CMSC 203, Assignment 3

Fall 2015

**Concepts Utilized in this Project**

* Javadoc
* UML diagrams
* Java classes and aggregation
* UI design & implementation, including widgets such as:
  + Stage
  + Scene
  + Event programming
  + Radio Buttons
  + Check Boxes, Labels, Buttons

**Overview**

A small bank has recently hired you to design & implement an application which the bank will use to manage accounts and (account) transactions for its customers. A typical customer would be able to possess one or more checking, saving, money market, and/or certificate of deposit (CD) accounts with the bank. The customer would also be able to conduct typical banking transactions, such as:

* making deposits to their account
* withdrawing cash from their account
* obtaining account balances and such

Using Java, design and implement a GUI application for the bank to support its banking needs.

## Specifications & Requirements

You will need the following classes:

**Data Element – Customer class**

Create a Customer class. It will contain instance variables of type String for the first name and last name. Provide constructors, getters and setters. Add any additional methods as required. Follow the Javadoc given you.

**Data Element – BankAccount class**

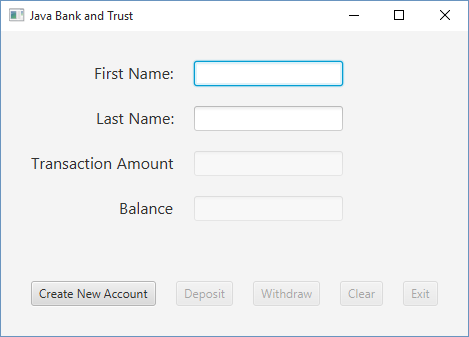
Create a BankAccount class that contains instance variables of a Customer and a double that holds the balance of the account. Provide constructors, getters and setters. You will need methods for deposit, withdraw and getting the account balance. Add any additional methods as required. Follow the Javadoc given you.

**GUI driver – JavaFX**

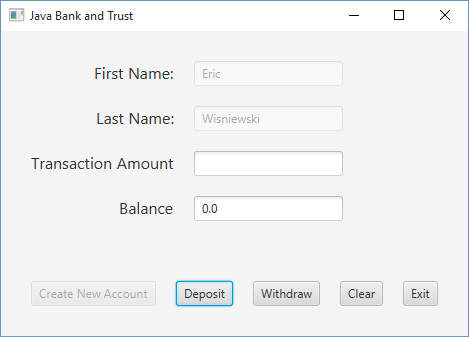
Include items such as labels, text boxes, buttons, dropdown lists, etc. Provide events/event handling for the following buttons:

* **Create** – extract the first name and last name from the text boxes, create a BankAccount object.
* **Deposit** – extract the amount from the text box, call the deposit method of the BankAccount class. Display the new balance by calling the method that returns the account balance.
* **Withdraw** - extract the amount from the text box, call the withdraw method of the BankAccount class. Display the new balance by calling the method that returns the account balance.
* **Clear** – clear the transaction amount text box
* **Exit** – exit the application

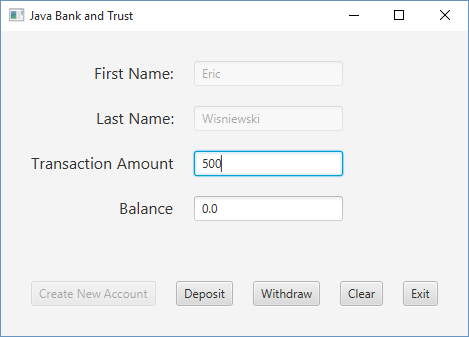
When starting the program, only firstname and lastname text boxes and create account button should be editable.



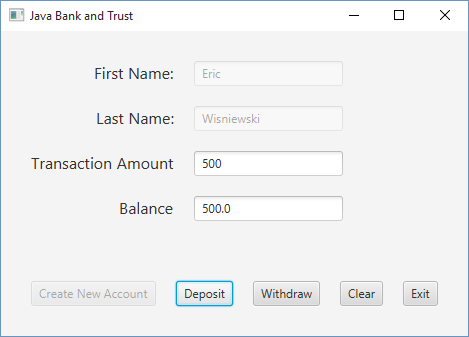
After you create the new account, the first and last name text boxes and the create account button become uneditable and the other buttons and text boxes become editable.



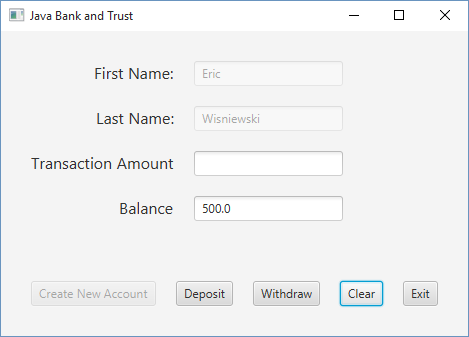
Enter the Transaction Amount



Select Deposit button



Select Clear button – clears the transaction amount text box



**Deliverables / Submissions:**

In addition to completing the Java application, Javadoc files and a write-up are required. Be sure to review the provided project rubric to understand project expectations. At a minimal, the write-up should include:

* Pseudo-code in English showing the steps to solve the problem
  + It must be detailed enough for someone else to write the code based on this algorithm
  + Complete this step first, and then write your code
* Test Cases
  + Prepare a test table with a list of test cases (expected versus actual results) that you are testing the application with
  + Same expectations as Assignment 1
* UML diagrams
* Any assumptions that you are making for this project
* In three or more paragraphs, highlights of your learning experience
* CheckList for Assignment 3

Each student must submit two compressed (.zip) files to the assignment’s folder on Blackboard for this project with the following contents:

* LastNameFirstName\_Assignment3\_Complete.zip, a compressed file in the zip format, with the following:
  + Write up (Word document)
  + Javadoc (directory)
    - File1.html (example)
    - File2.html (example)
  + src (directory) 
    - File1.java (example)
    - File2.java (example)
* LastNameFirstName\_Assignment3\_Moss.zip, a compressed file containing one or more Java files:
  + File1.java (example)
  + File2.java (example)
  + This folder should contain Java source files only

NAME\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Assignment 3 Grading Sheet

**EXTERNAL DOCUMENTATION** 24 (total)

CheckList for Assignment 3 is included and completed 1 pt \_\_\_\_\_

Algorithm Design

List ordered steps in English in to solve the problem 8 pts \_\_\_\_\_

(detailed enough for someone else to write the code based on this algorithm)

Test Cases

Proper Format (Inputs, Expected Output, Runs Correctly) 2 pts \_\_\_\_\_

4 test cases with valid data (unique input data for each case) 6 pts \_\_\_\_\_

Assumptions (if any) 2 pts \_\_\_\_\_

Lessons Learned 5 pt \_\_\_\_\_

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?

**PROGRAMMING** 76 (total)

Programming Style

Consistent indentation is used 2 pts \_\_\_\_\_

Appropriate identifiers are used 2 pts \_\_\_\_\_

Internal class documentation (within source code)

Description of what class does 6 pts \_\_\_\_\_

Author’s Name 2 pts \_\_\_\_\_

Appropriate comments within each section of the code 6 pts \_\_\_\_\_

Java file compiles and runs 8 pts \_\_\_\_\_

Accurate output from additional test cases 20 pts \_\_\_\_\_

Program user interface

Clear to user how data is to be entered 4 pts \_\_\_\_\_

Output is easy to understand 4 pts \_\_\_\_\_

Data Validation (amounts > 0) 4 pts \_\_\_\_\_

Program Details

Bank Account

Constructors, getters and setters, instance variables 5 pts \_\_\_\_\_

Customer

Constructors, getters and setters, instance variables 5 pts \_\_\_\_\_

User Interface

Event handlers 5 pts \_\_\_\_\_

Display the account total 3 pts \_\_\_\_\_