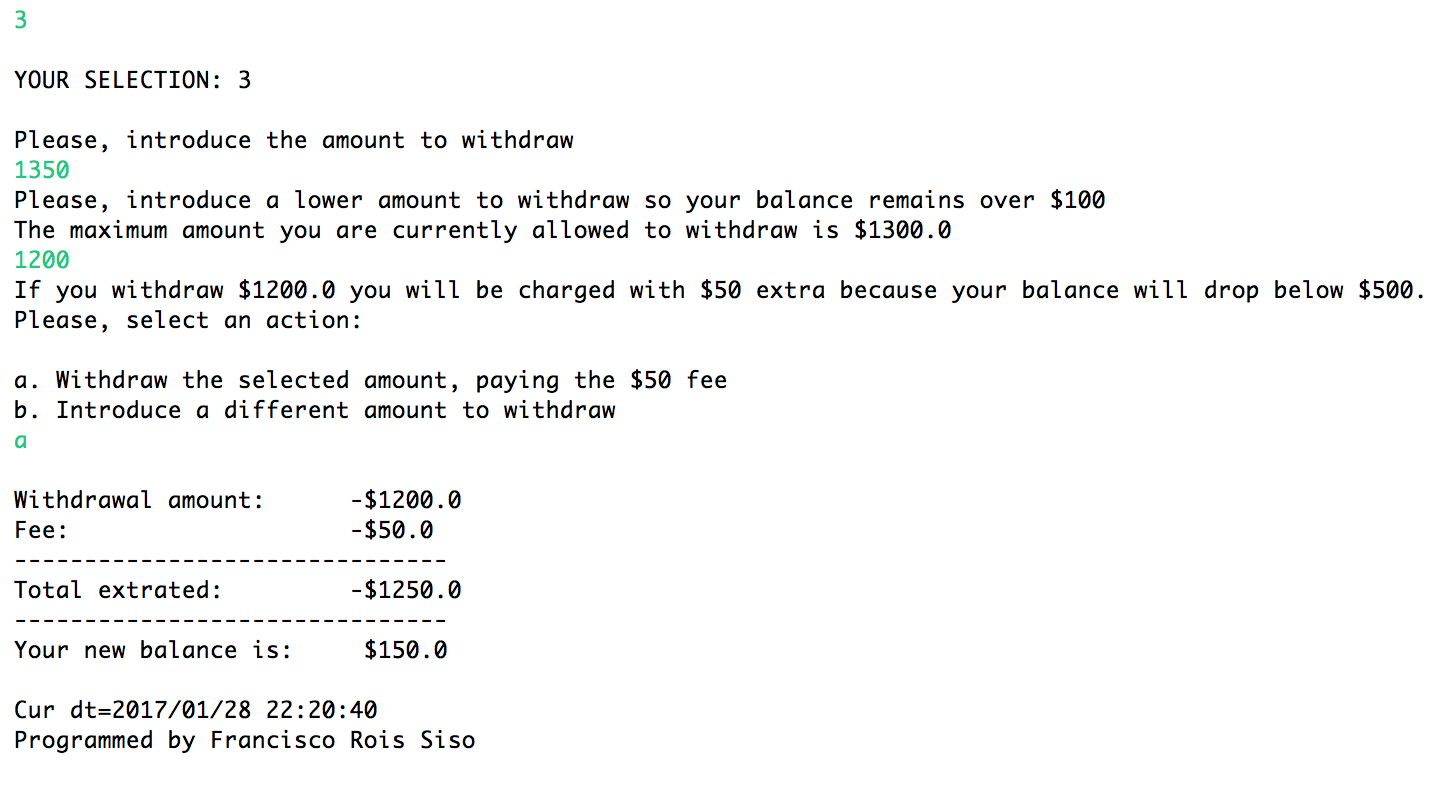
*Selection 1: Create an account*

*An initial balance is required to the user. The user introduces a negative value (negative) and is asked to choose an action. In this case the user decides to introduce a different amount as initial balance.*

****

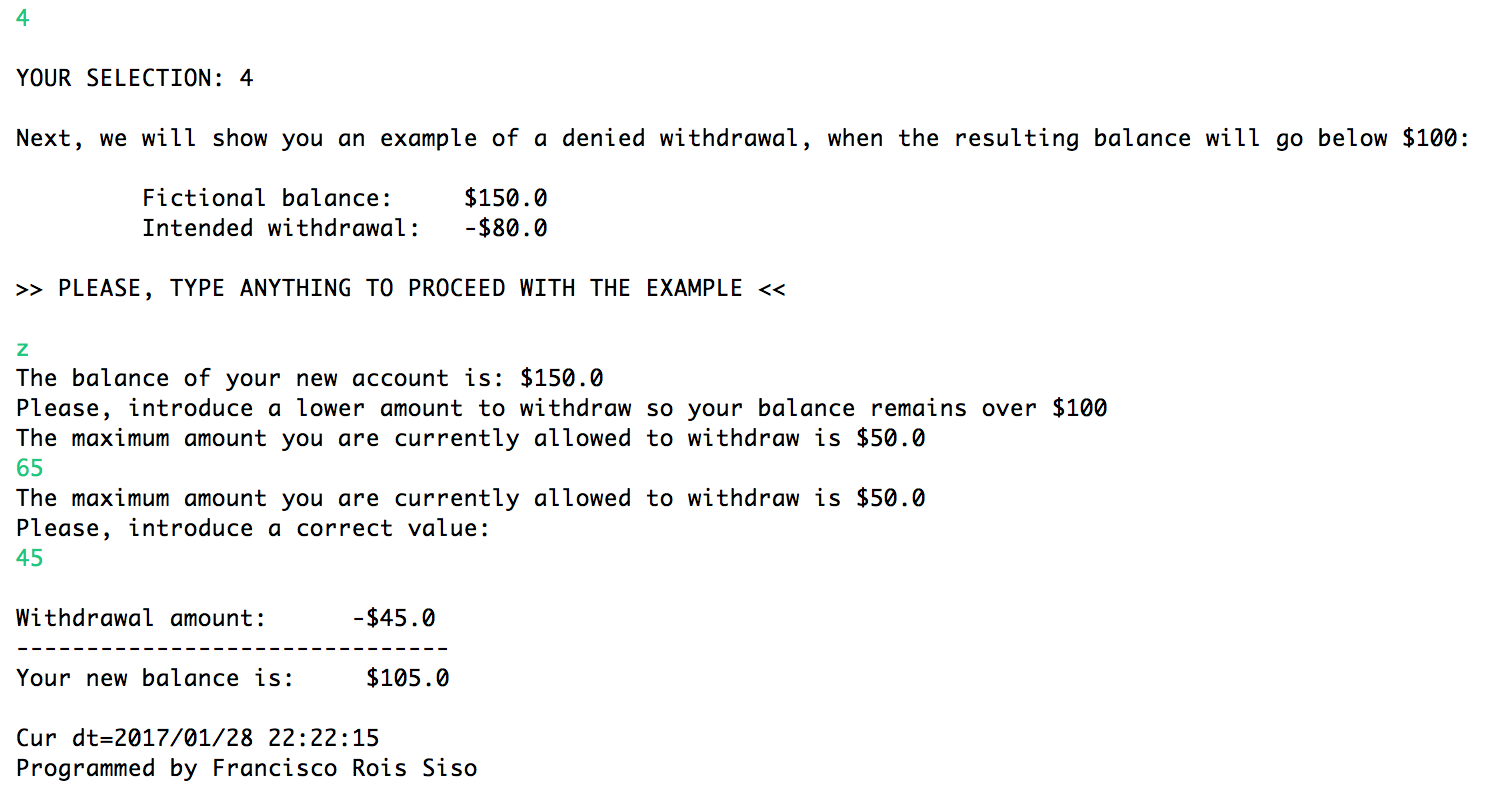
*Selection 2: Make a deposit*

*The user is asked to introduce an amount to deposit. In this example the user introduces $600. The amount is added to the current balance and then it is displayed.*

****

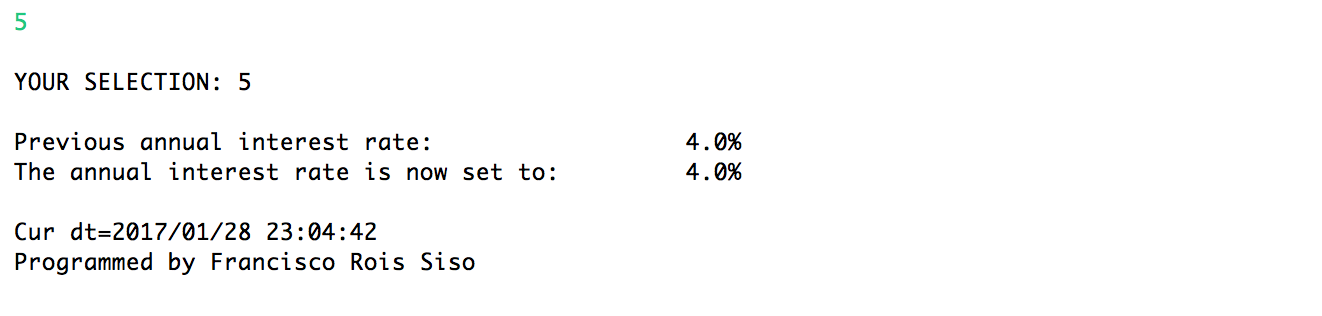
*Selection 3: Make a withdrawal*

*The user is asked to introduce an amount to withdraw. In this example the user introduces 1350, which would drop the balance below $100. This is not permitted and therefore the user is asked to select a different amount. The user introduces 1200, which is an accepted value. However, it would drop the balance below $500 and therefore the user would be charged with $50 extra. The user is informed of the situation and asked to choose an action. In this example the user chooses to withdraw the indicated amount anyway, paying the fee. The operation is indicated to the user, as the final state of the balance.*

****

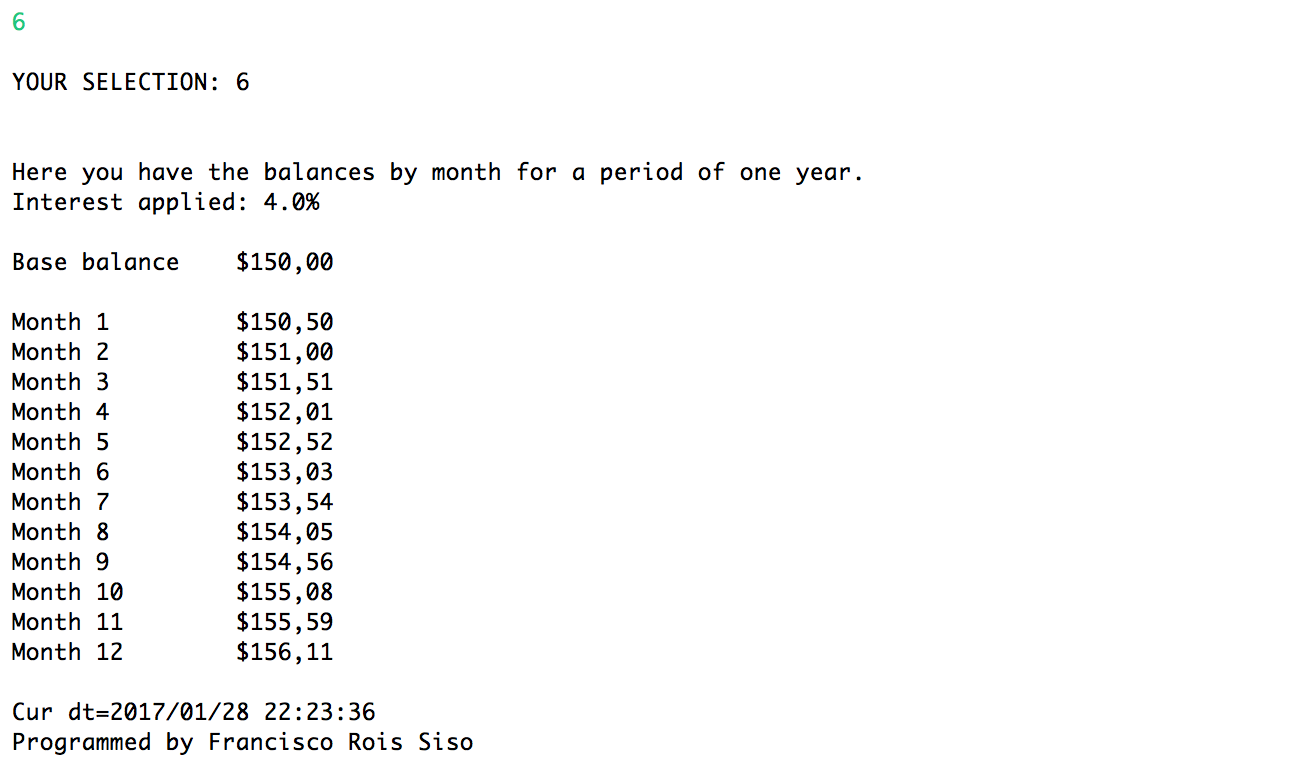
*Selection 4: Show me what happens when I try a withdrawal that drops my balance below $100*

*In this case it is shown a fictional example in which a user tries to withdraw more money than he is allowed to, in order to keep the balance over $100. All the process is shown, as well as the final state of the account.*

****

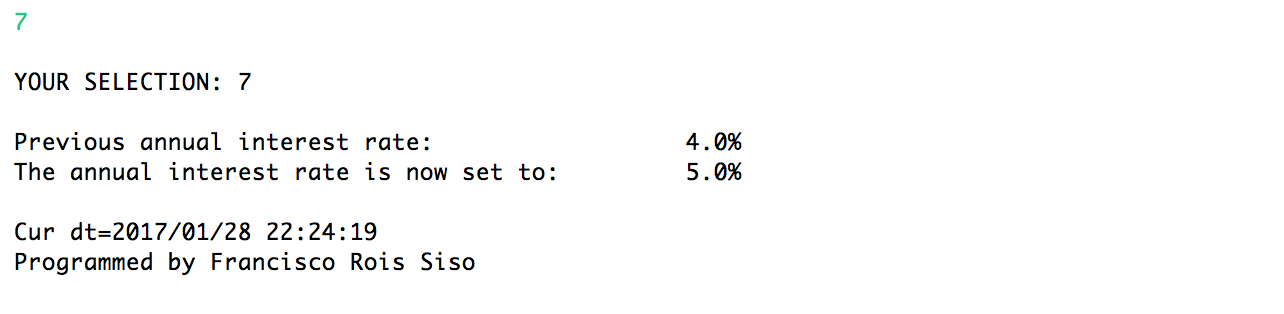
*Selection 5: Set the interest to 4%*

*The annual interest rate is updated to 0.04. Previous and new values of the interest rate are show (same in this case).*

****

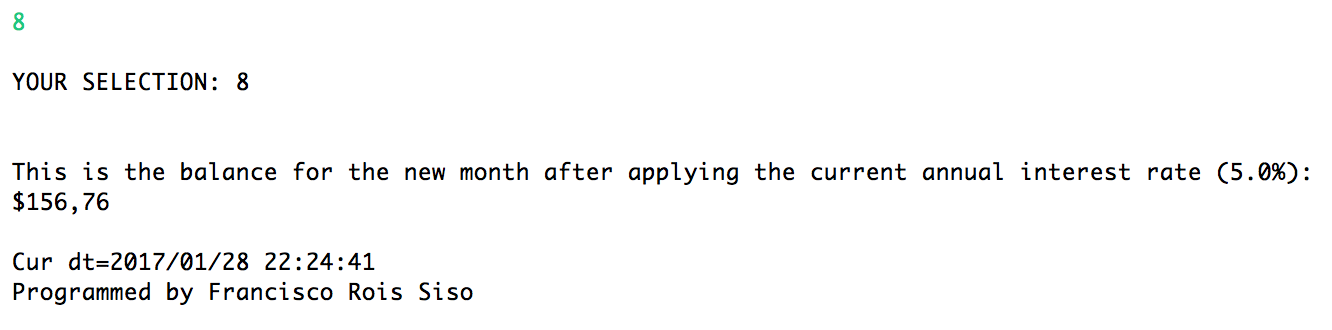
*Selection 6: Show interest being added over a period of 12 months*

*The current monthly interest, calculated from the current 0.04 annual interest rate, is applied every month during 12 months, starting from a base balance of $150.*

****

*Selection 7: Update the interest amount to 5%*

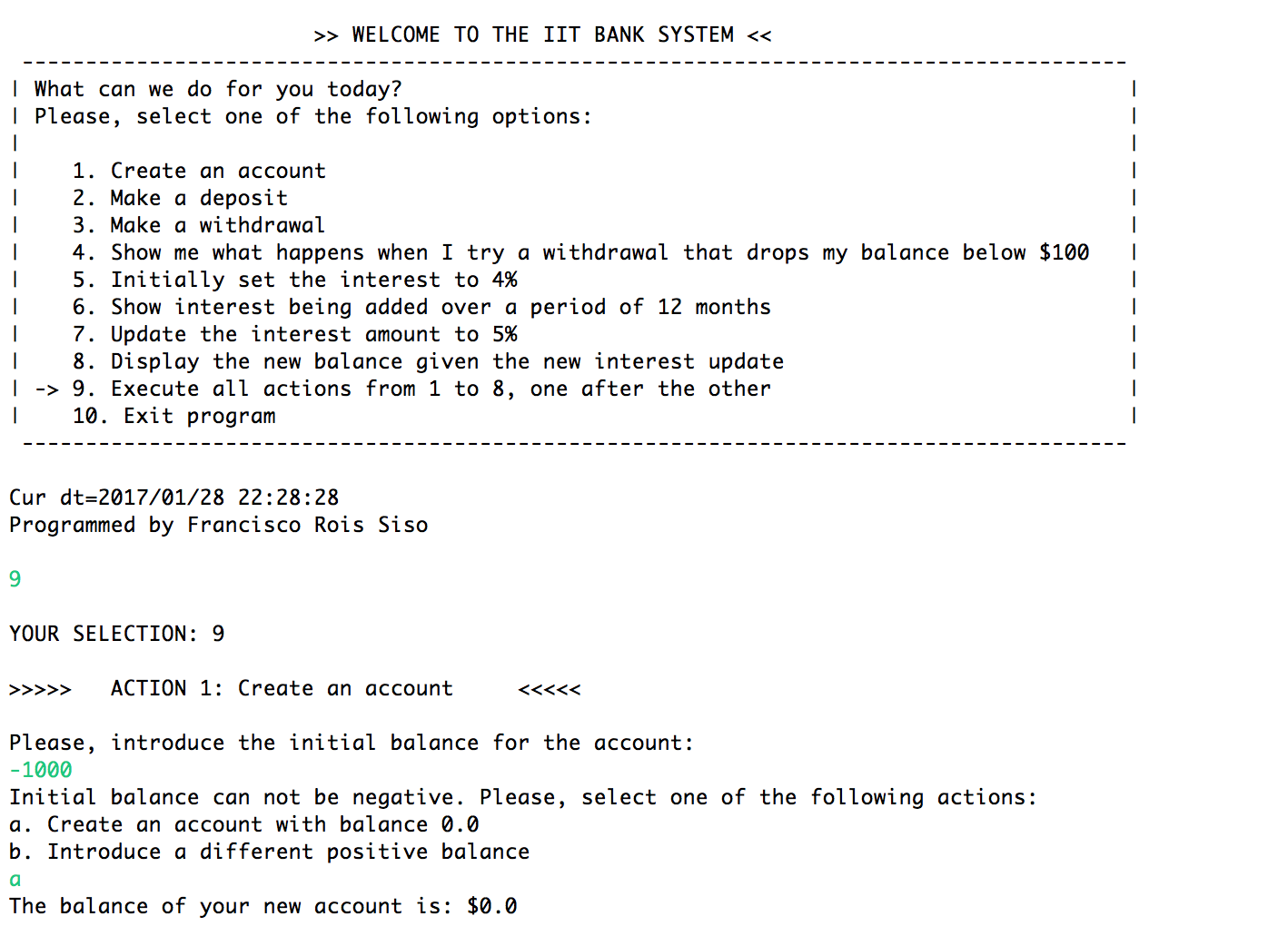
*The annual interest rate is updated to 0.05. Both the previous and current values are shown (0.04 and 0.05 in this case).*

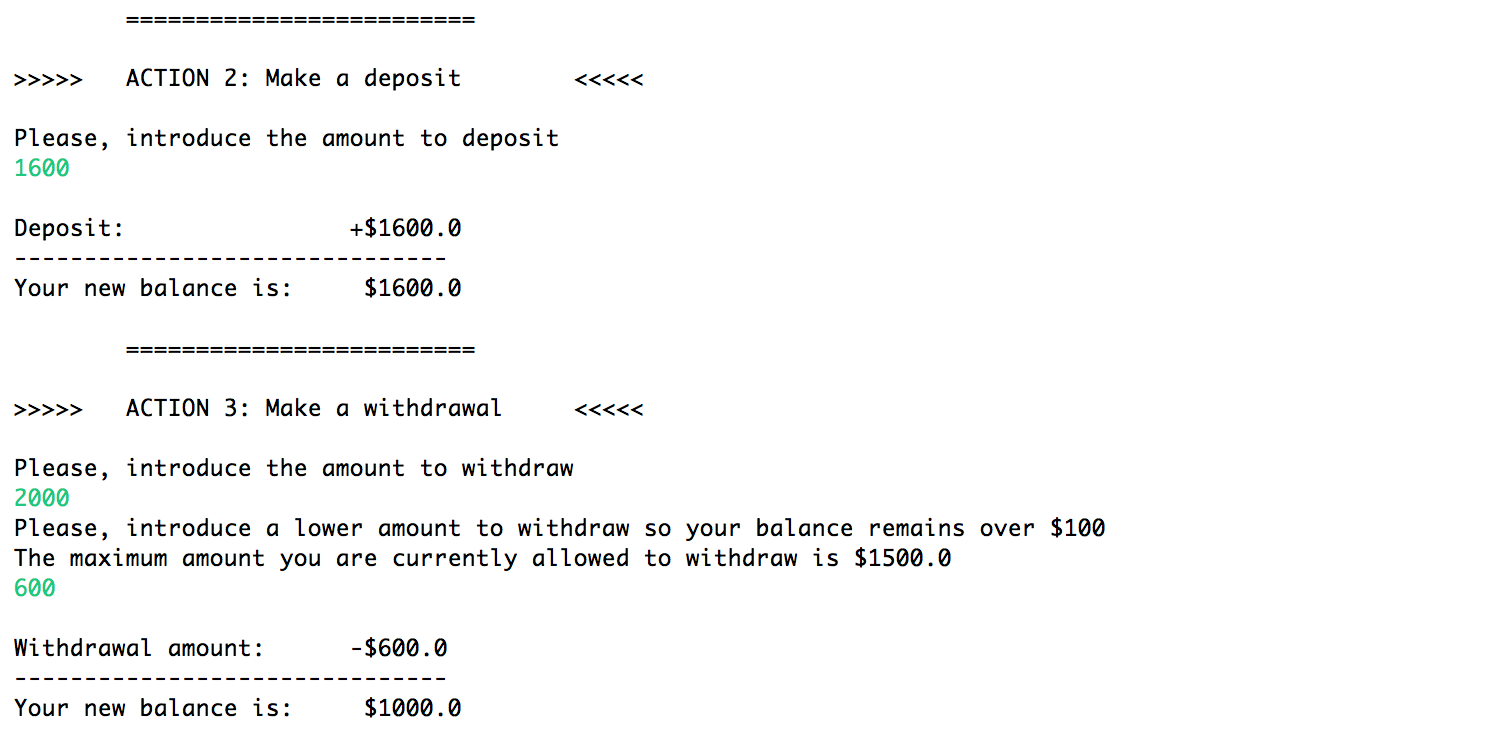
****

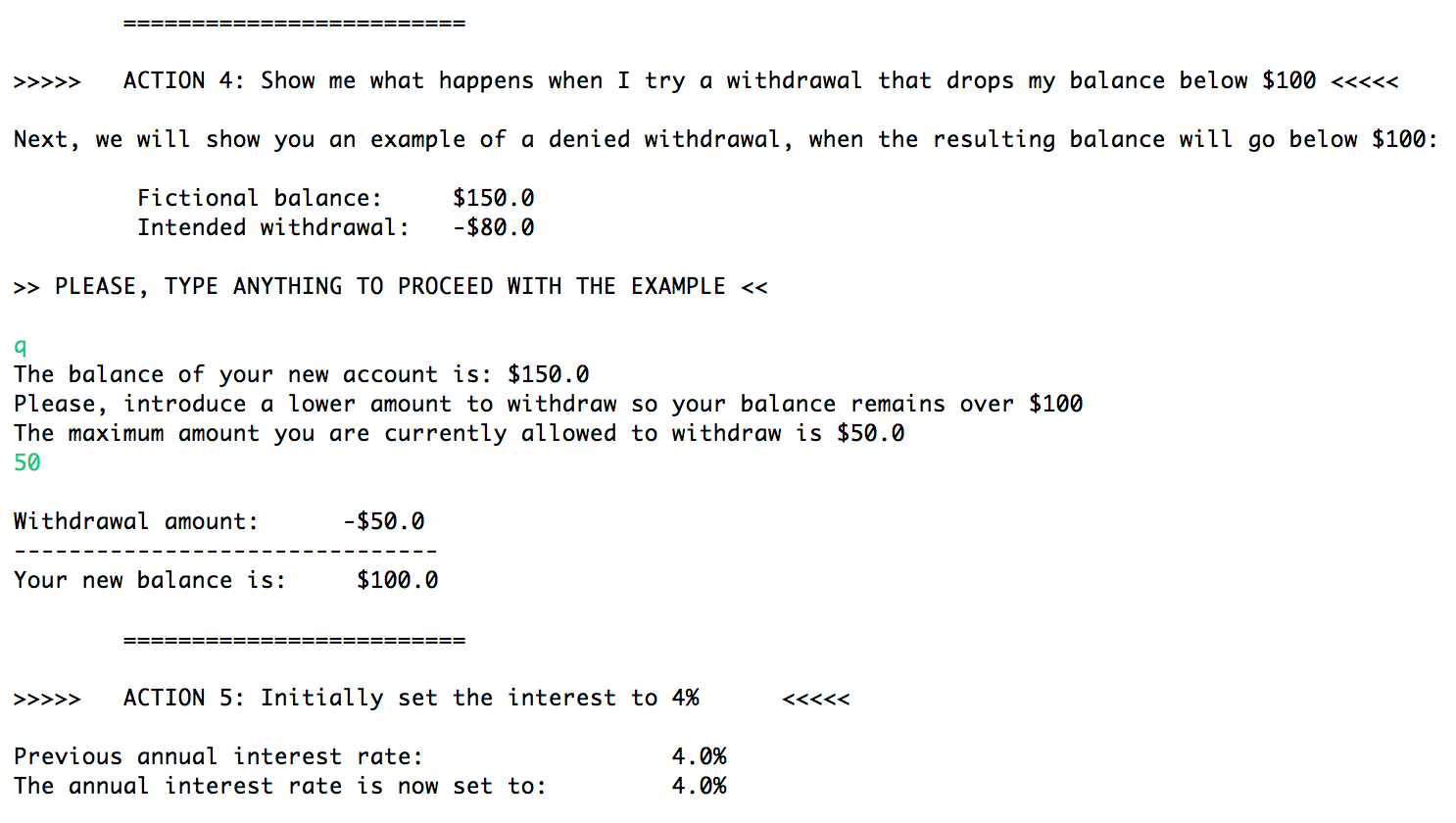
*Selection 8: Display the new balance given the new interes update*

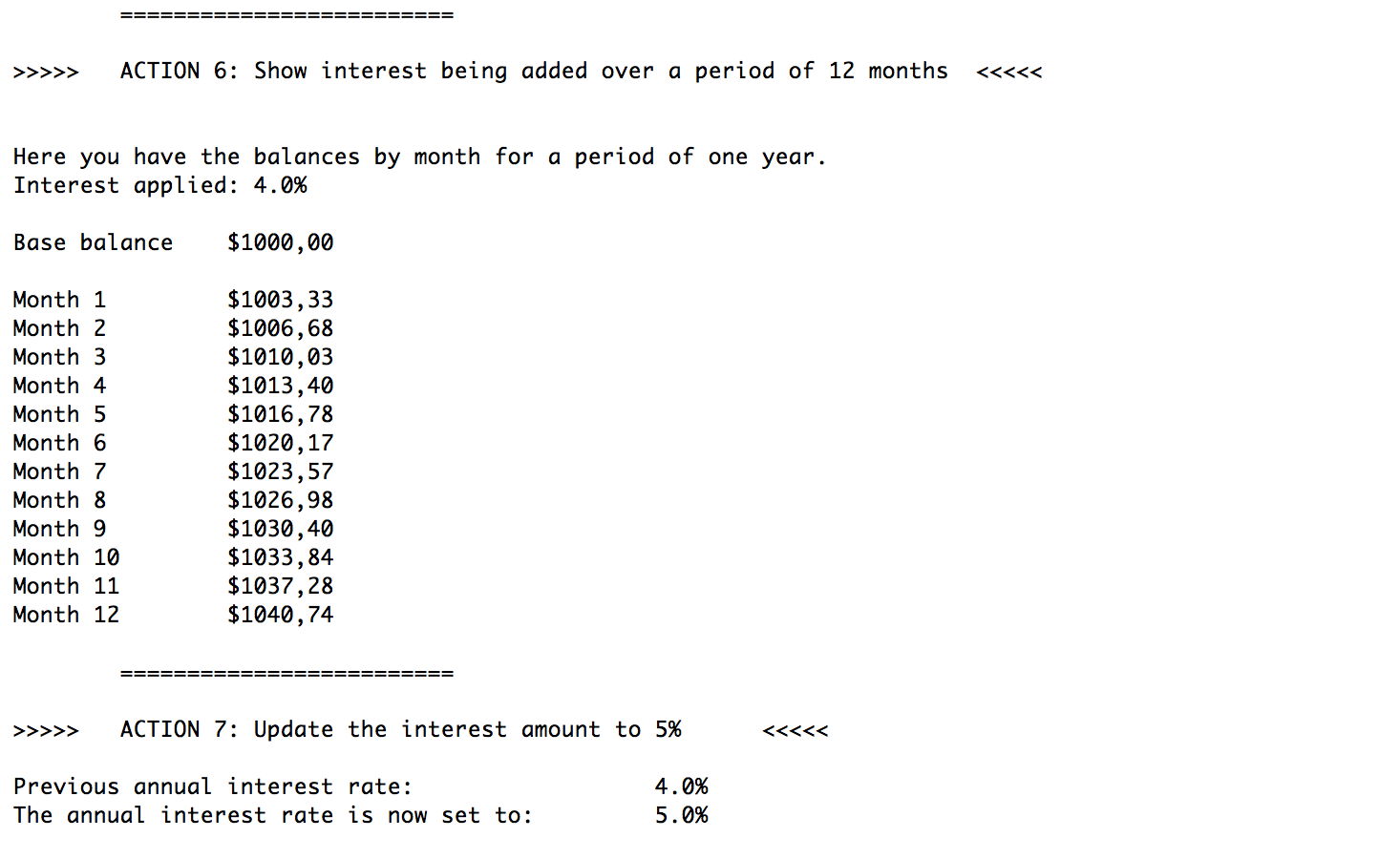
*The balance for the new month is calculated by applying the new monthly interest rate, which is obtained from the current annual interest rate (0.05 in this case).*

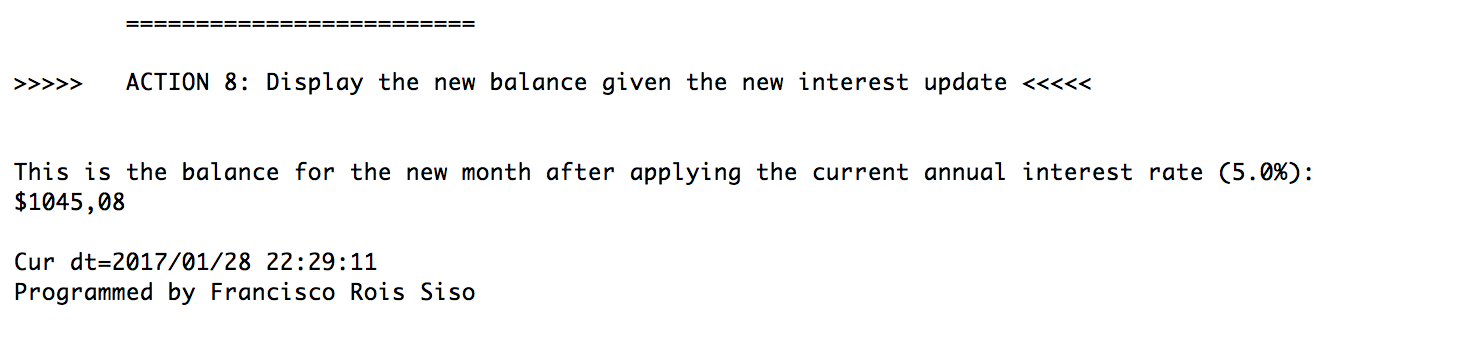
*The following screenshots show the result from selecting the action number 9, which executes all actions from 1 to 8, all in a row:*

****

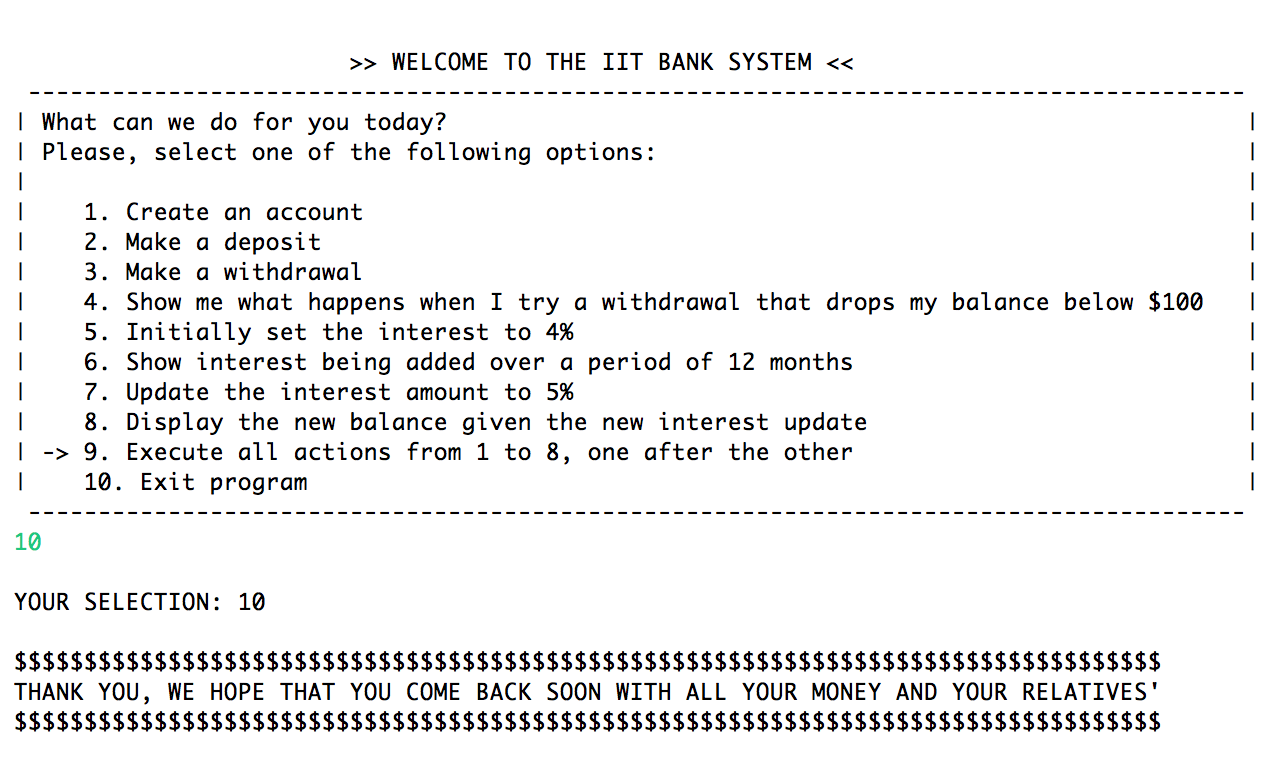
****

****

****

****

*Finally, the following screenshot is the result from selecting the action number 10, “Exit program”, which shows the following message to the user:*

****

1. **AccountHolderTest.java (source code)**

/\*

Program to simulate an account holder and actions related to the account.

The user is shown a menu of actions and can select to create an account with a certain initial balance,

make deposits or withdrawals, show examples of not fulfillment of certain withdrawal conditions, change the annual interest rate,

add interests and see results over a period of 12 months, apply the rate to a month or execute several actions in a row.

- Programmer: Francisco Rois Siso

- Date: 01/28/2017

- Source File Name: AccountHolderTest.java

- Lab 1

- ITMD510 Object-Oriented Application Development

\*/

// import InputMismatchException to catch the exception when the user types something incorrect

**import** java.util.InputMismatchException;

// import Scanner to receive input from the user

**import** java.util.Scanner;

// import libraries for date and time display, for lab submission purposes

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

/\*\*

\* The class AccountHolderTest displays a menu to the user to select an action between 10 possibilities.

\* All of the actions use the class AccountHolder.java and test its functionalities.

\* The actions available are:

\* 1. Create an account,

\* 2. Make a deposit,

\* 3. Make a withdrawal,

\* 4. Show me what happens when I try a withdrawal that drops my balance below $100,

\* 5. Initially set the interest to 4%,

\* 6. Show interest being added over a period of 12 months,

\* 7. Update the interest amount to 5%,

\* 8. Display the new balance given the new interest update,

\* 9. Execute all actions from 1 to 8, one after the other,

\* 10. Exit program

\*

\* **@author** Francisco Rois Siso

\*

\*/

**public** **class** AccountHolderTest {

// static AccountHolder object, which will be created and modified by the user

**static** AccountHolder *account*;

// boolean to indicate that the account has already been created. This boolean is used if the user

//tries to make a deposit or a withdrawal without having before introduced an initial balance amount

**static** **boolean** *accountCreated* = **false**;

// Scanner variable sc declaration

**static** Scanner *sc*;

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

// display the Welcome Menu and the possible actions that the user can select to perform

*displayWelcomeMenu*();

// show date and time for lab submission purposes

*showDateAndTime*();

// declaration of the String that will store the number of the action selected by the user

String selection;

// boolean to indicate that the user has finished and therefore he wants to exit the program

// It will be used in the main while loop

**boolean** finish = **false**;

// scan input from the user

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

//the program will display again the Main Menu to the user after every action,

// until the user has finished and therefore selects to exit the program (action 10)

**while**(!finish){

// scan input from the user

selection = *sc*.next();

// show the selection if is one of the available ones

**if**(selection.equals("1")||selection.equals("2")||selection.equals("3")||selection.equals("4")||selection.equals("5")||selection.equals("6")||selection.equals("7")||selection.equals("8")||selection.equals("9")||selection.equals("10")){

System.*out*.println("\nYOUR SELECTION: "+selection+"\n");

}

**try**{

// perform a different action depending on the selection made by the user

**switch** (selection){

// 1. Create an account

**case** "1":

*createAnAccount*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 2. Make a deposit

**case** "2":

*makeADeposit*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 3. Make a withdrawal

**case** "3":

*makeAWithdrawal*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 4. Show me what happens when I try a withdrawal that drops my balance below $100

**case** "4":

*example100Withdrawal*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 5. Initially set the interest to 4%

**case** "5":

*updateAnnualInterestRate*(0.04);

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 6. Show interest being added over a period of 12 months

**case** "6":

*showInterestAddedByMonth*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 7. Update the interest amount to 5%

**case** "7":

*updateAnnualInterestRate*(0.05);

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 8. Display the new balance given the new interest update

**case** "8":

*displayBalanceNextMonthWithInterest*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 9. Execute all actions from 1 to 8, one after the other

**case** "9":

*executeActions1to8*();

// show date and time for lab submission purposes

*showDateAndTime*();

// display again the Main Menu

*displayWelcomeMenu*();

**break**;

// 10. Exit program

**case** "10":

// change boolean to true in order to exit the while loop

finish = **true**;

*exitProgram*();

**break**;

**default**:

**break**;

}

}

// catch InputMismatchException in the case that the user types something unexpected

**catch**(InputMismatchException e){

// the user is asked to try again the action and the Main Menu is displayed

System.*out*.println("\nThe input introduced is not valid. Please, try again.");

*displayWelcomeMenu*();

}

// catch any other kind of exception in the case that it happens

**catch**(Exception e){

// The user is informed, asked to try again the action and the Main Menu is displayed

System.*out*.println("\nAn unexpected exception occurred. Please, try again.");

*displayWelcomeMenu*();

}

}

}

/\*\*

\* displayedWelcomeMenu displays the Menu Menu so the user can select one of the available actions

\*/

**static** **void** displayWelcomeMenu(){

System.*out*.println("\n\t\t\t>> WELCOME TO THE IIT BANK SYSTEM <<");

System.*out*.println(" ---------------------------------------------------------------------------------------");

//MENU

System.*out*.print("| What can we do for you today?");

System.*out*.print("\t\t\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| Please, select one of the following options:");

System.*out*.print("\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("|\t\t\t\t\t\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("|  1. Create an account");

System.*out*.print("\t\t\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("|  2. Make a deposit");

System.*out*.print("\t\t\t\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| 3. Make a withdrawal");

System.*out*.print("\t\t\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| 4. Show me what happens when I try a withdrawal that drops my balance below $100");

System.*out*.print("\t");

System.*out*.println("|");

System.*out*.print("| 5. Initially set the interest to 4%");

System.*out*.print("\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| 6. Show interest being added over a period of 12 months");

System.*out*.print("\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| 7. Update the interest amount to 5%");

System.*out*.print("\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| 8. Display the new balance given the new interest update");

System.*out*.print("\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| -> 9. Execute all actions from 1 to 8, one after the other");

System.*out*.print("\t\t\t\t");

System.*out*.println("|");

System.*out*.print("| 10. Exit program");

System.*out*.print("\t\t\t\t\t\t\t\t\t");

System.*out*.println("|");

System.*out*.println(" ---------------------------------------------------------------------------------------");

}

/\*\*

\* createAnAccount asks the user for an initial balance value and creates the account if the value is accepted

\*/

**static** **void** createAnAccount(){

// ask for an initial balance amount

System.*out*.println("Please, introduce the initial balance for the account: ");

// get the input amount specified by the user

**double** initialAmount = *sc*.nextDouble();

// try to create a new AccountHolder object with the initial balance amount specified, by calling the constructor

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

// boolean changed to true to indicate that the account has been created and deposits or withdrawals can be done

*accountCreated* = **true**;

}

/\*\*

\* makeDeposit sums a certain quantity to the current account's balance

\*/

**static** **void** makeADeposit(){

// if an account has not been created yet, the user is informed and the deposit process is stopped

**if**(*accountCreated* == **false**){

System.*out*.println("You need to create an account first in order to make a deposit.");

}

// else the user is asked to introduce the amount to deposit and the deposit is performed

**else**{

System.*out*.println("Please, introduce the amount to deposit");

**double** depositAmount = *sc*.nextDouble();

*account*.deposit(depositAmount);

}

}

/\*\*

\* makeAWithdrawal allows the user to make a withdraw

\*/

**static** **void** makeAWithdrawal(){

// if an account has not been created yet, the user is informed and the withdrawal process is stopped

**if**(*accountCreated* == **false**){

System.*out*.println("You need to create an account first in order to make a withdrawal.");

}

// else, the user is asked for the amount to withdraw and the process is then initialized

**else**{

System.*out*.println("Please, introduce the amount to withdraw");

**double** withdrawalAmount = *sc*.nextDouble();

*account*.withdrawal(withdrawalAmount);

}

}

/\*\*

\* example100Withdrawal shows an example to the user in which an account holder with a balance of $150 tries to perform

\* a withdrawal of $80, so the balance would be below $100. This action is therefore not permitted and the user is asked

\* to introduce a suitable amount to proceed with the withdrawal. This is a simulation, therefore the balances shown in

\* this example are not used by other methods outside this one.

\*/

**static** **void** example100Withdrawal(){

// fictional balance to illustrate the example

**double** fictionalBalance = 150.0;

// fictional intended withdrawal to illustrate the example

**double** intendedWithdrawal = 80.0;

// the user is informed about the example

System.*out*.println("Next, we will show you an example of a denied withdrawal, when the resulting balance will go below $100:");

System.*out*.println("\n\t Fictional balance:\t$"+fictionalBalance);

System.*out*.println("\t Intended withdrawal:\t-$"+intendedWithdrawal+"\n");

System.*out*.println(">> PLEASE, TYPE ANYTHING TO PROCEED WITH THE EXAMPLE <<\n");

*sc*.next();

// an AccountHolder object is created with the fictional initial balance to illustrate the example

AccountHolder example\_account = **new** AccountHolder(fictionalBalance);

// the fictional withdrawn process is started

example\_account.withdrawal(intendedWithdrawal);

}

/\*\*

\* updateAnnualInterestRate allows to update the value of the static variable annualInterestRate, defined in AccountHolder.java

\* **@param** newAnnualInterest The new value for the annual interest rate, as a double

\*/

**static** **void** updateAnnualInterestRate(**double** newAnnualInterest){

// the user is informed of the previous and new values, and the static variable value is updated

System.*out*.println("Previous annual interest rate: \t\t\t"+(*account*.getAnnualInterestRate())\*100.0+"%");

//AccountHolder.annualInterestRate = newAnnualInterest;

*account*.modifyMonthlyInterest(newAnnualInterest);

System.*out*.println("The annual interest rate is now set to: \t"+(*account*.getAnnualInterestRate())\*100.0+"%");

}

/\*\*

\* showInterestAddedByMonth shows to the user the balances resulting from the addition of the interests month by month,

\* during a period of 12 months. The current value of the annual interest rate is applied

\*/

**static** **void** showInterestAddedByMonth(){

//if the account has not been created yet by using the action 1 on the menu, it is required to the user to continue

**if**(*account* == **null**){

System.*out*.println("\nPlease, select option 1 first and create an account in our bank, then you will be able to see the interests applied.");

}

**else**{

// the user is informed of the annual interest rate that will be applied, and then the balances for 12 months are displayed

System.*out*.println("\nHere you have the balances by month for a period of one year.");

System.*out*.println("Interest applied: "+(*account*.getAnnualInterestRate()\*100.0)+"%\n");

// when the AccountHolder is added to the output String, the method toString is automatically called

System.*out*.println("Base balance\t"+*account*+"\n");

System.*out*.print("Month 1");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 2");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 3");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 4");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 5");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 6");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 7");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 8");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 9");

*account*.monthlyInterest();

System.*out*.println("\t\t" + *account*);

System.*out*.print("Month 10");

*account*.monthlyInterest();

System.*out*.println("\t" + *account*);

System.*out*.print("Month 11");

*account*.monthlyInterest();

System.*out*.println("\t" + *account*);

System.*out*.print("Month 12");

*account*.monthlyInterest();

System.*out*.println("\t" + *account*);

}

}

/\*\*

\* displayBalanceNextMonthWithInterest applies the current monthly interest rate to the following month, calculated from

\* the current annual interest rate

\*/

**static** **void** displayBalanceNextMonthWithInterest(){

//apply the current monthly interest rate, calculated from the current annual interest rate

*account*.monthlyInterest();

// information is shown

System.*out*.println("\nThis is the balance for the new month after applying the current annual interest rate ("+*account*.getAnnualInterestRate()\*100.0+"%):");

System.*out*.println(*account*);

}

/\*\*

\* executeActions1to8 is a method that executes all the actions of the Main Menu from 1 to 8, one after the other

\*/

**static** **void** executeActions1to8(){

System.*out*.println(">>>>>\tACTION 1: Create an account\t<<<<<\n");

*createAnAccount*();

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 2: Make a deposit\t<<<<<\n");

*makeADeposit*();

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 3: Make a withdrawal\t<<<<<\n");

*makeAWithdrawal*();

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 4: Show me what happens when I try a withdrawal that drops my balance below $100\t<<<<<\n");

*example100Withdrawal*();

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 5: Initially set the interest to 4%\t<<<<<\n");

*updateAnnualInterestRate*(0.04);

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 6: Show interest being added over a period of 12 months\t<<<<<\n");

*showInterestAddedByMonth*();

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 7: Update the interest amount to 5%\t<<<<<\n");

*updateAnnualInterestRate*(0.05);

System.*out*.println("\n\t=========================");

System.*out*.println("\n>>>>>\tACTION 8: Display the new balance given the new interest update\t<<<<<\n");

*displayBalanceNextMonthWithInterest*();

}

/\*\*

\* exitProgram shows a goodbye message when the user chooses to exit the program

\*/

**static** **void** exitProgram(){

System.*out*.println("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$");

System.*out*.println("THANK YOU, WE HOPE THAT YOU COME BACK SOON WITH ALL YOUR MONEY AND YOUR RELATIVES'");

System.*out*.println("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$");

}

/\*\*

\* showDateAndTime allows to show the current date and time for lab submission purposes

\*/

**static** **void** showDateAndTime(){

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

System.*out*.println("\nCur dt=" + timeStamp + "\nProgrammed by Francisco Rois Siso\n");

}

}

1. **AccountHolder.java**

/\*

- Programmer: Francisco Rois Siso

- Date: 01/28/2017

- File Name: AccountHolder.java

- Lab 1

- ITMD510 Object-Oriented Application Development

\*/

// import java.util library

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

/\*\*

\* The class AccountHolder simulates an account holder, including its balance, annual interest rate,

\* methods to access and modify the data, and methods to make a deposit or a withdrawal under certain conditions.

\*

\* **@author** Francisco Rois Siso

\*

\*/

**public** **class** AccountHolder {

// variables declaration and initialization in some case

**private** **static** **double** *annualInterestRate* = 0.04;

**private** **double** balance;

/\*\*

\* AccountHolder constructor without input parameters.

\* Creates an object AccountHolder with balance 0.0.

\*/

**public** AccountHolder(){

//if no parameter is included, the balance is set to 0.0 by default

**this**.balance = 0.0;

//annualInterestRate = 0.04;

}

/\*\*

\* AccountHolder constructor with the initial balance as an input parameter.

\* Creates an object AccountHolder with the initial balance indicated, if a condition is fulfilled: the initial balance is not negative.

\* If the input is negative, the user is informed and asked to select a different amount or create an account with initial balance 0.0

\* Once the initial balance is correct, the AccountHolder is created with that value

\*

\* **@param** initialBalance The initial balance for the new AccountHolder, as a double

\*/

**public** AccountHolder(**double** initialBalance){

// if the initial balance is not negative, proceed normally

**if**(initialBalance >= 0.0){

**this**.balance = initialBalance;

}

// if the initial balance is negative, do the following:

**else**{

// show possible actions to proceed. Create and account with balance 0.0 or introduce a different balance

System.*out*.println("Initial balance can not be negative. Please, select one of the following actions:");

System.*out*.println("a. Create an account with balance 0.0");

System.*out*.println("b. Introduce a different positive balance");

// open scanner to receive input from the user

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

String answer = scanner.next();

// the user must select one of the possible actions to proceed. The user is asked again until he proceeds correctly

**while**(!answer.equals("a") && !answer.equals("A") && !answer.equals("b") && !answer.equals("B")){

System.*out*.println("Please, select one of the possible actions (a or b)");

answer = scanner.next();

}

// a. Create an account with balance 0

**if**(answer.equals("a")||answer.equals("A")){

**this**.balance = 0.0;

}

// b. Introduce a different positive balance

**else** {

// ask for the new positive balance

System.*out*.println("Please, introduce the new balance: ");

**double** newBalance = scanner.nextDouble();

// if the balance introduced continues to be negative, ask again until the value introduced is positive

**if**(newBalance < 0.0){

// boolean to indicate that the balance introduced by the user is negative

**boolean** negativeBalance = **true**;

System.*out*.println("Please, introduce a positive value for the balance: ");

// the user is asked until the balance introduced is correct (non negative)

**while**(negativeBalance){

newBalance = scanner.nextDouble();

// if the value introduced is negative, ask again

**if**(newBalance < 0.0){

System.*out*.println("Please, introduce a positive value for the balance: ");

}

// when the value is finally positive or 0, then exit the while loop and proceed to create the account

**else**{

// exit the loop by changing the value of the boolean

negativeBalance = **false**;

}

}

}

//when the new balance introduced is positive or 0, create the new account with that balance

**this**.balance = newBalance;

}

}

// inform the user about the balance of the new account created

System.*out*.println("The balance of your new account is: $"+balance);

}

/\*\*

\* setBalance method

\* **@param** balance The new balance of the account, as a double

\*/

**public** **void** setBalance(**double** balance){

**this**.balance = balance;

}

/\*\*

\* getBalance method

\* **@return** The balance of the account, as a double

\*/

**public** **double** getBalance(){

**return** **this**.balance;

}

/\*\*

\* Method to deposit a certain amount in the account

\* **@param** depositAmount Amount to deposit in the account, as a double

\*/

**public** **void** deposit(**double** depositAmount){

// add the deposit amount to the current balance

balance += depositAmount;

// show the deposit amount

System.*out*.println("\nDeposit:\t\t+$"+depositAmount);

System.*out*.println("-------------------------------");

// inform about the new balance after the deposit

System.*out*.println("Your new balance is:\t $"+balance);

}

/\*\*

\* Method to withdraw a certain amount from the account.

\* The withdrawal can only be done if the resulting balance is equal or higher than $100.

\* If a withdrawal allows the account balance to drop below $500,

\* a transaction fee of $50 will be deducted from the current account holder’s balance.

\*

\* **@param** withdrawalAmount Amount requested to withdraw from the account, as a double

\*/

**public** **void** withdrawal(**double** withdrawalAmount){

**double** withdrawal = withdrawalAmount;

// open scanner to receive input from the user

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

// if the balance is lower than $100, the user is not allowed to withdraw, since the balance is already under the minimum

**if**(balance <= 100.0){

// inform the user

System.*out*.println("You are not allowed to withdraw, your balance is below $100");

// withdrawal is not permitted, therefore a withdrawal of $0 is performed

withdrawal = 0.0;

}

// the withdrawal is done only if the resulting balance remains above 100

**else** **if**(balance - withdrawal < 100.0){

// ask for a correct withdrawal value

System.*out*.println("Please, introduce a lower amount to withdraw so your balance remains over $100");

//calculate maximum withdrawal

**double** maximumWithdrawal = balance-100;

System.*out*.println("The maximum amount you are currently allowed to withdraw is $"+maximumWithdrawal);

// receive answer from user

**double** newWithdrawal = sc.nextDouble();

// check that the new amount is correct

// if the amount is still higher than the maximum permitted, keep asking for a correct value

**if**(newWithdrawal > maximumWithdrawal){

// boolean to indicate that the amount to withdraw indicated by the user is not accepted

**boolean** invalidWithdrawal = **true**;

// the user is asked to introduce an amount until the value introduced is a valid withdrawal amount

**while**(invalidWithdrawal){

// user is informed of the maximum he is allowed to withdraw to stay under the required conditions

System.*out*.println("The maximum amount you are currently allowed to withdraw is $"+maximumWithdrawal);

System.*out*.println("Please, introduce a correct value: ");

//receive amount from user

newWithdrawal = sc.nextDouble();

// when the amount is valid, get out of the loop by changing the value of the boolean

**if**(newWithdrawal <= maximumWithdrawal){

invalidWithdrawal = **false**;

}

}

}

// change the value of the withdrawal by the new correct value, after all the conditions are fulfilled

withdrawal = newWithdrawal;

}

// calculate how would it be the balance after the withdrawal

**double** potentialBalance = balance - withdrawal;

// if the balance is going to be lower than 500, the user will be charged with 50.

// The user is alerted of this fact and is offered a different action

**if**(balance >= 500.0 && potentialBalance < 500.0){

System.*out*.println("If you withdraw $"+withdrawal+" you will be charged with $50 extra because your balance will drop below $500.");

System.*out*.println("Please, select an action: ");

System.*out*.println("\na. Withdraw the selected amount, paying the $50 fee");

System.*out*.println("b. Introduce a different amount to withdraw");

// receive answer from user

String action = sc.next();

// the answer must be a, A, b or B

**while**(!action.equals("a")&&!action.equals("A")&&!action.equals("b")&&!action.equals("B")){

System.*out*.println("\nPlease, select an action to proceed:");

System.*out*.println("a. Withdraw the selected amount, paying the $50 fee");

System.*out*.println("b. Introduce a different amount to withdraw");

action = sc.next();

}

// a. Withdraw the selected amount: simply continue the program

// b. Introduce a different amount to withdraw

**if**(action.equals("b")||action.equals("B")){

**double** maximumWithdrawalWithoutFee = balance - 500;

// It is shown the maximum amount to withdraw by the user if he does not want to be charged a $50 fee

System.*out*.println("The maximum amount to withdraw to keep the balance at $500 is $" + maximumWithdrawalWithoutFee);

// The user is asked again to introduce a quantity to withdraw, now that he is aware of the situation

System.*out*.println("Please, select the amount that you want to withdraw");

**double** newWithdrawalProbablyWithoutFee = sc.nextDouble();

// the withdrawal amount is updated with the new specified amount

withdrawal = newWithdrawalProbablyWithoutFee;

}

}

// show final amount to withdraw

System.*out*.println("\nWithdrawal amount:\t-$"+ withdrawal);

// charge $50 fee if balance drops below $500

**if**(balance >= 500.0 && (balance - withdrawal) < 500.0){

balance -= 50.0;

// show fee and total extracted

System.*out*.println("Fee:\t\t\t-$50.0");

System.*out*.println("-------------------------------");

**double** totalExtracted = withdrawal+50.0;

System.*out*.println("Total extrated:\t\t-$"+totalExtracted);

}

//update balance

balance -= withdrawal;

System.*out*.println("-------------------------------");

// inform of the new balance

System.*out*.println("Your new balance is:\t $"+balance);

}

/\*\*

\* monthlyInterest applies to the balance the equivalent monthly interest rate, calculated from the annual interest rate

\*/

**public** **void** monthlyInterest(){

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

}

/\*\*

\* modifyMonthlyInterest allows to update the private static variable annualInterestRate, in the case that the conditions

\* are correctly fulfilled (0.0<=newrate<=1.0)

\* **@param** rateUpdate new annual interest rate, as a double

\*/

**public** **void** modifyMonthlyInterest(**double** rateUpdate){

// check that the new interest rate is equal or higher than 0.0 and lower or equal to 1.0

**if**(rateUpdate < 0.0){

System.*out*.println("The new annual interest rate can not be negative");

}

**else** **if**(rateUpdate > 1.0){

System.*out*.println("The new annual interest can not be higher than 1.0");

}

// if the conditions are fulfilled, then update the value for the interest rate

**else**{

*annualInterestRate* = rateUpdate;

}

}

/\*\*

\* getAnnualInterestRate get the private static variable annualInterestRate

\*/

**public** **double** getAnnualInterestRate(){

**return** *annualInterestRate*;

}

/\*\*

\* Method toString modified to show the balance in a certain format, as a String

\*/

**public** String toString(){

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

}

}