**CMSC 203**

**Fall 2015**

**Assignment #3**

This assignment is to implement a GUI application for a bank to support its banking needs. A customer should be able to conduct typical banking transactions, such as making deposits, withdrawing cash, and obtaining their account balance.

**Algorithm/Pseudocode**

1. Create a GUI using the javafx application and name the class “BankingSystem,” which extends the Application.

2. In the BankingSystem class, create the labels and textfields for the banking system GUI application, which will show the customer’s first name, last name, transaction amount, and account balance.

3. Create the buttons that will be used to execute the banking transactions:

* “Create Account Button: - will take in the customer's first and last name and create a new account.
* “Deposit” - will add money into the user’s account balance
* “Withdraw” - will add money into the user’s account balance
* “Clear” - will add clear the transaction amount text field.
* “Exit” - will exit the application.

4. Make sure at the start, only the first name and last name textfields are editable, and the “Create New Account” button is active. The rest of the fields and buttons should be uneditable or inactive.( Only after the user enters their first and last name, and creates a new account by pressing the “Create New Account” button, will then the rest of the fields and buttons become active in the application. The balance textfield should always be uneditable.)

5. Create a vertical box, and add all the labels to the children of that box pane. Make the text right aligned, in the center of the pane.

6. Create another vertical box, and add all the textfields to the children of that box pane. Keep it center aligned.

7. Create a horizontal box that will nest the labels box and then the textfields box to the children of the horizontal box. This will align the first name label with the first name textfield, etc... Keep the pane in a centered alignment.

8. Create another horizontal box, and add the buttons to the children of that box. Keep it center aligned.

9. Create the parent vertical box, and then add the horizontal box (with the nested labels and textfields) and the horizontal box (with the buttons), to the children of that box. This will group all of the contents that will be shown in the Banking system GUI. Keep it center aligned.

10. Place the parent vertical box (which contains the labels, textfields, and buttons) into the center area of a borderpane.

11. Set the borderpane, which contains the contents of the GUI as the root of the scene, and place the scene on the stage.

12. Add functionality to all the buttons by creating the event handlers for them, which correspond to different classes that will perform the functions of the buttons, when clicked.

13. First start by creating a new class called Customer, which will store the name of the customer creating a new bank account.

* Create a constructor for the Customer class, which will take in the customer's first and last name as parameters and store it.
* Create a method called getFirstName, which will returns the customer's first name
* Create a method called getLastName, which will returns the customer's last name
* Create a method called toString, which will return the customer's full name in String representation.

14. Second, create a class called BankAccount, which will contain all the methods needed to perform typical banking transactions for the customer.

* Create a constructor for the BankAccount class, which will take in a customer's first and last name as string parameters, and set the starting balance to 0. It will also create a new object of the Customer class using the parameters of the first and last name.
* Create another constructor which will take the parameter of a customer, and sets the starting balance to 0.
* Create a method called deposit, which will take in a parameter of type double, and add that to the current balance amount.
* Create a method called withdraw, which will take in a parameter of type double, and subtract that from the current balance amount.
* Create a method called getBalance, which will return the balance.
* Create a method called getCustomer, which will return the customer's full name by calling the toString() method of the Customer class.
* Create a method called toString, which will return the customer's name followed by their account balance in String represenation.

15. After you have the BankAccount and Customer classes and their methods set, you can start creating the event handler methods in the BankingSystem class, for the buttons to perform their respective functions.

* For the “Create New Account” button, the event handler must extract the users first and last name from the textfields, and use that to create a new BankAccount object from the BankAccount class
  + Once the new account is made, make the first name and last name textfields uneditable, and disable the “Create New Account” button. Also, enable the deposit, withdraw, clear, and exit buttons. Also make the transaction amount textfield editable.
* For the “Deposit” button, the event handler must extract the number from the transaction amount textfield.
  + If the number extracted is less than or equal to 0, then display a JOption pane asking the customer to re-enter a positive dollar amount, for a deposit to be made.
  + Else, use the instance of the new BankAccount class that was created to call the deposit method, which will add the number extracted to the customer's balance, and display the amount in the balance textfield.
* For the “Withdraw” button, the event handler must extract the number from the transaction amount textfield.
  + If the number extracted is less than or equal to 0, then display a JOption pane asking the customer to re-enter a positive dollar amount.
  + Else, use the instance of the new BankAccount class that was created to call the withdraw method, which will subtract the number extracted from the customer's balance, and display the amount in the balance textfield. .
* For the “Clear” button, the event handler must clear the text that as entered in the transaction amount textfield.
* For the “Exit button, the event handler must exit/quit the application

16. Add the event handlers to the buttons, in order for them to execute when clicked on.

17. Add the main method in the BankingSystem class, which will launch and run the banking application.

**Classes and UML Diagrams:**

**//description:** Will implement the GUI of the banking system, and display its contents, layout, and function of the buttons.

public class BankingSystem extends Application

{

Label firstNameLabel, lastNameLabel, transactionAmtLabel, balanceLabel;

TextField firstNameTextField, lastNameTextField, transactionAmtTextField, balanceTextField; Button newAccountButton, depositButton, withdrawButton, clearButton, exitButton;

BankAccount accuser;

NumberFormat defaultFormat;

}

|  |
| --- |
| BankAccount |
| - balance : double  - name : Customer |
| + BankAccount ( firstName, lastName : String)  + BankAccount (c : Customer)  + deposit (amount : double) : void  + withdraw (amount : double) : void  + getBalance () : void  + getCustomer () : void  + toString () : String |

**//description:** Keeps track of the customer’s account balance, and performs the deposit, and withdrawal of money in a user’s bank account.

public class BankAccount

{

private double balance;

private Customer name;

public BankAccount (String firstName, String lastName)

public BankAccount (Customer c)

public void deposit (double amount)

public void withdraw (double amount)

public void getBalance ()

public void getCustomer ()

public String toString ()

}

|  |
| --- |
| Customer |
| - firstName : String  - lastName : String  - fullName : String |
| + BankAccount ( f, l : String)  + getFirstName()  : String  + getLastName() : String  + toString () : String |

**//description:** Keeps tracks of the customer’s name, whose bank account the transactions are being performed on.

public class Customer

{

private String firstName;

private String lastName;

private Sting fullName;

public Customer (String f, String l)

public String getFirstName()

public String getLastName()

public String toString ()

}

**Method Documentation:**

public void start (Stage stage)

//precondition : none

//postcondition: Banking application GUI is set.

//description: Sets the layout of the GUI, with all the labels, textfields, and buttons the banking application will need.

public static void main (String[] args)

//precondition : GUI must be set, and event functionality must be programmed for it to perform the banking transactions.

//postcondition: Application will execute.

//description: Will launch the GUI application, and perform the banking functions the user wishes to do.

public BankAccount (String firstName, String lastName)

//precondition: need the users first and last name

//postcondition: Will store the customer’s name and set their starting balance to $0.

//description: A constructor to the BankAccount class, which creates a new bank account for a user, and sets their starting balance to $0.

public BankAccount (Customer c)

//precondition: Need the name of the customer

//postcondition: Will set the new customers starting balance to $0.

//description: A constructor to the BankAccount class, which creates a new bank account for a user, and sets their starting balance to $0.

public void deposit (double amount)

//precondition: amount must be > 0, in order to deposit any amount

//postcondition: adds the amount to the users current balance

//description: Will add the amount of money the user wants to deposit, into their bank account balance.

public void withdraw (double amount)

//precondition: Amount must be > 0 and the users current balance must be greater than or equal to the amount that the users wants to take out during the current transaction.

//postcondition: subtracts the amount from the users current balance

//description: Will subtract the amount of money the user wants to withdraw, from their bank account balance.

public void getBalance ()

//precondition: none

//postcondition: gets the users account balance

//description: Returns the users account balance

public void getCustomer ()

//precondition: none

//postcondition: gets the customer’s name

//description: Returns the customer’s full name.

public String toString ()

//precondition: Customers full name must be given

//postcondition: returns customers name and account balance.

//description: Returns the customers full name followed by their account balance in a string.

public Customer (String f, String l)

//precondition: Need the customers first and last name

//postcondition: Stores the customer’s name

//description: Stores the full name of the customer.

public String getFirstName ()

//precondition: Need the customers first name

//postcondition: returns customers first name

//description: returns customers first name

public String getLastName ()

//precondition: Need the customers last name

//postcondition: returns customers last name

//description: returns customers last name

public String toString ()

//precondition: Customers full name must be given

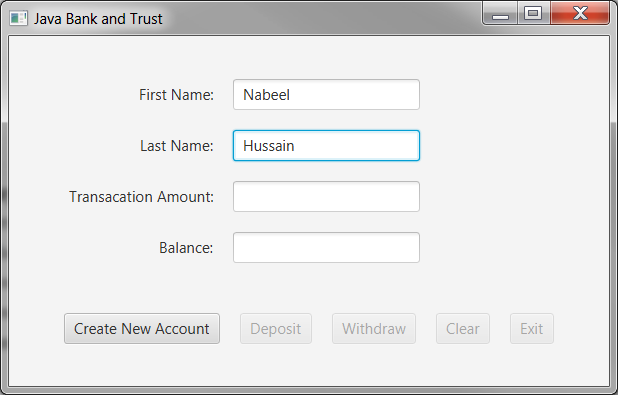
//postcondition: returns customers name

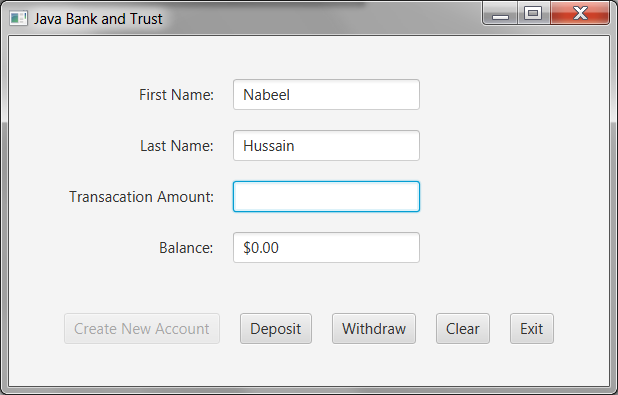
//description: Returns the customers full name in a string.

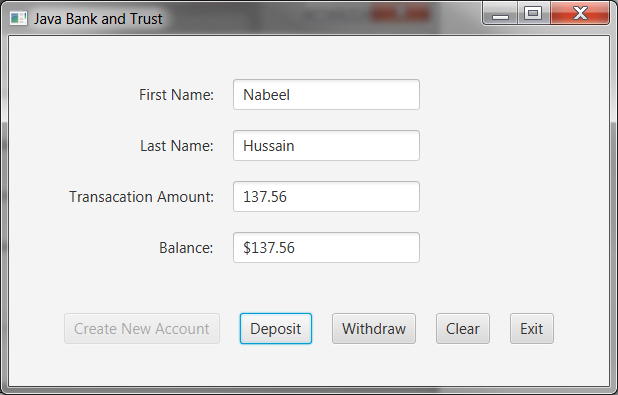
**Test table:**

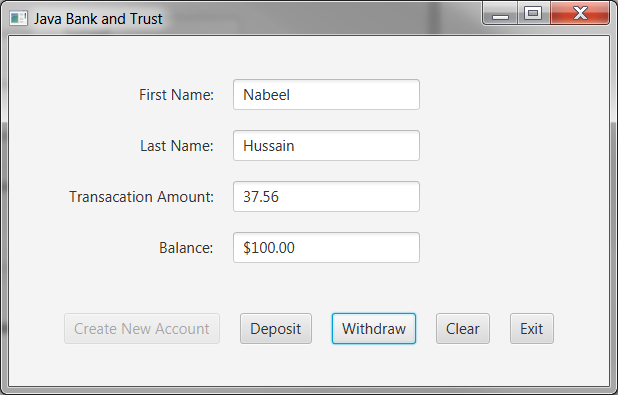
|  |  |  |
| --- | --- | --- |
| Input | Expected Output | Program runs correctly |
| First Name: Nabeel  Last Name: Hussain  Press “**Create New Account**” button | “First Name” textfield becomes uneditable  “Last Name” textfield becomes uneditable  “Create New Account” button becomes inactive  “Deposit,” “Withdraw,” “Clear;” “Exit” buttons become active.  “Transaction Amount” textfield becomes editable. |  |
| Transaction Amount: 255.69  Balance: 0  Press “**Deposit**” button | Balance: $255.69 |  |
| Transaction Amount: 55.69  Balance: $255.69  Press “**Withdraw**” button | Balance: $200.00 |  |
| Transaction Amount: 350.00  Balance: $200.00  Press “**Withdraw**” button | “You must enter a transaction amount that is less than your current balance, in order to withdraw money from your account. Please try again” |  |
| Transaction Amount: -50  Balance: $255.69  Press “**Withdraw**” button  Or  Press “**Deposit**” button | “You must enter a transaction amount greater than zero, for it to be withdrawn from your account. Please try again.”  Or  “You must enter a transaction amount greater than zero, for it to be deposited into your account. Please try again.” |  |
| Transaction Amount: -50  Balance: $255.69  Press “**Clear**” button | Transaction amount: |  |
| Transaction Amount:  Balance: $255.69  Press “**Exit**” button | Application exits |  |

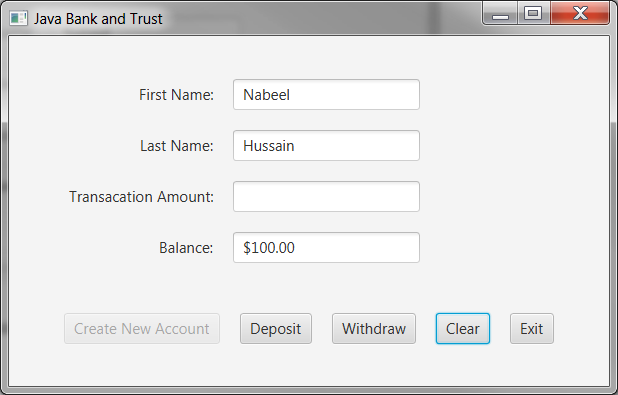
**Screenshots:**

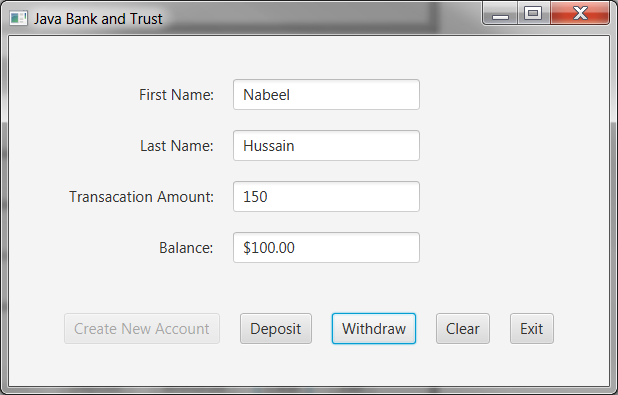
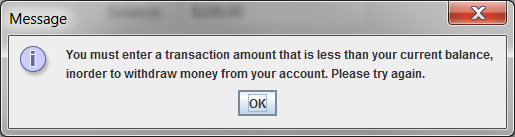


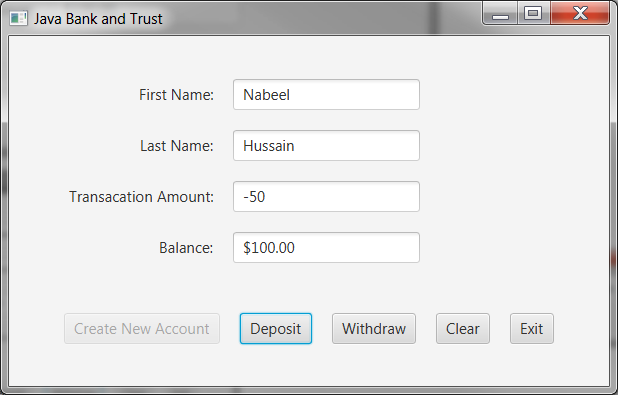
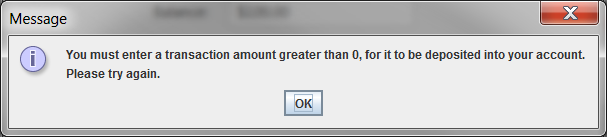












**Assumptions (if any)**

1. Transaction amount must always be greater than 0.

2. If the customer enters a negative number in the transaction amount field, then do not allow them to deposit or withdraw that amount. Instead display a jOption Pane asking them to re-enter a positive amount.

3. The transaction amount that is going to be withdrawn by the customer should not be more than their account balance. This is to ensure that an account balance does not reach a negative amount.

**Lessons Learned:**

**In 3+ paragraphs, highlight your lessons learned and learning experience from working on this project. How did you do? What have you learned? What did you struggle with? How will you approach your next project differently?**

While reading Chapter 12 and 15 of the textbook for module 4, there were a lot of new and interesting concepts that I learned which were applied in this project. This assignment also helped me reinforce the material I learned and gain a better understanding of the way a GUI is set up and works, using javafx , to create applications.

I learned many new things while working on this assignment, such as how to use the borderpane, VBox, and HBox methods to organize and structure the layout of my content in a GUI. I also learned how to add different types of buttons to an application, and how to make them functional using event handlers. One thing that I found cool was how to parse numbers that were entered into a string textfield, and convert it into an Integer value. In my opinion, I found using javafx much easier to create GUI applications compared to what I learned in chapter 12 about java swing. Although the concepts and ideas were similar, javafx seemed more straightforward to me, and I found it much easier to use.

I felt like I did pretty well in this assignment, and understood most all of the concepts that were needed to run this program correctly. There was a lot of new material that I had to re-read several times in order to get a good grasp of it, but once everything clicked, this assignment came easy to me. One thing I initially struggled with was how to properly use the padding and spacing features to accurately place my labels and buttons in the correct orientation, but after fiddling around with it, I got the hang of it.

I will approach my next assignment the same way I approached all the previous ones. I was able to manage my time wisely and start on the assignment very early, allowing me to finish everything up ahead of schedule. Overall, I found this assignment as the most difficult so far in this course, as almost everything was brand new material for me. However, once I was able to comprehend the material, I found it not too difficult and pretty straightforward.