MAY 4, 2020

SOFTWARE DEVELOPMENT -ASSIGNMENT 2

CREDIT UNION

ROBERT KACSO - B00123508

Table of Contents

DECLARATION OF OWN WORK **2**

ABOUT THE ASSIGNMENT **3**

THE CODE **4**

THE RECORD CLASS 4
THE CREATERANDOMFILE CLASS 8
THE READWRITE CLASS 9

PROBLEMS ENCOUNTERED **29**

1ST PROBLEM – DUPLICATE ACCOUNTS 29
2ND PROBLEM – LESS BLANK FILES THAN INTENDED 31
3RD PROBLEM – FIELDS DIRECTLY EDITABLE 36
4TH PROBLEM – NEGATIVE INPUT 39
5TH PROBLEM – DELETE METHOD 46

METHODS **52**

ASS2RECORD CLASS METHODS 52
public void read(RandomAccessFile file) method *52*
public void write(RandomAccessFile file) method *52*
Methods to set and get details from the text field *53*
public static in size() method *53*
ASS2CREATERANDOMFILE CLASS METHODS 53
public Ass2CreateRandomFile () constructor *53*
ASS2READWRITE CLASS METHODS 54
Ass2ReadWrite() constructor *54*
public double paneInput (String tmpInput) method *57*
public void accountDetails(DecimalFormat df2) method *57*
public void updateAccount() method *58*
public void addRecord() method *58*
public void readRecordByInput() method *59*
public void lodgement() method *59*
public void withdrawal() method *60*
public void reqOverdraft() method *61*
public void delete() method *61*
public void editTextFields() method *62*
public void resetTextFields() method *62*
public void closeFile() method *63*
public void actionPerformed() method *63*
Main method *65*

ILLUSTRATION OF THE PROGRAM RUNNING **65**

INTERFACE 65
Create new account *65*
Select existing account *67*
Lodgement *69*
Withdrawal *71*
Overdraft *74*
Overdrawing and deleting an account *76*
Clear Textfiled *85*
Exit program *87*

CONCLUSION **87**

Declaration Of Own Work

Student Name:	**Robert Kacso**
Student Number:	**B00123508**
Course Title:	**Software Development 2**
Lecturer:	**Ann Marie Cosgrave**
Title of Work:	**Ass2B00123508cu**
Set Submission Date:	**04 May 2020**

I confirm that all this work is my own and that I have:

- Clearly referenced/listed all sources as appropriate
- Given the sources of all pictures, data etc. that are not my own
- Any work of any other student(s) either past or present is clearly referenced
- Not submitted for assessment work previously submitted for any other course, degree or qualification
- Not incorporated any text acquired from external agencies other than extracts from attributed sources (including online facilities)
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- My work may be electronically checked for plagiarism, including, but not exclusively, by the use of plagiarism detection software and stored for future comparison
- Any false claim for this work will be penalized in accordance with the College's regulations

Signature …….…… **Date** …….…….

SD2Ass2B00123508cu

About the assignment

Background
This assignment will test your knowledge of

- Objects
- Input and Output Streams
- Files
- Graphical user interface

Specification
Design and implement a class called **CreditUnion** that manages user accounts using an appropriate graphical user interface in Swing or AWT. Each instance of the **CreditUnion class** will have a name and a maximum number of accounts that it can handle.

The **CreditUnion** class should have the following methods:

- **openAccount** (allow a user to open a new Credit Union account)
- **closeAccount** (allow an account holder to close an account on condition that the balance is currently zero)
- **makeLodgement** (allow an account holder to lodge money)
- **makeWithdrawal** (allow an account holder to withdraw money from their account)
- **requestOverdraft** (allow an account holder to request a new overdraft limit).

Each **CreditUnion Account** should have a number, a name, a current balance and an overdraft limit. The methods for each **account** are:

- **lodgement** (make a lodgement)
- **withdrawal** (remove a sum of money from the account on condition that there are sufficient funds available to meet the request)
- **balance** (check current balance)
- **overdraftLimit** (check current overdraft limit)
- **setNewOverdraft** (set the overdraft limit to some given value).

The Code

Last assignment I took screen shots of the code and presented them like that. Now I tried to paste the code in here and adjust after. This is much longer to do as every line of code have to be separately adjust to keep a proper indentation, plus the colours of the words in the code to be applied too. The advantage of this is that the code can be copied out from here and pasted into an IDE directly.

The Record class

Used to create a RandomAccessFile to which the user can write too and read from. Contains methods to get and set details to the record. Depending on the amount of details that the user would like to input to each individual file of the record class, a size in byte is allocated to that file. In our case is 80 bytes. That is 4 bytes for the int, 8 bytes per double, 2 bytes per character per String. We have 1 int, 2 doubles and 2 Strings of 15 characters each. That's 4+(2*8)+2*(2*15) = 4+16+2*30 = 4+16+60 = 20+60=80

```
/*
Programmer: Robert Kacso
Student ID: B00123508
Note: Software Development 2, Assignment 2 - Credit Union program
Due date: 27/04/2020
*/

import java.io.*;

// main class to create a record
public class Ass2Record
{
   // declaring variables
   private int accountNo, activeAccounts;
   private String lastName;
   private String firstName;
   private double balance, overdraft;
   public RandomAccessFile randAccFile;

   //==============================================================================
   // method to read a record with a input/output exception
   public void read( RandomAccessFile file ) throws IOException
   {
      accountNo = file.readInt(); // initialize variable accountNo and read it in
```

```
    //----------------------------------------
            // array for the first name of max 15 characters
    char[] first = new char[15];

            // "for" loop to read in all characters
    for ( int i=0; i < first.length; i++ )
       first[ i ] = file.readChar();

            // initialize variable firstName and read it in
    firstName = new String (first);

    //----------------------------------------
            // array for the last name of max 15 characters
    char[] last = new char[15];

            // "for" loop to read in all characters
    for (int i =0; i<last.length; i++)
       last[i] = file.readChar();

            // initialize variable lastName and read it in
    lastName = new String (last);

    //----------------------------------------
            // initialize variable balance and read it in
    balance = file.readDouble();

    //----------------------------------------
            // initialize variable overdraft and read it in
    overdraft = file.readDouble();
  }


//==========================================================================
  // method to read a record with a input/output exception
  public void write( RandomAccessFile file) throws IOException
  {
    StringBuffer buf; // declaring StringBuffer "buf" to append, reverse,
                                    replace, concatenate and manipulate Strings or sequence
```

```
                                    of characters

file.writeInt( accountNo ); // write account number inputted to file in the
                                                        record

//----------------------------------------
// if/else statement to help append, reverse, replace, concatenate and
        manipulate variable firstName type Strings of max 15 characters
if (firstName != null)
   buf = new StringBuffer( firstName);
else
   buf = new StringBuffer( 15 );

        // buffer length set to 15 characters
buf.setLength( 15 );

        // write firstName inputted to file in the record
file.writeChars( buf.toString() );


//----------------------------------------
// if/else statement to help append, reverse, replace, concatenate and
        manipulate variable lastName type Strings of max 15 characters
if ( lastName != null )
   buf = new StringBuffer( lastName );
else
   buf = new StringBuffer( 15);

        // buffer length set to 15 characters
buf.setLength( 15 );

        // write lastName inputted to file in the record
file.writeChars( buf.toString() );

//----------------------------------------
        // write balance inputted to file in the record
file.writeDouble( balance );

//----------------------------------------
```

```
            // write overdraft inputted to file in the record
    file.writeDouble( overdraft );    }

  //==========================================================================
  // methods to set and get details from the record
  public void setAccountNo(int a ) { accountNo = a; }
  public int getAccountNo() { return accountNo;}

  public void setFirstName( String f) {firstName = f;}
  public String getFirstName() { return firstName; }

  public void setLastName ( String l) { lastName = l; }
  public String getLastName() { return lastName; }

  public void setBalance( double b) { balance = b; }
  public double getBalance() { return balance; }

  public void setOverdraft( double o) { overdraft = o; }
  public double getOverdraft() { return overdraft; }

  //-------------------------------------
  //determines size (bytes) of each file
  public static int size() { return 80;}
```

Here I created a method to read data from the random file. I am aware that data.read(randAccFile); does the same, but I wanted to practice and have options later in the program, which I am using in the updateDetails() method, to see if it is working.

```
 //==========================================================================
     // method to bring in data from the random access file created (a second option
  for "data.read(randAccFile)" method)
  // used for updating the details of the account holder in the updateAccount
  method situated in Ass2ReadWrite class
  public RandomAccessFile randomAccessFile()
  {
    try
    {
              // instance of the randAccFile variable
      randAccFile = new RandomAccessFile("creditUnion.dat", "rw");
```

```
        }
        catch (Exception e)
        {
            System.out.println("Error loading record " + e.toString());
            System.exit(1);
        }
                // return (output) the result of the method
        return randAccFile;
    }


}
```

This class is used to create the size of the RandomAccessFile. In our case 100 files.

The CreateRandomFile class

```
/*
Programmer: Robert Kacso
Student ID: B00123508
Note: Software Development 2, Assignment 2 - Credit Union program
Due date: 27/04/2020
*/

import java.io.*;

// main class to create random access file
public class Ass2CreateRandomFile
{
    // constructor
    public Ass2CreateRandomFile()
    {
        Ass2Record blankRec = new Ass2Record(); //instance of variable "blankRec"

        try
        {
                // instance of the randAccFile variable
            RandomAccessFile randAccFile = new RandomAccessFile("creditUnion.dat",
                                                                                "rw");
```

```
            // start seeking for empty file in the record at position 0
        randAccFile.seek(0);

            // "for" loop to create 100 empty files onto the record
        for (int i=0; i<100; i++)
            blankRec.write(randAccFile);
    } // end try

    catch(IOException e)
    {
        System.err.println("File not opened properly\n" + e.toString() );
        System.exit( 1 );
    } // end catch
  }

  // main method that runs constructor to create a random access file of 100
  empty files
  public static void main( String[] args )
  {
    new Ass2CreateRandomFile();
  }
}
```

This is the main class that we use to run the program and it contains the GUI as well as all the methods to write to the files of the record, read from the files of the record, lodge, withdraw, request overdraft, update details, delete, close programme and a few more that help the entire process, which I'll describe further down this document.

The ReadWrite class

```
/*
Programmer: Robert Kacso
Student ID: B00123508
Note: Software Development 2, Assignment 2 - Credit Union program
Due date: 27/04/2020
*/

import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
```

```
import java.text.DecimalFormat;

// main class extending JFrame for GUI and implementing ActionListener for the actionPerformed method
public class Ass2ReadWrite extends JFrame implements ActionListener
{
   //==================================================================================
   // declaring variables

   // variable randAccFile for all inputs & outputs
   private RandomAccessFile randAccFile;
   // declare instance variable "data" of class Ass2Record
   private Ass2Record data;

   //---------------------------------------
   // create GUI components

   // items in the menu bar
   private JMenuItem create,select,lodge,withdraw,overdraftReq, delete;
   // text fields for input
   private JTextField accountNoField, firstNameField, lastNameField, balanceField,
                  overdraftField;
   // buttons in the frame to clear the text fields and to close/end the program
   private JButton clear, done;

   //---------------------------------------
   // instance of the decimal format to show only 2 digits after the decimal in
   the balance and overdraft fields
   DecimalFormat df2 = new DecimalFormat( "0.00" );

   //==================================================================================
   // constructor - initialises the Frame
   public Ass2ReadWrite()
   {
      super("Credit Union"); // title of the frame

      // open file
      data = new Ass2Record(); //instance of variable "data"

      // access record
```

```
try
{
            // instance of variable randAccFile
   randAccFile = new RandomAccessFile("creditUnion.dat", "rw");
   System.out.println("In first try block");
} // end try

catch ( IOException e)
{
   System.err.println( e.toString() );
   System.exit(1);
} // end catch

//=========================================================================================
// dropdown menu learned it from
        https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

        // instance of JFrame "f"
JFrame f = new JFrame("Credit Union");
        // set layout of the frame
f.getContentPane().setLayout(new GridLayout(9,2));

JMenuBar mb = new JMenuBar(); // declare and initialize JMenubar
f.setJMenuBar(mb); // set the menu bar to the frame

//---------------------------------------
// menu and its submenus (new account and existing account)

        // declare and initialize JMenu
JMenu menu = new JMenu("Menu");
mb.add(menu); // add menu to the menu bar

        // declare and initialize JMenuItem
create=new JMenuItem("Create New Account");
        // key short cut alt+N
create.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
                                    InputEvent.ALT_DOWN_MASK));
create.addActionListener(this); // add action listener to "create"
        menu.add(create); // add create to menu
```

```
        // declare and initialize JMenuItem
select=new JMenuItem("Select an Existing Account");
        // key short cut alt+E
select.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
                                    InputEvent.ALT_DOWN_MASK));
select.addActionListener(this); // add action listener to "view"
menu.add(select); // add view to menu

//----------------------------------------
// lodge/withdraw and its submenus (lodge and withdraw)

        // declare and initialize JMenu
JMenu transactions = new JMenu("Account Transactions");
mb.add(transactions); // add transactions to the menu bar

        // declare and initialize JMenuItem
lodge=new JMenuItem("Lodgement");

        // key short cut alt+L
lodge.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L,
                                    InputEvent.ALT_DOWN_MASK));
lodge.addActionListener(this); // add action listener to "lodge"
transactions.add(lodge); // add lodge to transactions

        // declare and initialize JMenuItem
withdraw=new JMenuItem("Withdrawal");
        // key short cut alt+W
withdraw.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W,
                                    InputEvent.ALT_DOWN_MASK));
withdraw.addActionListener(this); // add action listener to "withdraw"
transactions.add(withdraw); // add withdraw to transactions

//----------------------------------------
// overdraftReq/delete and its submenus (overdraft request and delete
        account)

        // declare and initialize JMenu
JMenu reqDel = new JMenu("Account requests/deletion");
```

```
mb.add(reqDel); // add reqDel to the menu bar

        // declare and initialize JMenuItem
overdraftReq=new JMenuItem("Overdraft Adjustment");
        // key short cut alt+O
overdraftReq.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
                                                    InputEvent.ALT_DOWN_MASK));
overdraftReq.addActionListener(this); // add action listener
reqDel.add(overdraftReq); // add overdraftReq to reqDel

        // declare and initialize JMenuItem
delete=new JMenuItem("Delete Account");
        // key short cut alt+BACK_SPACE
delete.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_BACK_SPACE,
                                        InputEvent.ALT_DOWN_MASK));
delete.addActionListener(this); // add action listener to "delete"
reqDel.add(delete); // add delete to reqDel

//---------------------------------------
// labels that include image icons added to the frame
// all images created by me and downloaded for free online at
        https://logomakr.com/
f.getContentPane().add( new JLabel(
                                        new ImageIcon("creditUnionWelcome.png") ));

        f.getContentPane().add(new JLabel(
                                        new ImageIcon("creditUnionHereToHelp.png") ));

        f.getContentPane().add( new JLabel(
                                        new ImageIcon("creditUnionMenu.png") ));

        f.getContentPane().add(new JLabel(
                                        new ImageIcon("creditUnionExit.png") ));

//---------------------------------------
// text fields and its corresponding labels added to the JFrame "f"
f.getContentPane().add( new JLabel()); // empty JLabel for spacing
f.getContentPane().add( new JLabel()); // empty JLabel for spacing

f.getContentPane().add( new JLabel("Account Number") );
```

```
accountNoField = new JTextField();
accountNoField.setEditable(false);
f.getContentPane().add( accountNoField );

f.getContentPane().add( new JLabel("First Name") );
firstNameField = new JTextField();
firstNameField.setEditable(false);
f.getContentPane().add( firstNameField );

f.getContentPane().add( new JLabel("Last Name") );
lastNameField = new JTextField();
lastNameField.setEditable(false);
f.getContentPane().add( lastNameField );

f.getContentPane().add( new JLabel("Account Balance") );
balanceField = new JTextField();
balanceField.setEditable(false);
f.getContentPane().add( balanceField );

f.getContentPane().add( new JLabel("Overdraft Balance") );
overdraftField = new JTextField();
overdraftField.setEditable(false);
f.getContentPane().add( overdraftField );

//----------------------------------------
// buttons at the bottom of the frame to clear the text fields and to
        close/terminate the program
clear = new JButton ("Clear Textfield");
clear.addActionListener(this);
f.getContentPane().add (clear);

done = new JButton("Exit Program");
done.addActionListener(this);
f.getContentPane().add (done);

//----------------------------------------
f.setSize(450,500); // frame size
f.setVisible(true); // frame set to be visible
```

```
            // close the program by "X"-ing out the frame window
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

} // end constructor

//==================================================================================
// take input from JOptionPane
public double paneInput (String tmpInput)
{
    double input = Double.parseDouble(JOptionPane.showInputDialog(this,
                                tmpInput, "Input", JOptionPane.QUESTION_MESSAGE));

    try
    {
        // "if" statement to give an error message if a negative number is
                inputted
        if (input < 0)
        {
            JOptionPane.showMessageDialog(this, "Invalid negative input. Enter
                                                a number higher than 0", "Error message",
                                                    JOptionPane.ERROR_MESSAGE);
        }
    } // end try

    catch (Exception e)
    {
        JOptionPane.showMessageDialog(this, "Invalid negative input. Enter a
                                                number higher than 0", "Error message",
                                                JOptionPane.ERROR_MESSAGE);
    } // end catch

    return input;
} // end paneInput method

//==================================================================================
// method to read in the account details from the text fields and make the text
fields uneditable
public void accountDetails(DecimalFormat df2)
{
    accountNoField.setText(String.valueOf(data.getAccountNo()));
```

```
    accountNoField.setEditable(false);

    firstNameField.setText(data.getFirstName());
    firstNameField.setEditable(false);

    lastNameField.setText(data.getLastName());
    lastNameField.setEditable(false);

    balanceField.setText(String.valueOf(df2.format(data.getBalance())));
    balanceField.setEditable(false);

    overdraftField.setText(String.valueOf(df2.format(data.getOverdraft())));
    overdraftField.setEditable(false);
} // end accountDetails method

//==================================================================================
// method to update account details and write new data to file
public void updatedAccount()
{
    // declaring and initializing temporary variables which then could be
            seeked and modified
    int tmpAccount = Integer.parseInt( accountNoField.getText() );
    double tmpBalance = Double.parseDouble( balanceField.getText() );
    double tmpOverdraft = Double.parseDouble( overdraftField.getText() );


            try
    {
        // call in method randomAccessFile from the record class and initialize
                it to be equal to variable randAccFile
        randAccFile = data.randomAccessFile();
                //data.read( randAccFile ); // can be used too, just wanted to practice
                                                                extra methods
        // writing out the new inputted data to the text fields
        data.setAccountNo( tmpAccount );
        data.setFirstName( firstNameField.getText() );
        data.setLastName( lastNameField.getText() );
        data.setBalance( tmpBalance );
        data.setOverdraft( tmpOverdraft );
```

```
        // seek for the requested file in the record
        randAccFile.seek( (long) ( tmpAccount-1 ) * Ass2Record.size() );
        // write data to the seeked file
        data.write( randAccFile );
    } // end try

    catch (IOException e ) // input/output exception catch
    {
        System.err.println("Error writing record\n " + e.toString() );
        System.exit( 1 );
    } // end catch

} // end updateAccount method

//==========================================================================
// create method for adding records to file
public void addRecord()
{
    editTextFields();
    accountNoField.getText();
    {
        try
        {
            int tmpAccount = Integer.parseInt( accountNoField.getText() );

            System.out.println("In second try block");

                //validate account number is in range
            if (tmpAccount < 1 || tmpAccount > 100)
            {
                JOptionPane.showMessageDialog(this, "Account number must be
                                                                    between 1 & 100");
            }

            else
            {
                //read file to check if account number already exists.
                randAccFile.seek((long) (tmpAccount - 1) * Ass2Record.size());
                data.read(randAccFile);
```

```
                    // if account exists, display dialog box to user
            if (data.getAccountNo() == tmpAccount)
            {
               JOptionPane.showMessageDialog(this,"Account Number already
                                          exists! Please try a different Account Number");
                      // clear account textfield
               accountNoField.setText("");
            }

                    //once conditions are met, data is written to file.
                    else
            {
               updatedAccount(); // method called in to update the account
                                                   with the new input
               JOptionPane.showMessageDialog(this, "Account created and
                                                              details saved");
            }
         }

         resetTextFields(); // method called in to reset/clear textfield

      } // end try

      catch (NumberFormatException nfe ) // number format exception catch
      {
         JOptionPane.showMessageDialog(this,"Please input the details of the
                                          new account and click \"Create New Account\"
                                          again. \nIf necessary, reset the text
                                          fields by clicking on \"Clear Textfield\".");
      } // end catch

      catch (IOException io) // input/output exception catch
      {
         System.err.println("error during write to file\n" + io.toString());
      } // end catch

   }//end initial if statement
} // end addRecord method

//==========================================================================
```

```
// create method to read the record by inputting the account number used in
delete, lodge, withdraw and overdraft methods
public void readRecordByInput()
{
   try
   {
      int accountNumber = (int) paneInput("ENTER ACCOUNT NUMBER");

      if (accountNumber < 1 || accountNumber > 100)
      {
         JOptionPane.showMessageDialog(this, "WRONG INPUT!!! Input a number
                                                                between 1 and 100.");
      }

      else
         {
            // seek for the requested file in the record
            randAccFile.seek((accountNumber - 1) * Ass2Record.size());
            // read data from the seeked file
            data.read(randAccFile);

                        // "if" statement to reset/clear accountNoFiled if account
                        doesn't exist
            if (data.getAccountNo() == 0)
            {
               JOptionPane.showMessageDialog(this, "Account does not
                                                                                exist");
               accountNoField.setText("");
            }
            else
            {
                           // if account exists output the account details
               accountDetails(df2);
            }
         }

   }//end try

   catch (EOFException eof )
   {
```

```
        JOptionPane.showMessageDialog(this, "Account does not exist");
    } // end catch

    catch (IOException e )
    {
        System.err.println("Error during read from file\n " + e.toString() );
        System.exit( 1 );
    } // end catch
} // end readRecordByInput method method

//==============================================================================
// method to lodge money
public void lodgement()
{
    double lodgeAmount, oldBalance, newBalance;

    // variable oldBalance initialized to be equal to the current text from the
            balanceField
    // If the lodgement field is empty, the "catch" from if(e.getSource () ==
            lodge) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // prompting user to input new lodgement
    lodgeAmount = paneInput("Enter amount you want to lodge: ");

    // "if" statement to don't let the balance update with a negative number
    if (lodgeAmount > 0)
    {
        // arithmetic calculation of the new balance
        newBalance = lodgeAmount + oldBalance;
        // set balanceField to show the new balance
        balanceField.setText(String.valueOf(df2.format(newBalance)));
        JOptionPane.showMessageDialog(this, "Balance updated");
    }

            // method called in to update the account with the new input
    updatedAccount();
} // end lodgement method

//==============================================================================
```

```
// method for withdrawal()
public void withdrawal()
{
    int accept;
    double withdrawAmount, oldBalance, oldOverdraft;

    // variable oldBalance initialized to be equal to the current text from the
            balanceField
    // If the balance field is empty, the "catch" from if(e.getSource () ==
            withdraw) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // variable oldOverdraft initialized to be equal to the current text from
            the overdraftField
    oldOverdraft = Double.parseDouble(overdraftField.getText());

    // prompting user to input amount to withdraw
    withdrawAmount = paneInput("Enter amount you want to withdraw: ");

    // if/else statements for various scenarios...
    if (withdrawAmount < 0) // ... for inputting negative number
    {
        return;
    }
    else
    {
                // ... for withdrawal of an amount less then or equal to the current
                balance
        if ((oldBalance - withdrawAmount) >= 0)
        {
            balanceField.setText(String.valueOf(oldBalance - withdrawAmount));
            JOptionPane.showMessageDialog(this, "Balance updated");
        }

                // ... for withdrawal of an amount that puts account holder into
                overdraft
        else if ((oldBalance - withdrawAmount) >= (0 - oldOverdraft))
        {
            accept = JOptionPane.showConfirmDialog(this, "You accept going into
                                                                    overdraft?", "Warning Message",
```

```
                                                    JOptionPane.YES_NO_OPTION,
                                                    JOptionPane.WARNING_MESSAGE);
            // if user accepts to go into overdraft, the balance is updated
        if (accept == 0)
        {
            balanceField.setText(String.valueOf(oldBalance –
                                                withdrawAmount));
            JOptionPane.showMessageDialog(this, "Balance updated");
        }
    }

            // ... for withdrawal of an amount greater then the max overdraft
            allowed
    else if ((oldBalance - withdrawAmount) < (0 - oldOverdraft))
    {
        JOptionPane.showMessageDialog(this, "Sorry, not enough funds. Try a
                                            lesser amount.", "Error message",
                                            JOptionPane.ERROR_MESSAGE);
    }
  }

          // method called in to update the account with the new input
  updatedAccount();
} // end withdrawal method

//=================================================================================
// method to request change for overdraft
public void reqOverdraft()
{
  double newOverdraft;

  // Needed here EVEN if it's result is ignored by the program.
  // If the overdraft field is empty, the "catch" from if(e.getSource () ==
          overdraftReq) of the actionPerformed method will come in effect.
  // Without Double.parseDouble(overdraftField.getText()); from bellow, the
          method would ask to input a new overdraft even if the account field is
          empty, rendering the process useless.
  Double.parseDouble(overdraftField.getText());

  // prompting user to input amount of new overdraft
```

```
    newOverdraft = paneInput("Enter New Overdraft Amount: ");

    if ( newOverdraft >=0 )
    {
        // set overdraftField to show the new overdraft
        overdraftField.setText(String.valueOf(df2.format(newOverdraft)));
        JOptionPane.showMessageDialog(this, "Overdraft amount updated");
    }

            // method called in to update the account with the new input
    updatedAccount();
} // end reqOverdraft method

//==============================================================================
// delete method to delete account info and set balances to 0
public void delete()
{
    try
    {
        // variable accountNumber, accountBalance and accountOverdraft
                initialized to be equal to the text from their respective fields
        // If the any of the fields are empty, the "catch" from if(e.getSource
                () == delete) of the actionPerformed method will come in effect.
        int accountNumber = Integer.parseInt( accountNoField.getText() );
        double accountBalance = Double.parseDouble( balanceField.getText() );
        double accountOverdraft = Double.parseDouble(overdraftField.getText());

        if (accountBalance < 0 && accountBalance > 0)
        {
            JOptionPane.showMessageDialog(this, "Account can't be deleted.
                            Please adjust balance and/or overdraft to be 0 and try again.");
        }

        else if (accountOverdraft > 0)
        {
            JOptionPane.showMessageDialog(this, "Account can't be deleted.
                            Please adjust balance and/or overdraft to be 0 and try again.");
        }

        else
```

```
    {
        // message prompting user to confirm deletion of the account
        int accept = JOptionPane.showConfirmDialog(this, "ARE YOU SURE YOU
                                                WANT TO DELETE THIS ACCOUNT?", "Warning Message",
                                                JOptionPane.YES_NO_OPTION,
                                                JOptionPane.WARNING_MESSAGE);
        // if option is accepted, reset all data in the file to null
        if (accept == 0)
        {
            data.setAccountNo( 0 );
            data.setFirstName( null );
            data.setLastName( null);
            data.setBalance( 0) ;
            data.setOverdraft( 0 );

            // seek for the requested file in the record
            randAccFile.seek( (long)
                                                ( accountNumber-1 ) * Ass2Record.size() );

                        // write new data to the seeked file
            data.write( randAccFile );

            JOptionPane.showMessageDialog(this, "All funds have been
                                    withdrawn and account has been deleted");

                        // method called in to reset/clear text fields
            resetTextFields();
        }
    }
  }//end try

  catch (IOException io)
  {
    System.err.println("error during write to file\n" + io.toString() );
  } // end catch

          // method called in to update the account with the new input
  updatedAccount();
} // end delete method
```

```
//==========================================================================
// method to set text fields to be editable
public void editTextFields()
{
   accountNoField.setEditable(true);
   firstNameField.setEditable(true);
   lastNameField.setEditable(true);
   balanceField.setEditable(true);
   overdraftField.setEditable(true);
} // end resetTextFields method

//==========================================================================
// method to reset/empty/clear text fields
public void resetTextFields()
{
   accountNoField.setText("");
   firstNameField.setText("");
   lastNameField.setText("");
   balanceField.setText("");
   overdraftField.setText("");
} // end resetTextFields method

//==========================================================================
// method to close file/end program
private void closeFile()
{
   try
   {
      // message to user when program is terminated
      JOptionPane.showMessageDialog(this,"Thank you for using CREDIT UNION! "
                                          + "\n            GOODBYE!");
      randAccFile.close();
      System.exit( 0 );
   } // end try

   catch (IOException io)
   {
      System.err.println( "File not closed properly\n" + io.toString() );
   } // end catch
```

```
    setVisible(false);
} // end closeFile method

//==========================================================================
// add actionPerformed for the buttons
public void actionPerformed ( ActionEvent e )
{
    //----------------------------------------
    // create account

            // when "create" JMenuItem is pressed
    if (e.getSource() == create)
    {
        addRecord(); // method called in
    }

    //----------------------------------------
    // read account details when account number entered

            // when "view" JMenuItem is pressed
    if (e.getSource() == select)
    {
        try
        {
            readRecordByInput(); // method called in
        } // end try

        catch ( NumberFormatException nfe ) // number format exception catch
        {
            JOptionPane.showMessageDialog(this,"Invalid input. Please enter an
                                                    integer number between 1 and 100.");
        } // end catch

    }

    //----------------------------------------
    //update balance by making a lodgement

            // when "lodge" JMenuItem is pressed
    if (e.getSource() == lodge)
```

```
{
   try
   {
      lodgement(); // method called in
   } // end try

   catch ( NumberFormatException nfe ) // number format exception catch
   {
      JOptionPane.showMessageDialog(this,"Go to \"Menu\" -> \"Select
                                 Existing Account\" and input an account number. " +
            "\nWhen the account details are displayed, click
                                 \"Lodgement\".");
   } // end catch
}

//----------------------------------------
// method to withdraw money from the account

         // when "withdraw" JMenuItem is pressed
if (e.getSource() == withdraw)
{
   try
   {
      withdrawal(); // method called in
   } // end try

   catch ( NumberFormatException nfe ) // number format exception catch
   {
      JOptionPane.showMessageDialog(this,"Go to \"Menu\" -> \"Select
                                 Existing Account\" and input an account number. " +
            "\nWhen the account details are displayed, click
                                 \"Withdrawal\".");
   } // end catch
}

//----------------------------------------
//method to request an overdraft adjustment

         // when "overdraftReq" JMenuItem is pressed
if (e.getSource() == overdraftReq)
```

```
{
   try
   {
      reqOverdraft(); // method called in
   } // end try

   catch ( NumberFormatException nfe ) // number format exception catch
   {
      JOptionPane.showMessageDialog(this,"Go to \"Menu\" -> \"Select
                                    Existing Account\" and input an account number. " +
            "\nWhen the account details are displayed, click
                                    \"Overdraft Adjustment\".");
   } // end catch
}

//----------------------------------------
//read account details when account number entered and delete account

        // when "delete" JMenuItem is pressed
if (e.getSource() == delete)
{
   try
   {
      delete(); // method called in
   } // end try

   catch ( NumberFormatException nfe ) // number format exception catch
   {
      JOptionPane.showMessageDialog(this,"Go to \"Menu\" -> \"Select
                                    Existing Account\" and input an account number. " +
            "\nWhen the account details are displayed, click \"Delete
                                    Account\".");
   } // end catch

}

//----------------------------------------
// clear textfield
```

```
            // when "clear" JButton is pressed
        if (e.getSource() == clear)
        {
            resetTextFields(); // method called in
        }

        //---------------------------------------
        // save last input and close program

            // when "done" JButton is pressed
        if (e.getSource() == done)
        {
            closeFile(); // method called in
        }
    }

    //==========================================================================
    //Instantiate a Ass2ReadWrite object and start the program by calling in the
    main method
    public static void main(String[] args )
    {
        Ass2ReadWrite app = new Ass2ReadWrite(); // calling in the constructor

            // assist with closing the window and terminate the program
            app.addWindowListener(new WindowAdapter()
            {public void windowClosing(WindowEvent e) {System.exit(0);}});
    } // end main method
}
```

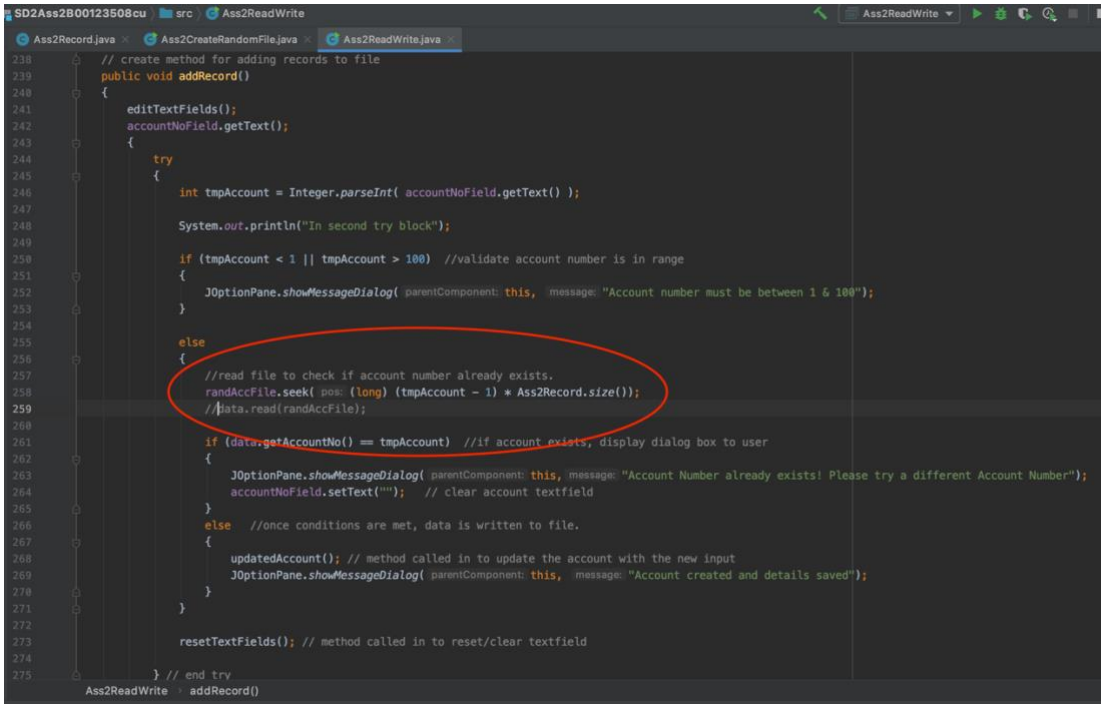
Problems encountered

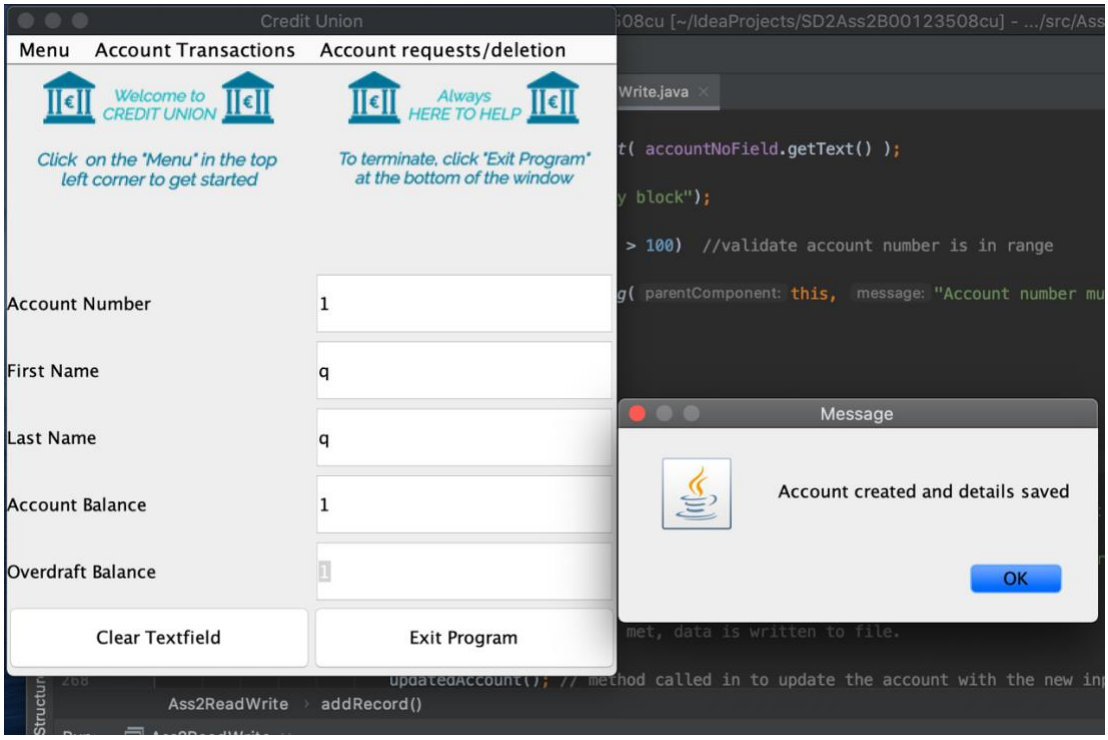
1st problem – duplicate accounts

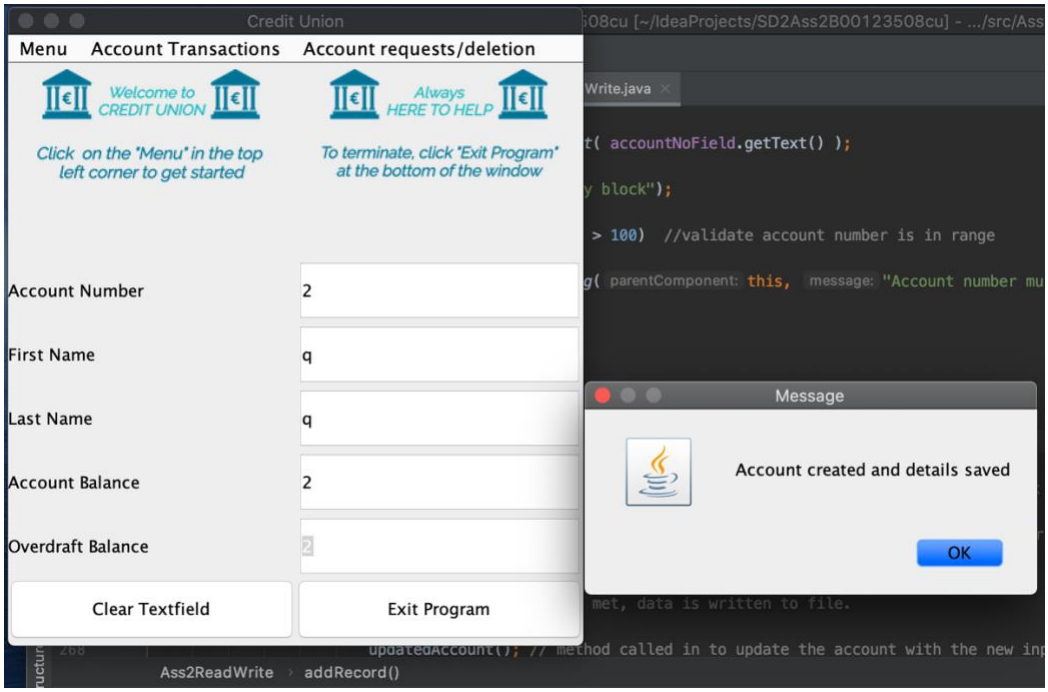
One of my main problems was that the program was allowing me to create duplicate files, as after I was seeking the RandomAccessFile, I wasn't reading it in too. I was missing a vital piece of code.
Here it is in the image bellow: "data.read(randAccFile);" was the code line I was missing.

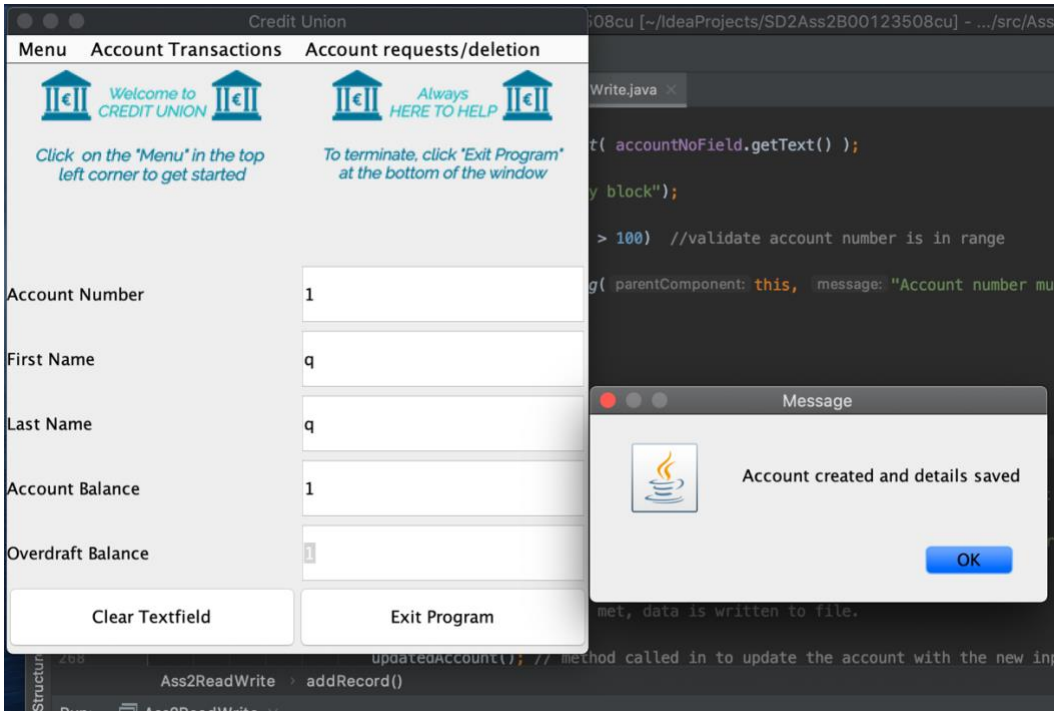
```
//read file to check if account number already exists.
randAccFile.seek( pos: (long) (tmpAccount - 1) * Ass2Record.size());
data.read(randAccFile);
```

This code is situated in the addRecord() method. This method writes out the details to the files from the blank record. I commented out the code for the purpose of this test.









As you can see above, I created twice the first account with the "data.read(randAccFile);" code commented out.

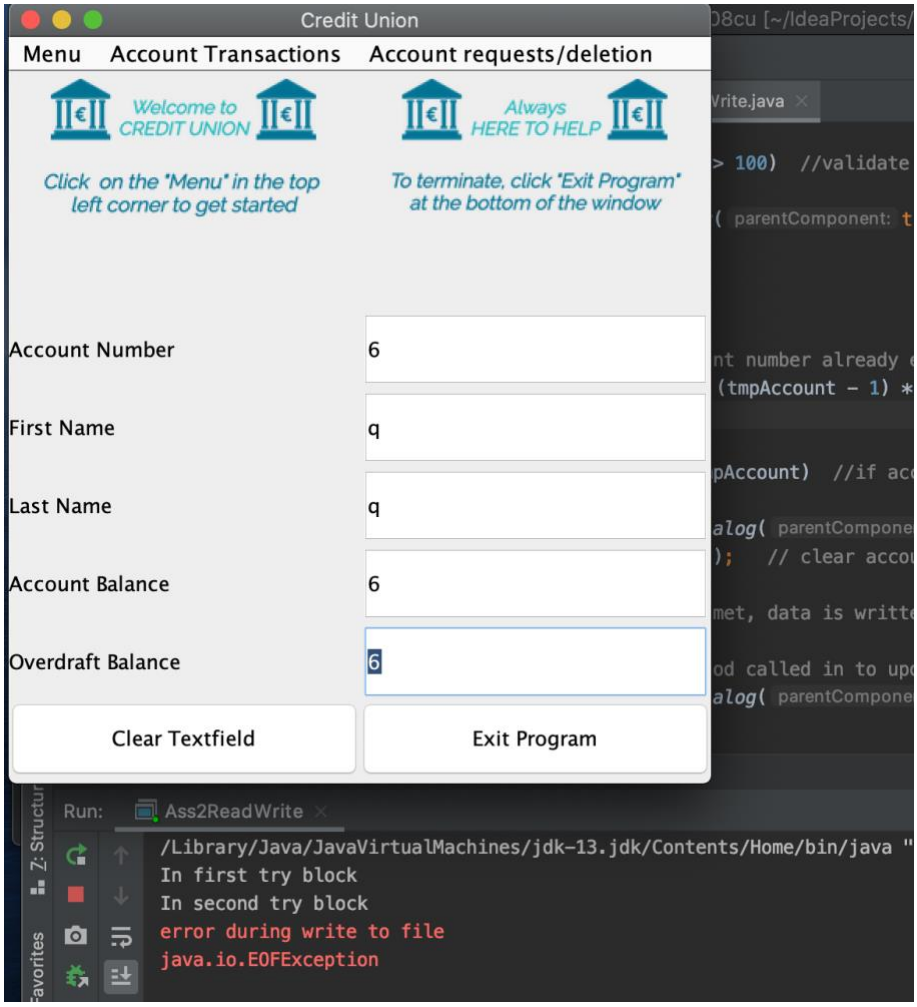
2nd problem – less blank files than intended

After I put in the code to read in the file, I couldn't put in more than 5 accounts, even though I have created 100 blank files. At least I thought I did. I tried absolutely everything and couldn't figure out why.

Check bellow to see the “for” loop that would create 100 blank files

```java
import java.io.*;

// main class to create random access file
public class Ass2CreateRandomFile
{
    // constructor
    public Ass2CreateRandomFile()
    {
        Ass2Record blankRec = new Ass2Record(); //instance of variable "blankRec"

        try
        {
            RandomAccessFile randAccFile = new RandomAccessFile( name: "creditUnion.dat",  mode: "rw"); // instance of the randAccFile variable
            randAccFile.seek( pos: 0); // start seeking for empty file in the record at position 0

            for (int i=0; i<100; i++) // "for" loop to create 100 empty files onto the record
                blankRec.write(randAccFile);
        } // end try

        catch(IOException e)
        {
            System.err.println("File not opened properly\n" + e.toString() );
            System.exit( status: 1 );
        } // end catch
    }

    // main method that runs constructor to create a random access file of 100 empty files
    public static void main( String[] args ) { new Ass2CreateRandomFile(); }
}
```

And here is the error that I get when trying to create a sixth account:



Eventually after much googling I have figured it out. The random access file that was created before I introduced the “data.read(randAccFile);” code into the addRecord() method, was only containing 5 accounts that I have created before, and not more. In fact I forgot to run the Ass2CreateRandomFile program that would actually create the 100 blank files.

In order to fix that issues, I had to recompile and run the Ass2CreateRandomFile, which would then generate the 100 blank files again.

```java
// main class to create random access file
public class Ass2CreateRandomFile
{
    // constructor
    public Ass2CreateRandomFile()
    {
        Ass2Record blankRec = new Ass2Record(); //instance of variable "blankRec"

        try
        {
            RandomAccessFile randAccFile = new RandomAccessFile( name: "creditUnion.dat",  mode: "rw"); // instance of the randAccFile variable
            randAccFile.seek( pos: 0); // start seeking for empty file in the record at position 0

            for (int i=0; i<100; i++) // "for" loop to create 100 empty files onto the record
                blankRec.write(randAccFile);
        } // end try

        catch(IOException e)
        {
            System.err.println("File not opened properly\n" + e.toString() );
            System.exit(  status: 1 );
        } // end catch
    }

    // main method that runs constructor to create a random access file of 100 empty files
    public static void main( String[] args ) { new Ass2CreateRandomFile(); }
}
```

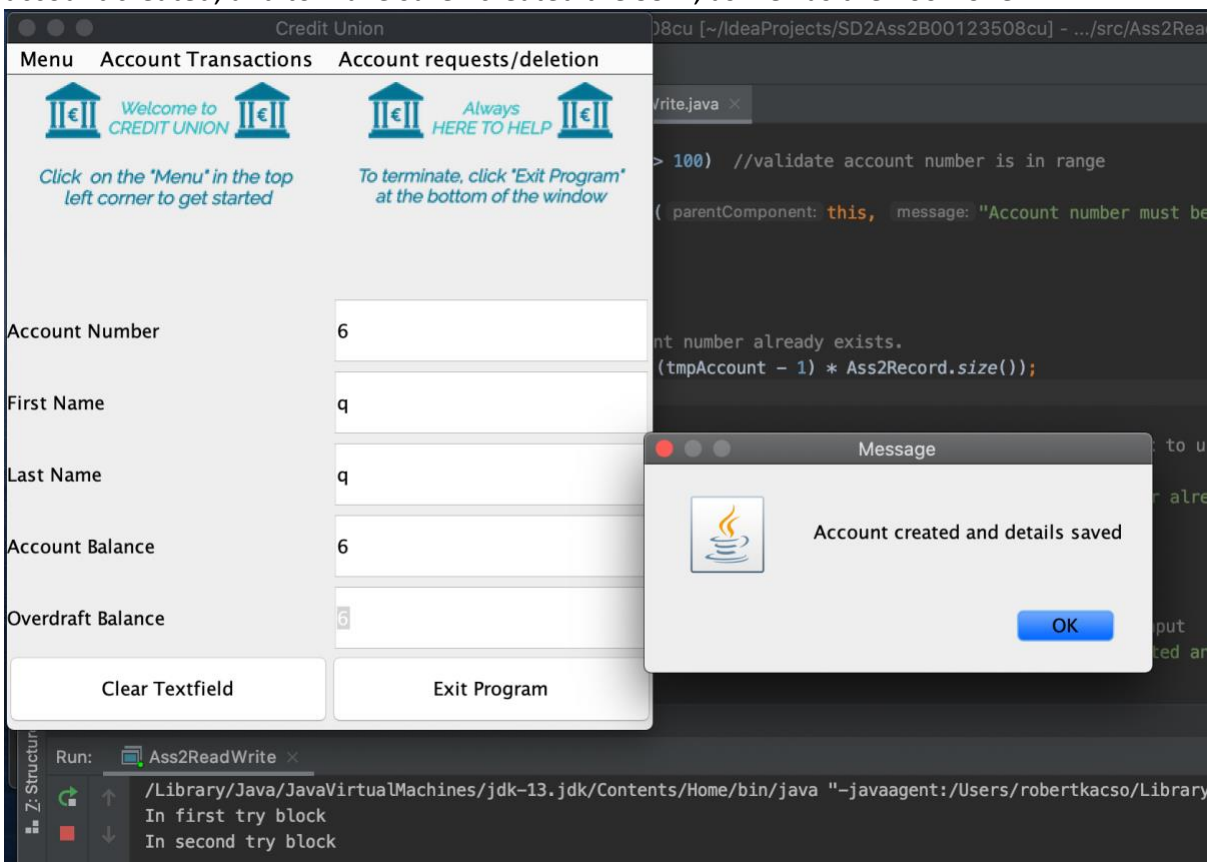
```
Ass2CreateRandomFile
Run: Ass2CreateRandomFile
/Library/Java/JavaVirtualMachines/jdk-13.jdk/Contents/Home/bin/java "-javaagent:/Users/robertkacso/Library/Application Support/JetBrains/Toolbox/apps/IDEA-U/ch-0/193.6

Process finished with exit code 0
```

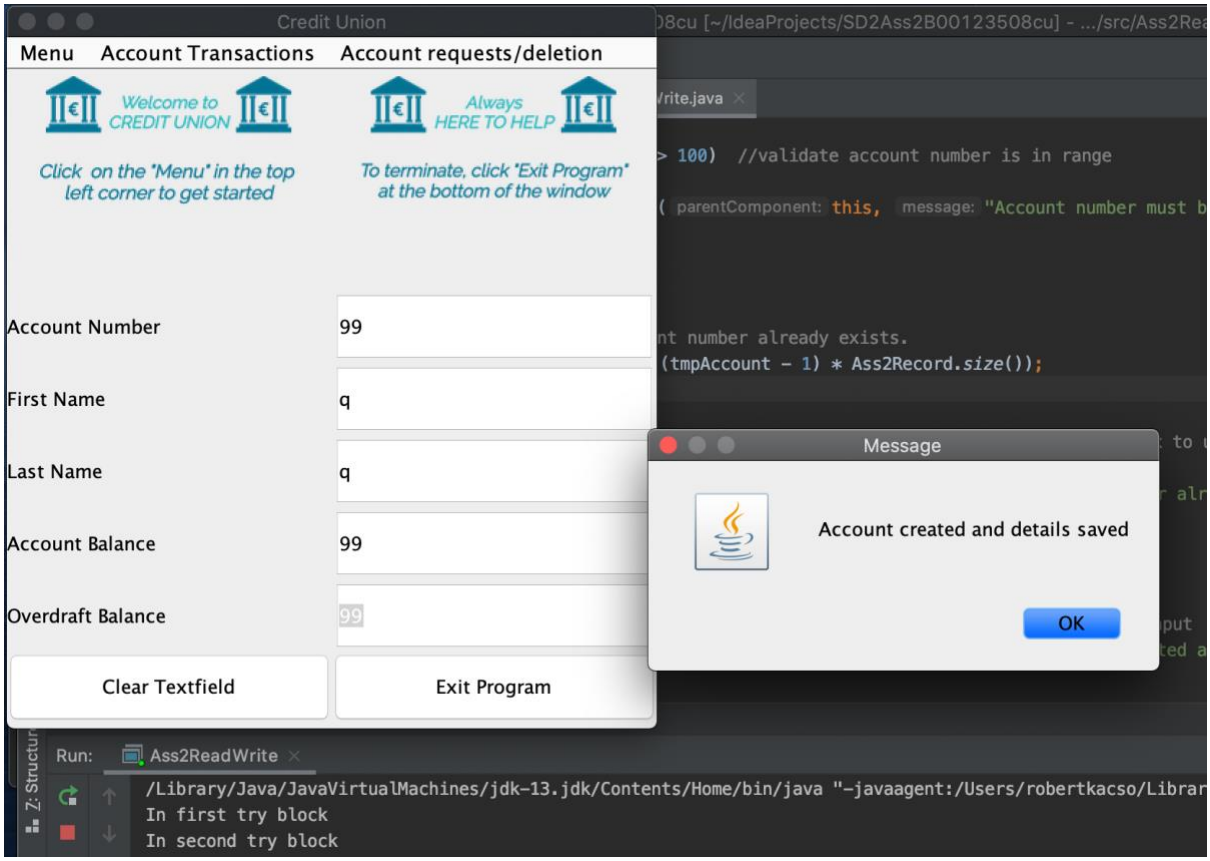
Here the “data.read(randAccFile);” is reintroduced:

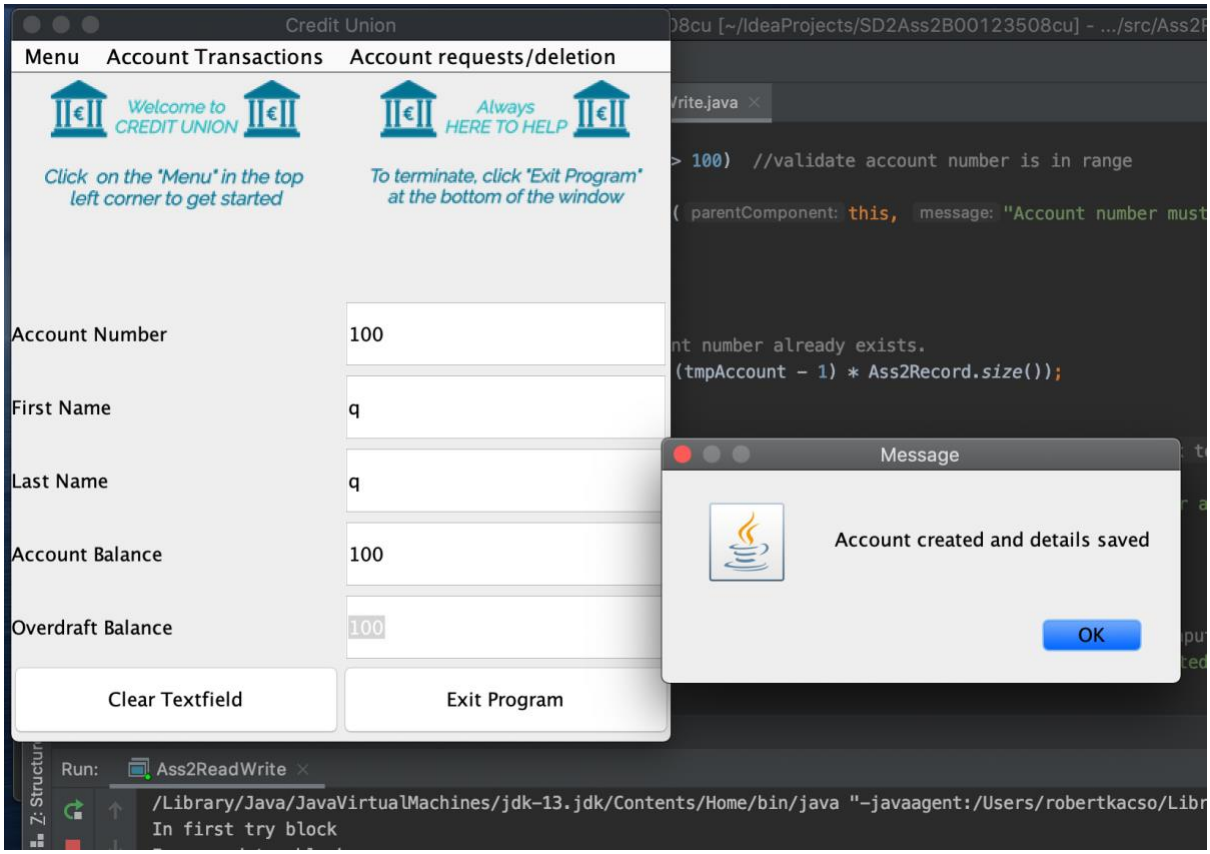
```java
// create method for adding records to file
public void addRecord()
{
    editTextFields();
    accountNoField.getText();
    {
        try
        {
            int tmpAccount = Integer.parseInt( accountNoField.getText() );

            System.out.println("In second try block");

            if (tmpAccount < 1 || tmpAccount > 100)  //validate account number is in range
            {
                JOptionPane.showMessageDialog( parentComponent: this,  message: "Account number must be between 1 & 100");
            }

            else
            {
                //read file to check if account number already exists.
                randAccFile.seek( pos: (long) (tmpAccount - 1) * Ass2Record.size());
                data.read(randAccFile);

                if (data.getAccountNo() == tmpAccount)  //if account exists, display dialog box to user
                {
                    JOptionPane.showMessageDialog( parentComponent: this, message: "Account Number already exists! Please try a different Account Number");
                    accountNoField.setText("");   // clear account textfield
                }
                else   //once conditions are met, data is written to file.
                {
                    updatedAccount(); // method called in to update the account with the new input
                    JOptionPane.showMessageDialog( parentComponent: this,  message: "Account created and details saved");
```

A simple thing as compiling and running that class, costed me about 3 hours.

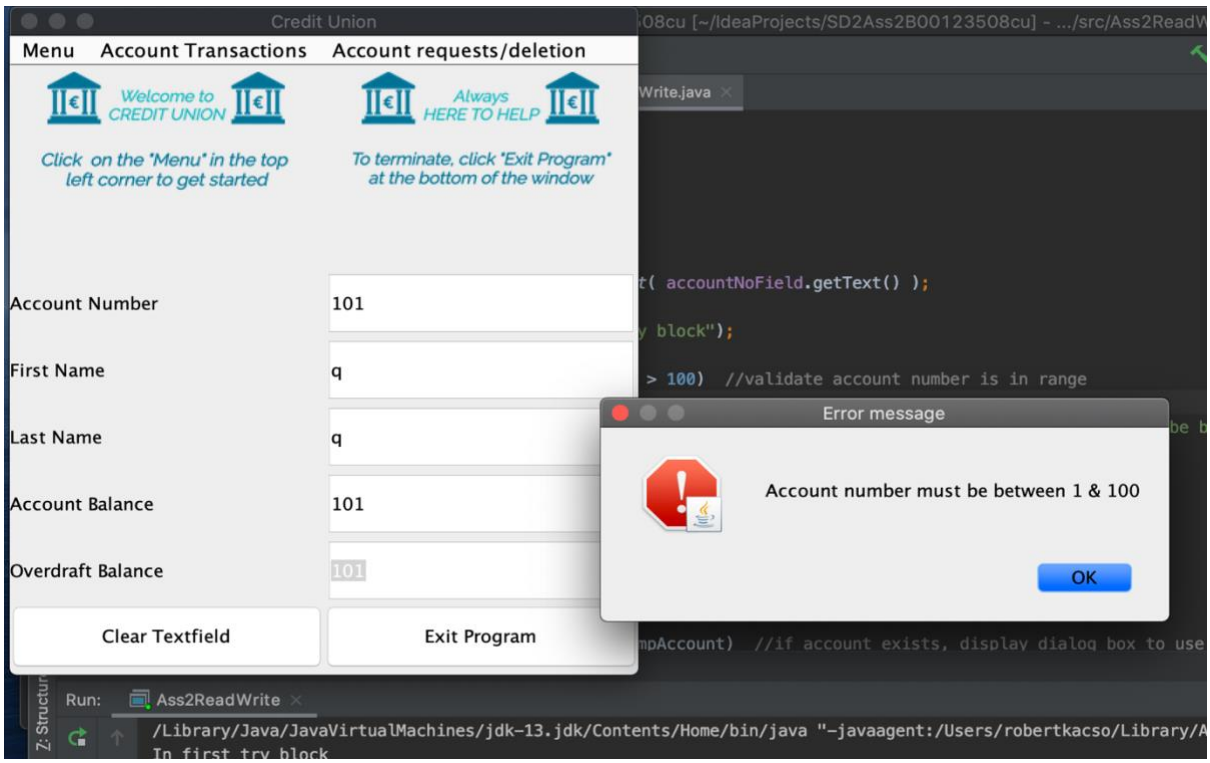
Once it created the new 100 blank files, I had no problem writing to them. Here is the sixth account created, and to make sure I created the 99th , as well as the 100th one.







Trying to see if it will let me to create the 101st account…



As you can see, it won't let me. It's telling me that the account numbers must be between 1 and 100. This problem is solved and it's all working fine.

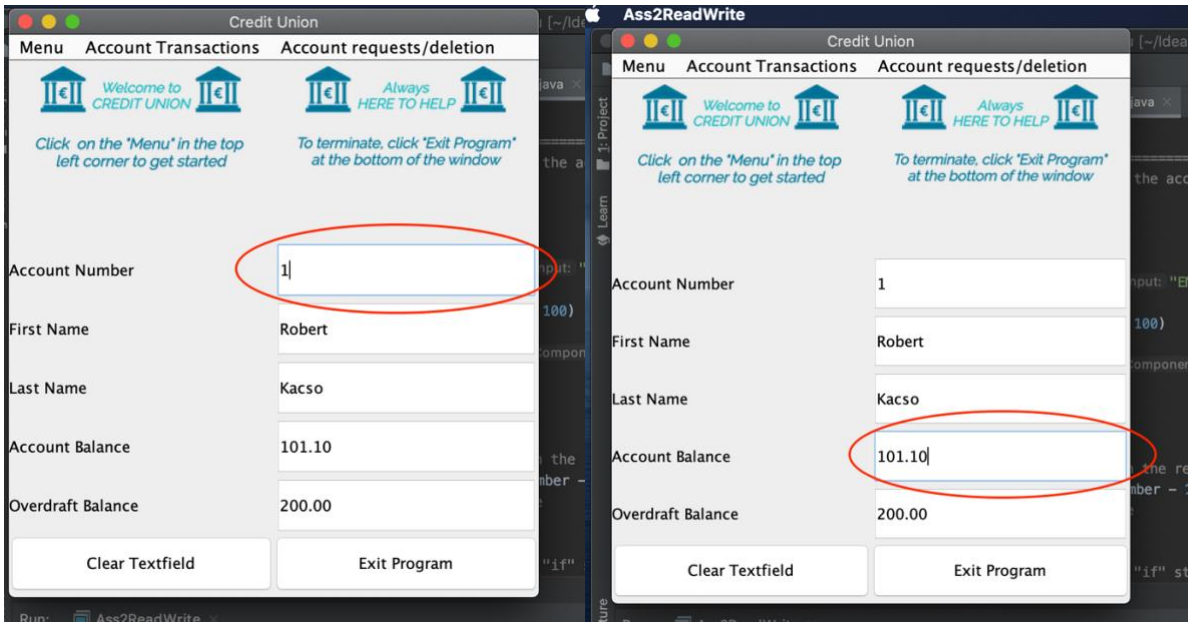
3rd problem – fields directly editable

Another problem that I had was with the lodgement, withdrawal and overdraft method. I was able to edit them directly in the text field, anytime I was calling them as my text fields were not set to “set.Editable(false)”. For the purpose of the demonstration, I commented out the text field .setEditable(false); and the method editTextFields() inside the addRecord() method.

```
//-----------------------------------------
// text fields and its corresponding labels added to the JFrame "f"
f.getContentPane().add( new JLabel()); // empty JLabel for spacing
f.getContentPane().add( new JLabel()); // empty JLabel for spacing

f.getContentPane().add( new JLabel( text: "Account Number") );
accountNoField = new JTextField();
//accountNoField.setEditable(false);
f.getContentPane().add( accountNoField );

f.getContentPane().add( new JLabel( text: "First Name") );
firstNameField = new JTextField();
//firstNameField.setEditable(false);
f.getContentPane().add( firstNameField );

f.getContentPane().add( new JLabel( text: "Last Name") );
lastNameField = new JTextField();
//lastNameField.setEditable(false);
f.getContentPane().add( lastNameField );

f.getContentPane().add( new JLabel( text: "Account Balance") );
balanceField = new JTextField();
//balanceField.setEditable(false);
f.getContentPane().add( balanceField );

f.getContentPane().add( new JLabel( text: "Overdraft Balance") );
overdraftField = new JTextField();
//overdraftField.setEditable(false);
f.getContentPane().add( overdraftField );
```

```
//=============================================================
// create method for adding records to file
public void addRecord()
{
    //editTextFields();
    accountNoField.getText();
    {
        try
        {
            int tmpAccount = Integer.parseInt( accountNoFie

            System.out.println("In second try block");

            if (tmpAccount < 1 || tmpAccount > 100)  //vali
            {
                JOptionPane.showMessageDialog( parentCompon
```

Here is the problem:



As you can see in the images above, every text field is editable, without having to call in any method for lodgement, withdrawal or overdraft adjustment.

To solve that, I made my text fields uneditable and I created a method called editTextFields(), that is being balled in at the beginning of the addRecord() method that will make the text fields editable again.

Here is the part of the GUI where the text fields were made uneditable:

```
//-----------------------------------------
// text fields and its corresponding labels added to the JFrame "f"
f.getContentPane().add( new JLabel()); // empty JLabel for spacing
f.getContentPane().add( new JLabel()); // empty JLabel for spacing

f.getContentPane().add( new JLabel( text: "Account Number") );
accountNoField = new JTextField();
accountNoField.setEditable(false);
f.getContentPane().add( accountNoField );

f.getContentPane().add( new JLabel( text: "First Name") );
firstNameField = new JTextField();
firstNameField.setEditable(false);
f.getContentPane().add( firstNameField );

f.getContentPane().add( new JLabel( text: "Last Name") );
lastNameField = new JTextField();
lastNameField.setEditable(false);
f.getContentPane().add( lastNameField );

f.getContentPane().add( new JLabel( text: "Account Balance") );
balanceField = new JTextField();
balanceField.setEditable(false);
f.getContentPane().add( balanceField );

f.getContentPane().add( new JLabel( text: "Overdraft Balance") );
overdraftField = new JTextField();
overdraftField.setEditable(false);
f.getContentPane().add( overdraftField );
```

And here is the editTextFields() method:

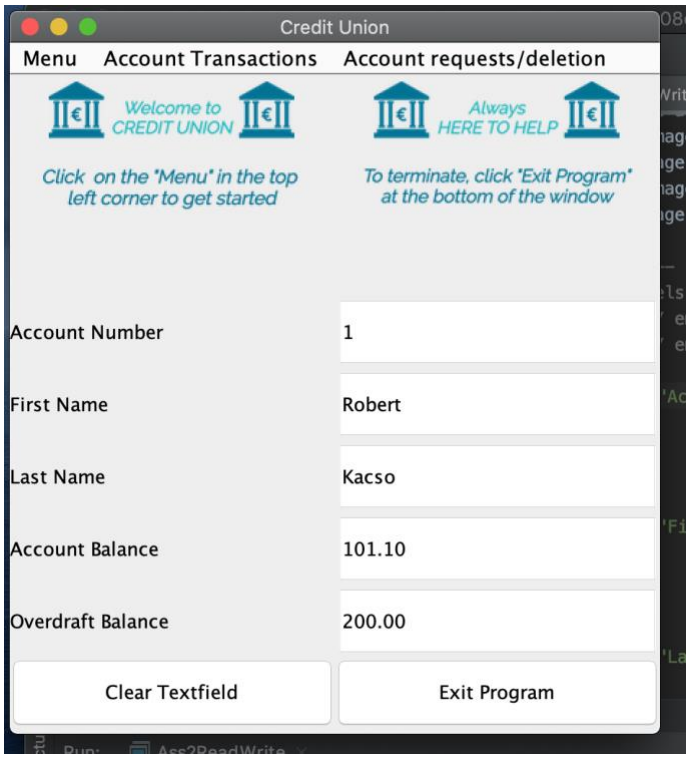
```
//==========================================================
// method to set text fields to be editable
public void editTextFields()
{
    accountNoField.setEditable(true);
    firstNameField.setEditable(true);
    lastNameField.setEditable(true);
    balanceField.setEditable(true);
    overdraftField.setEditable(true);
} // end resetTextFields method
```

Here you can see it added to the beginning of the addRecord() method, to make text fields editable:

```
//==========================================================================================
// create method for adding records to file
public void addRecord()
{
    editTextFields();
    accountNoField.getText();
    {
        try
        {
            int tmpAccount = Integer.parseInt( accountNoField.getText() );

            System.out.println("In second try block");

            if (tmpAccount < 1 || tmpAccount > 100)  //validate account number is in range
            {
                JOptionPane.showMessageDialog( parentComponent: this,  message: "Account number must be
            }

            else
            {
                //read file to check if account number already exists.
                randAccFile.seek( pos: (long) (tmpAccount - 1) * Ass2Record.size());
                data.read(randAccFile);
```

And the result is that the text fields can not be edited, only if you call in the appropriate methods to do so.

Here it shows the text fields not being editable:



4th problem – negative input

Within the same methods I had another problem. I had created a method called paneInput(), that basically takes the input from a JOptionPane and returns it. This method has an "if" statement warning the user that a negative number has been inputted. That's all good, as we don't want to lodge negative numbers. That's basically withdrawing. And the same we don't want to withdraw negative numbers, as that is basically lodging. Of course same goes for the overdraft.

Let's have a look at it:

```
//==========================================================================
// take input from JOptionPane
public double paneInput (String tmpInput)
{
    double input = Double.parseDouble(JOptionPane.showInputDialog( parentComponent: this, tmpInput,  title: "Input", JOptionPane.QUESTION_MESSAGE));

    try
    {
        // "if" statement to give an error message if a negative number is inputted
        if (input < 0)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Invalid negative input. Enter a number higher than 0",  title: "Error message", JOptionPane
        }
    } // end try

    catch (Exception e)
    {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Invalid negative input. Enter a number higher than 0",  title: "Error message", JOptionPane.ER
    } // end catch

    return input;
} // end paneInput method
```

All good except that this doesn't actually stop the user to lodge in, let's say -2 euros and the balance would still get updated by minus 2.

Here is the lodgement() and reqOverdraft methods with an "if" statement commented out for the purpose of this demonstration. That "if" statement going to fix the problem. The withdrawal() method was sorted by an "if statement too, only that it didn't wrap around a .setText code. Instead it gave a return. Let's check lodgement(), reqOverdraft() and withdraw() methods:

```
//==========================================================================================
// method to lodge money
public void lodgement()
{
    double lodgeAmount, oldBalance, newBalance;

    // variable oldBalance initialized to be equal to the current text from the balanceField
    // If the lodgement field is empty, the "catch" from if(e.getSource () == lodge) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // prompting user to input new lodgement
    lodgeAmount = paneInput( tmpInput: "Enter amount you want to lodge: ");

    // "if" statement to don't let the balance update with a negative number
    //if (lodgeAmount > 0)
    //{
        // arithmetic calculation of the new balance
        newBalance = lodgeAmount + oldBalance;
        // set balanceField to show the new balance
        balanceField.setText(String.valueOf(df2.format(newBalance)));
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Balance updated");
    //}

    updatedAccount(); // method called in to update the account with the new input
} // end lodgement method
```

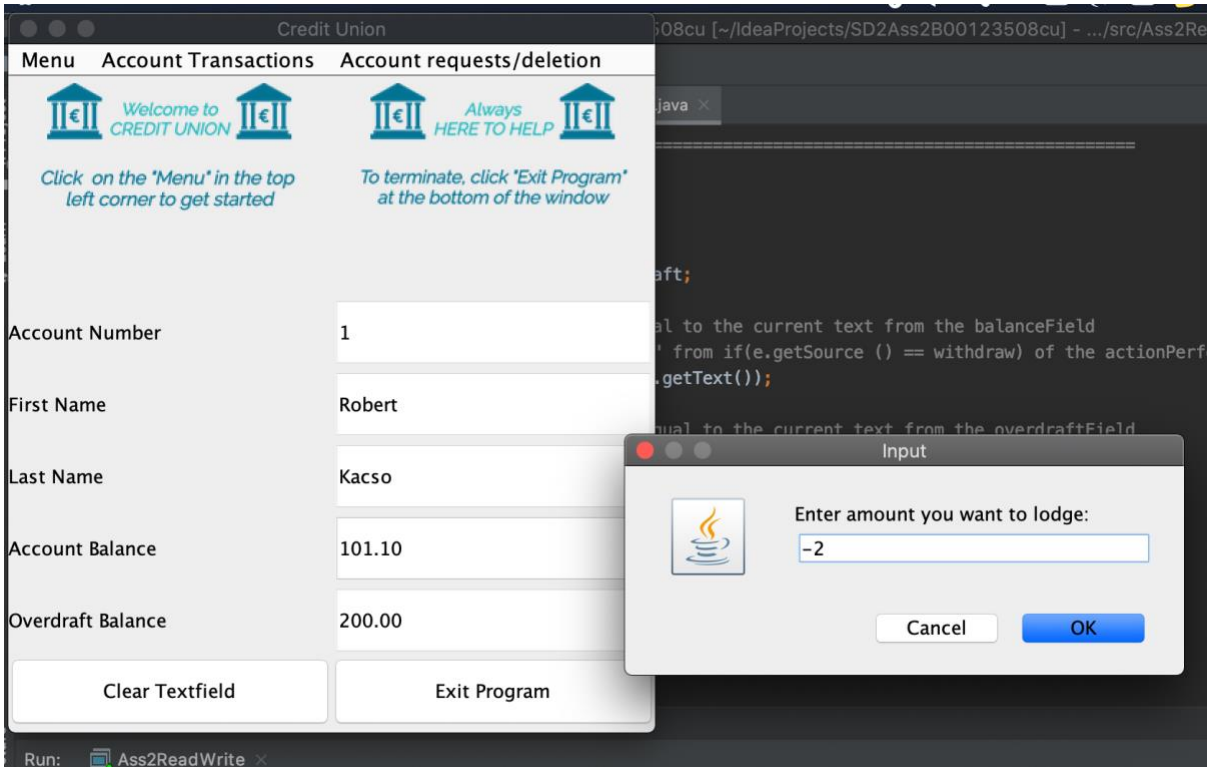
```
//==========================================================================================
// method to request change for overdraft
public void reqOverdraft()
{
    double newOverdraft;

    // Needed here EVEN if it's result is ignored by the program.
    // If the overdraft field is empty, the "catch" from if(e.getSource () == overdraftReq) of the actionPerformed method will come in effect.
    // Without Double.parseDouble(overdraftField.getText()); from bellow, the method would ask to input a new overdraft even if the account field is empty
    Double.parseDouble(overdraftField.getText());

    // prompting user to input amount of new overdraft
    newOverdraft = paneInput( tmpInput: "Enter New Overdraft Amount: ");

    //if ( newOverdraft >=0 )
    //{
        // set overdraftField to show the new overdraft
        overdraftField.setText(String.valueOf(df2.format(newOverdraft)));
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Overdraft amount updated");
    //}


    updatedAccount(); // method called in to update the account with the new input
} // end reqOverdraft method
```

Here is the withdraw() method fixed with the “if” statement

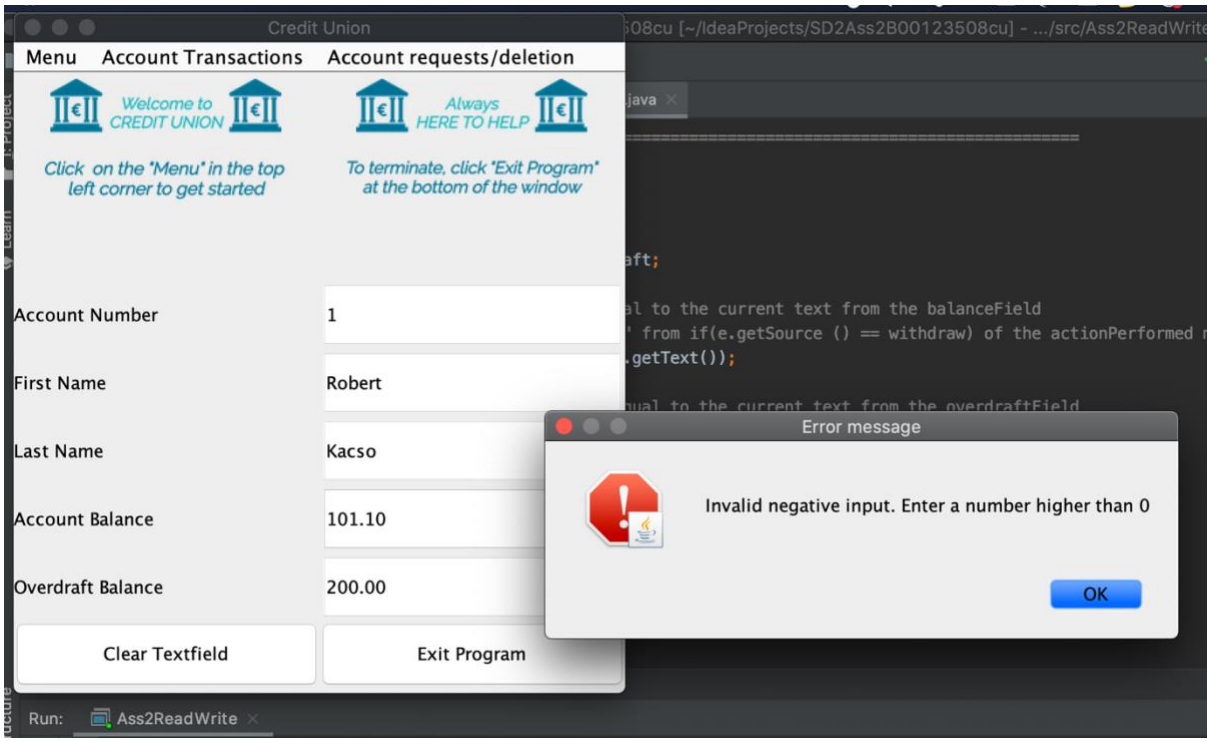
```
//==========================================================================================
// method for withdrawal()
public void withdrawal()
{
    int accept;
    double withdrawAmount, oldBalance, oldOverdraft;

    // variable oldBalance initialized to be equal to the current text from the balanceField
    // If the balance field is empty, the "catch" from if(e.getSource () == withdraw) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // variable oldOverdraft initialized to be equal to the current text from the overdraftField
    oldOverdraft = Double.parseDouble(overdraftField.getText());

    // prompting user to input amount to withdraw
    withdrawAmount = paneInput( tmpInput: "Enter amount you want to withdraw: ");

    // if/else statements for various scenarios...
    if (withdrawAmount < 0) // ... for inputting negative number
    {
        return;
    }
    else
    {
        if ((oldBalance - withdrawAmount) >= 0) // ... for withdrawal of an amount less then or equal to the current balance
        {
            balanceField.setText(String.valueOf(oldBalance - withdrawAmount));
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Balance updated");
```

The problem:

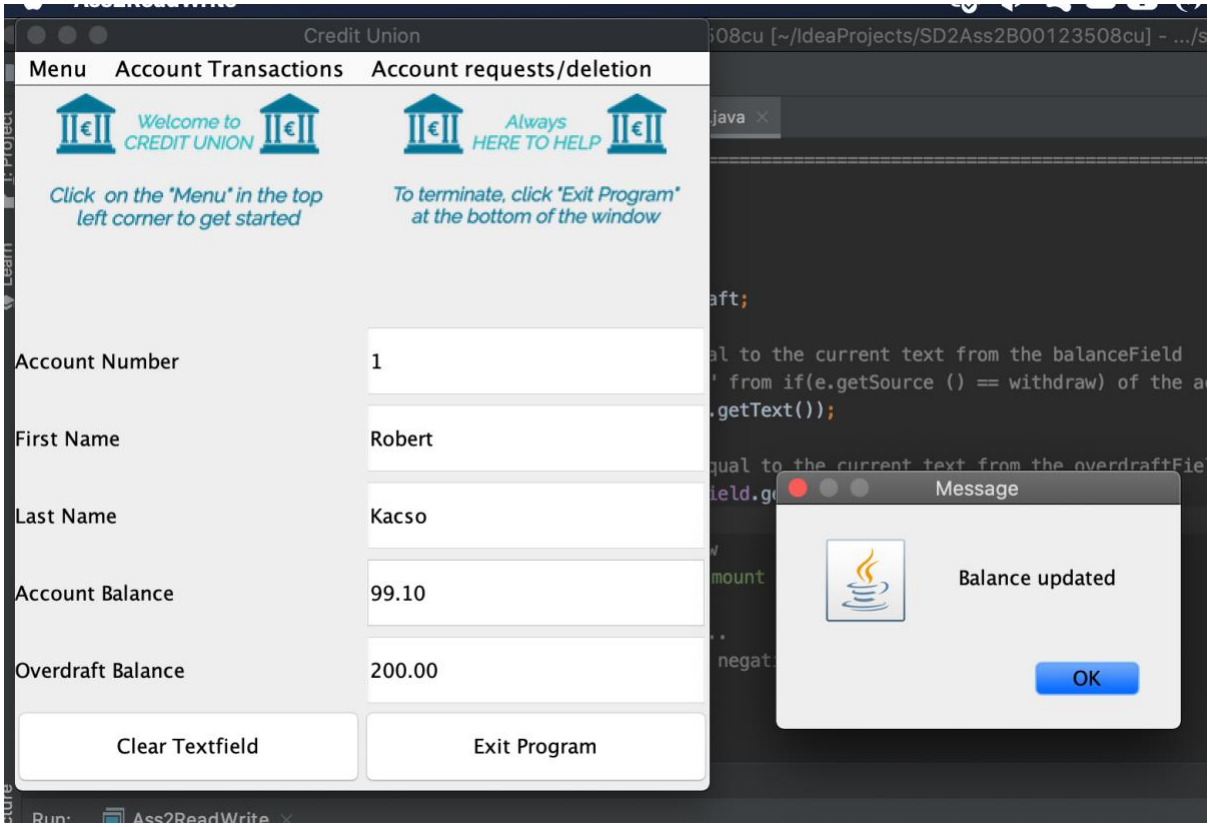
- negative 2 being lodged



- the paneInput() method warns of negative input



- the balance still gets updated



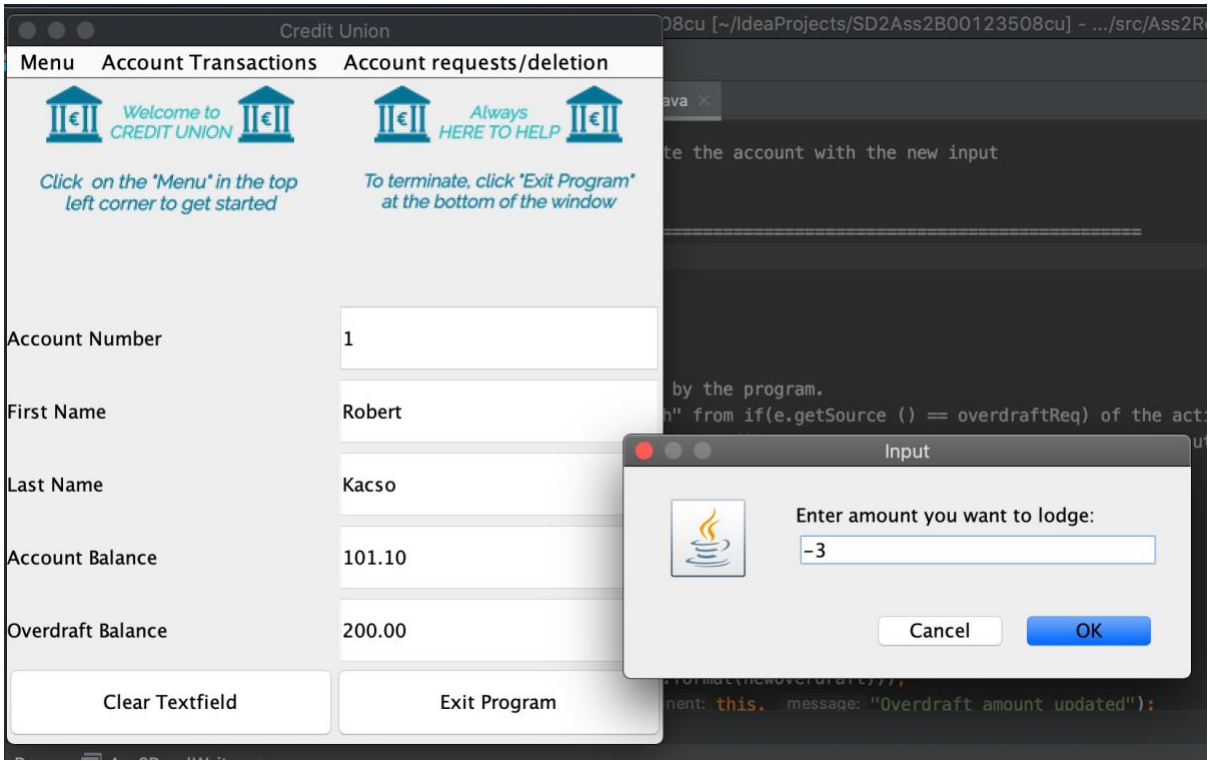
As you can see, even if I executed a lodgement, because it was a negative number, it gets withdrawn from the balance.

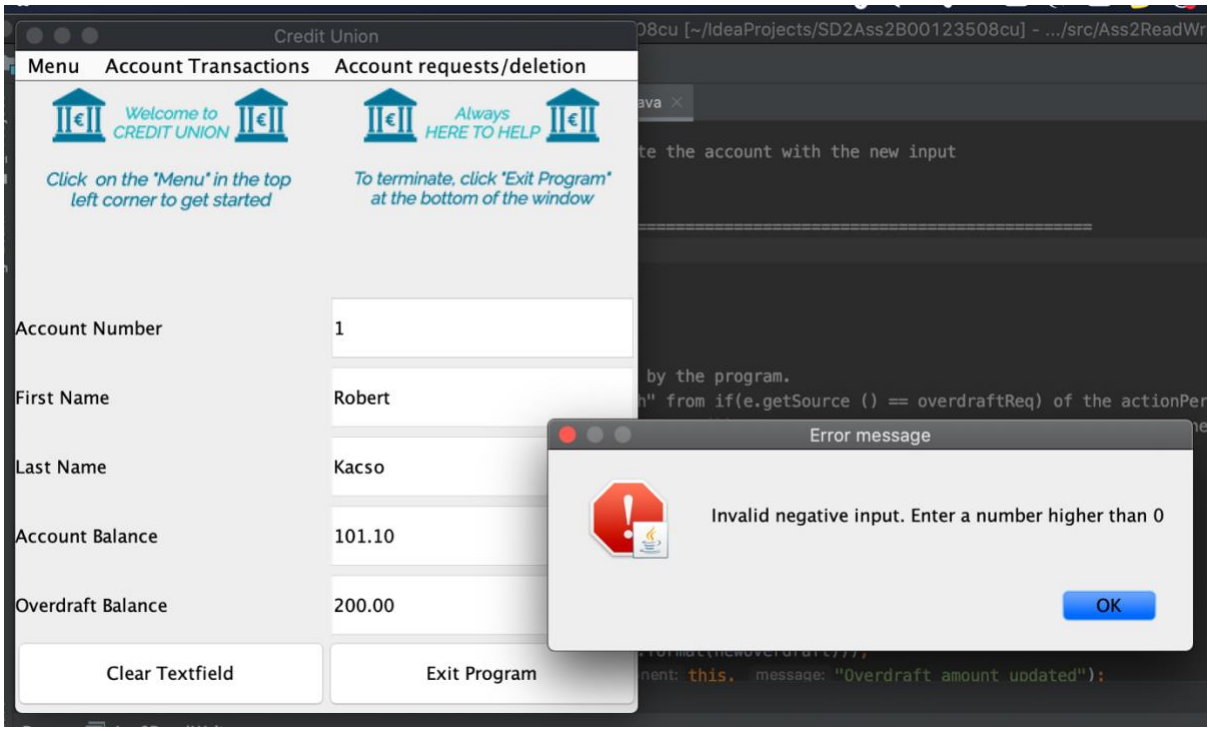
It was solved by introducing an “if” statement wrapped around the output of the respective methods. The witdraw() method gets an extra “if” statement that would prohibit the update of the balance if a number under 0 is being inputted.

Let’s see the outcome. First the code with the “if” statements, and then the running program:

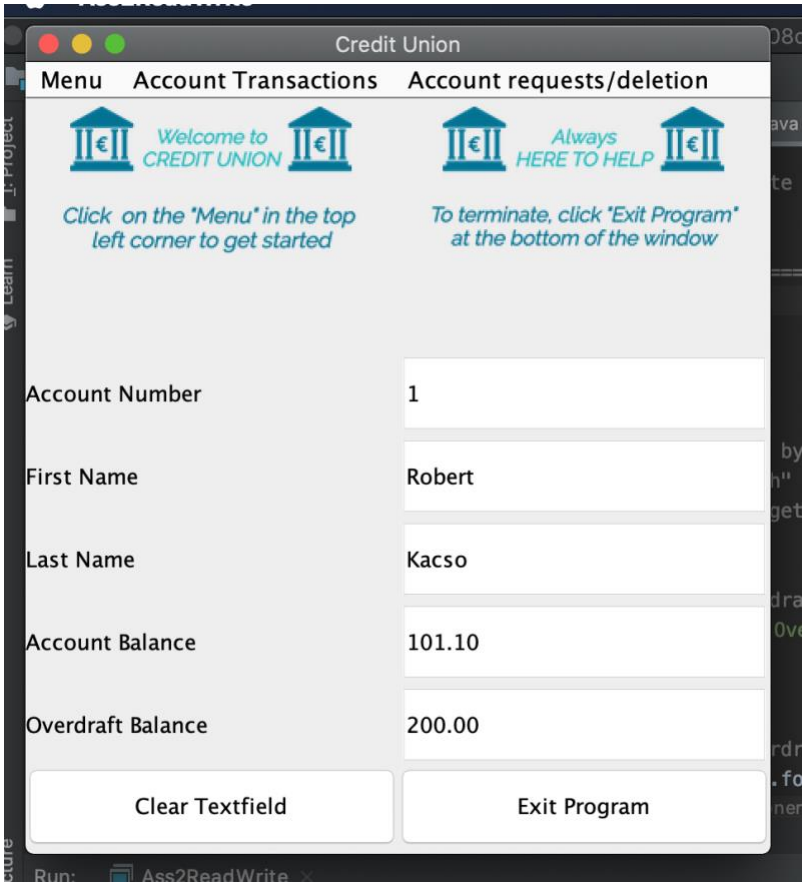
```
//==========================================================================================
// method to lodge money
public void lodgement()
{
    double lodgeAmount, oldBalance, newBalance;

    // variable oldBalance initialized to be equal to the current text from the balanceField
    // If the lodgement field is empty, the "catch" from if(e.getSource () == lodge) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // prompting user to input new lodgement
    lodgeAmount = paneInput( tmpInput: "Enter amount you want to lodge: ");

    // "if" statement to don't let the balance update with a negative number
    if (lodgeAmount > 0)
    {
        // arithmetic calculation of the new balance
        newBalance = lodgeAmount + oldBalance;
        // set balanceField to show the new balance
        balanceField.setText(String.valueOf(df2.format(newBalance)));
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Balance updated");
    }

    updatedAccount(); // method called in to update the account with the new input
} // end lodgement method
```

```
//==========================================================================================
// method to request change for overdraft
public void reqOverdraft()
{
    double newOverdraft;

    // Needed here EVEN if it's result is ignored by the program.
    // If the overdraft field is empty, the "catch" from if(e.getSource () == overdraftReq) of the actionPerformed method will com
    // Without Double.parseDouble(overdraftField.getText()); from bellow, the method would ask to input a new overdraft even if th
    Double.parseDouble(overdraftField.getText());

    // prompting user to input amount of new overdraft
    newOverdraft = paneInput( tmpInput: "Enter New Overdraft Amount: ");

    if ( newOverdraft >=0 )
    {
        // set overdraftfield to show the new overdraft
        overdraftField.setText(String.valueOf(df2.format(newOverdraft)));
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Overdraft amount updated");
    }


    updatedAccount(); // method called in to update the account with the new input
} // end reqOverdraft method
```

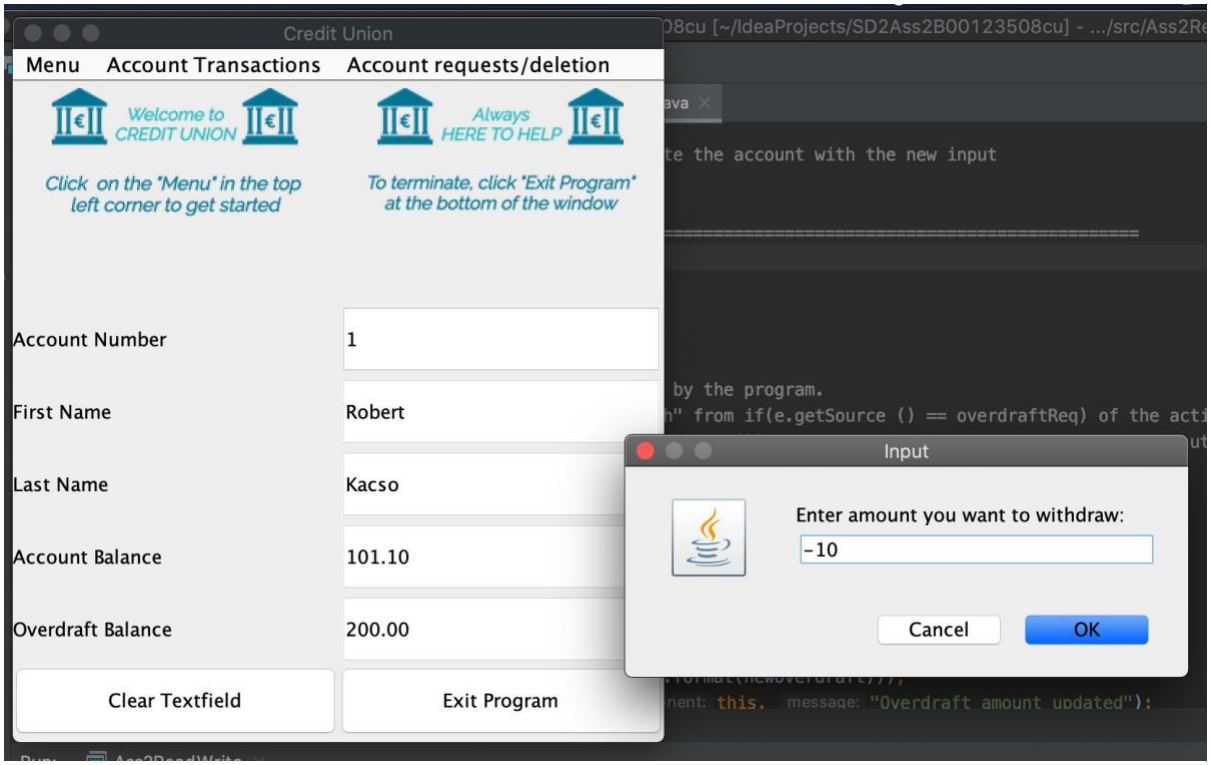


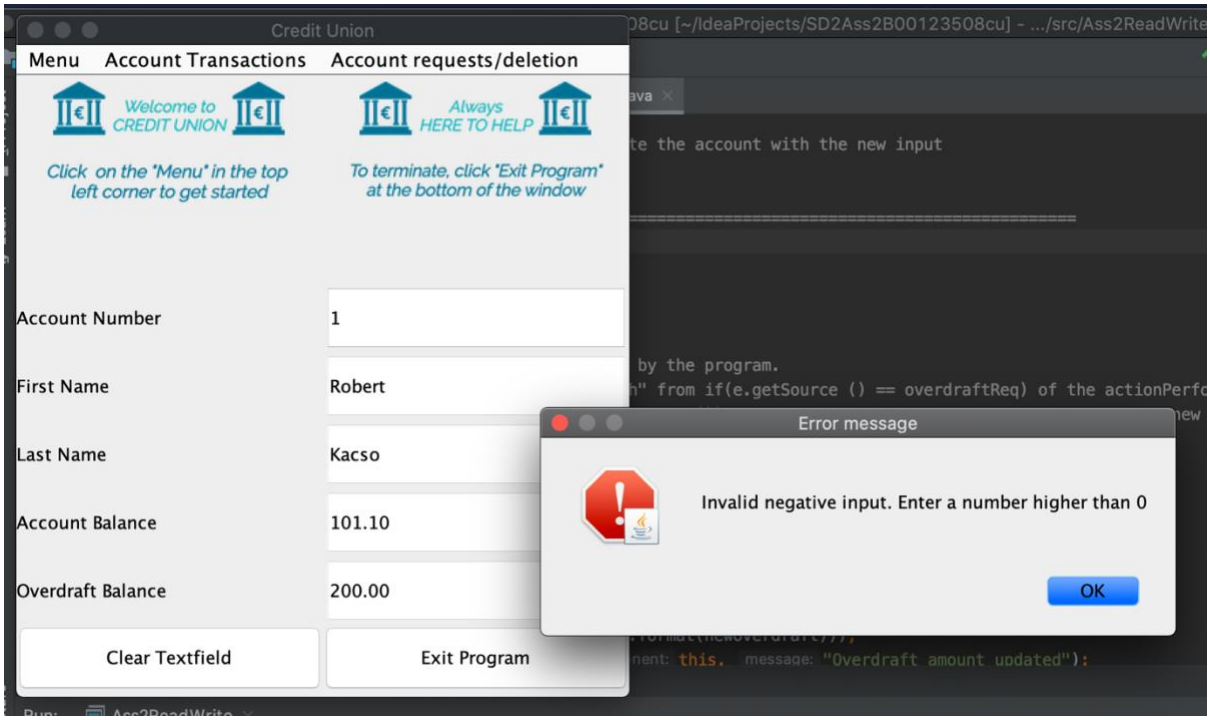


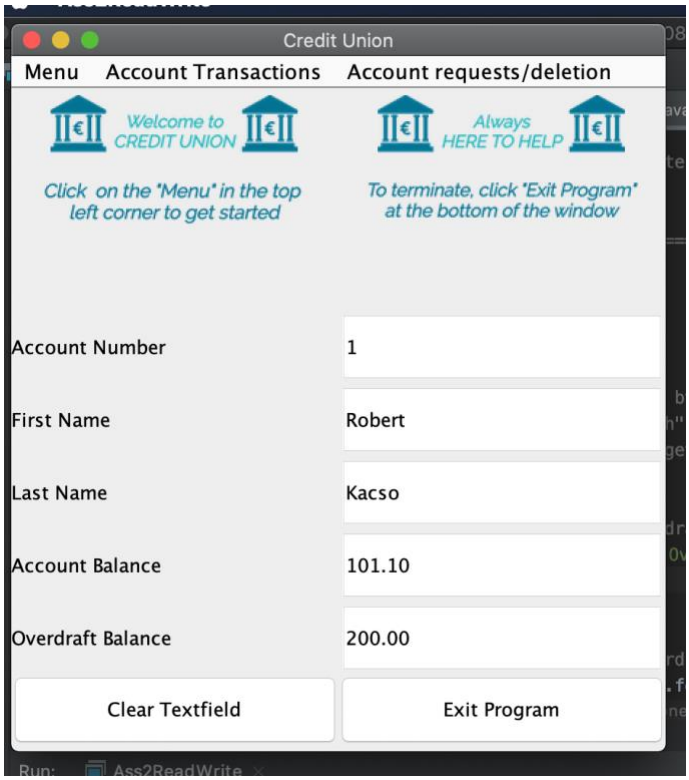
And when I press OK, the balance will not change. Problem solved.



Same for the reqOverdraft() method, but let's look at the more complicated withdraw() method. Will we achieve the same result?







It works as intended.

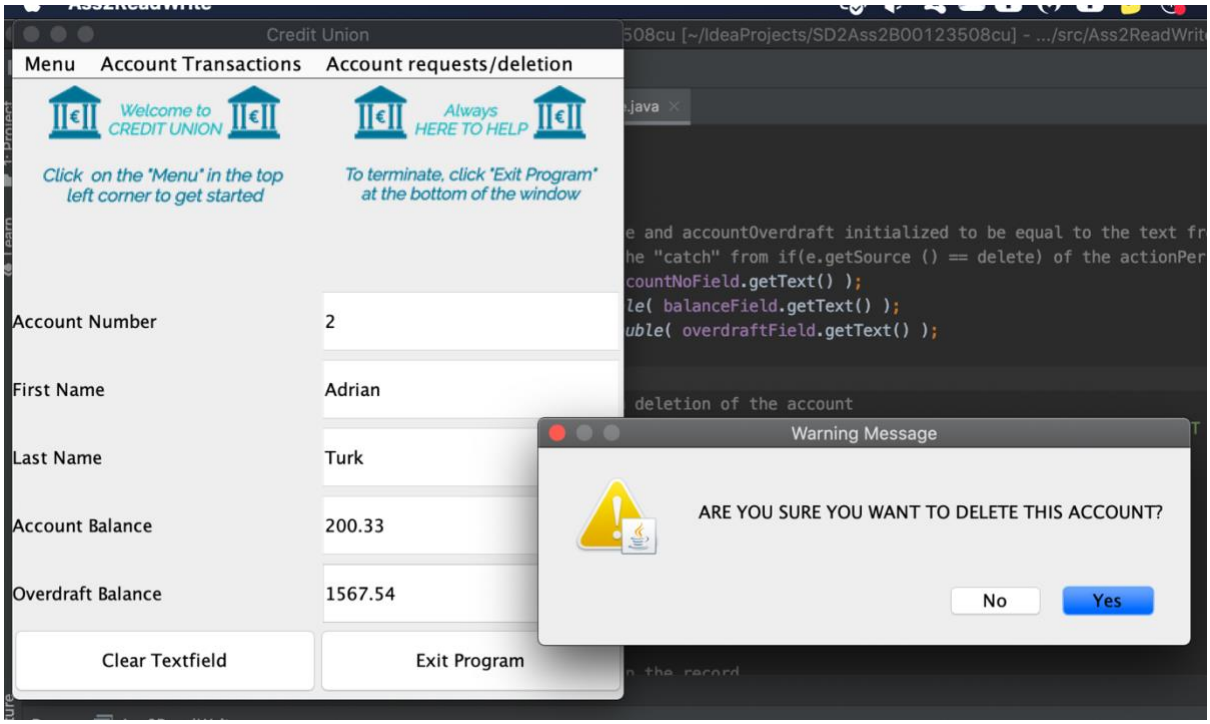
5th problem – delete method

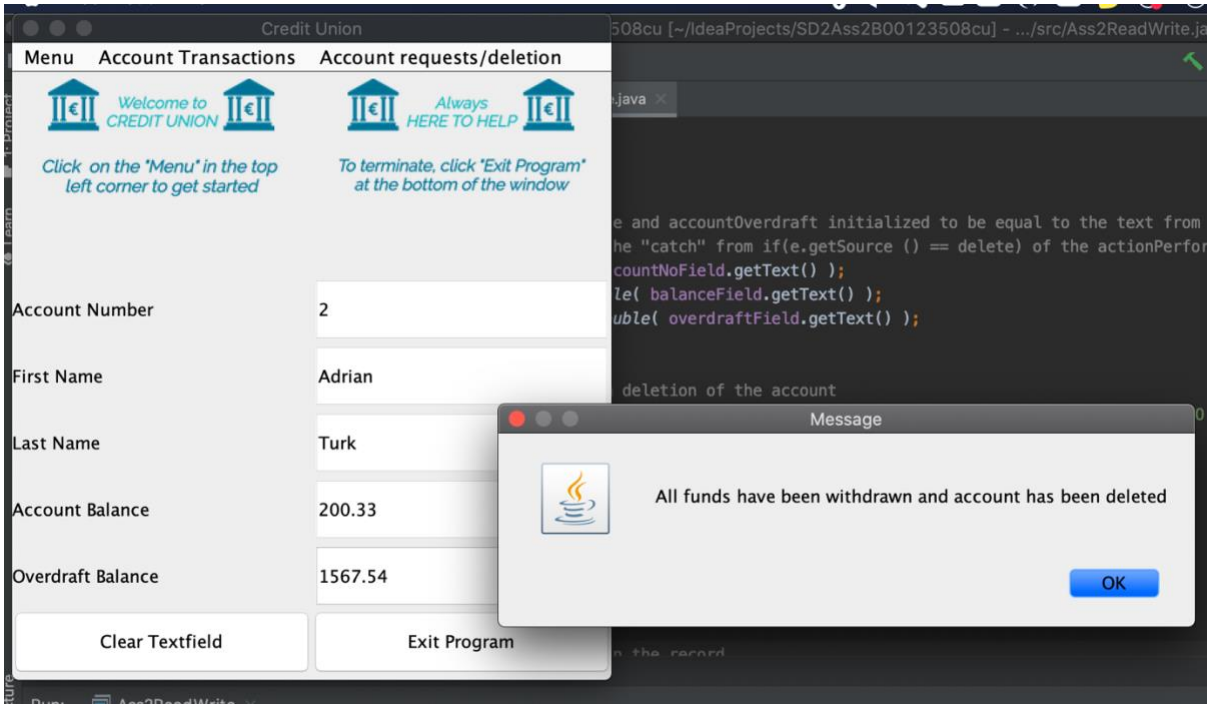
I was able to delete any account, even if the balance and overdraft were not 0. It was a quick and easy fix, but I omitted it when creating the delete() method.

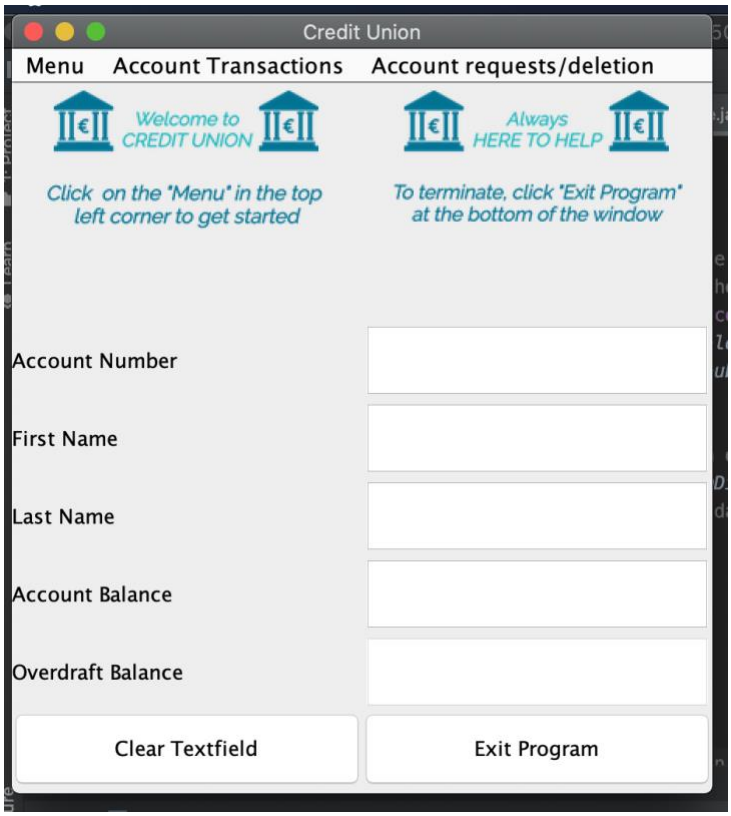
Here is the method with the commented out (and minimized) “if” statements that fixed the problem:

```java
public void delete()
{
    try
    {
        // variable accountNumber, accountBalance and accountOverdraft initialized to be equal to the text from their respective fields
        // If the any of the fields are empty, the "catch" from if(e.getSource () == delete) of the actionPerformed method will come in effect.
        int accountNumber = Integer.parseInt( accountNoField.getText() );
        double accountBalance = Double.parseDouble( balanceField.getText() );
        double accountOverdraft = Double.parseDouble( overdraftField.getText() );

        /*...*/
            // message prompting user to confirm deletion of the account
            int accept = JOptionPane.showConfirmDialog( parentComponent: this,  message: "ARE YOU SURE YOU WANT TO DELETE THIS ACCOUNT?",  title: "Warning Message",
            // if option is accepted, reset all data in the file to null
            if (accept == 0)
            {
                data.setAccountNo( 0 );
                data.setFirstName( null );
                data.setLastName( null);
                data.setBalance( 0) ;
                data.setOverdraft( 0 );

                // seek for the requested file in the record
                randAccFile.seek(  pos: (long) ( accountNumber-1 ) * Ass2Record.size() );
                // write new data to the seeked file
                data.write( randAccFile );

                JOptionPane.showMessageDialog( parentComponent: this,  message: "All funds have been withdrawn and account has been deleted");

                resetTextFields(); // method called in to reset/clear text fields
            }
        //}
    }//end try

    catch (IOException io)
    {
        System.err.println("error during write to file\n" + io.toString() );
    } // end catch
```

And this is the problem that was causing:







As you see, even though the balance field and the overdraft field were not zero, the account has still been deleted.

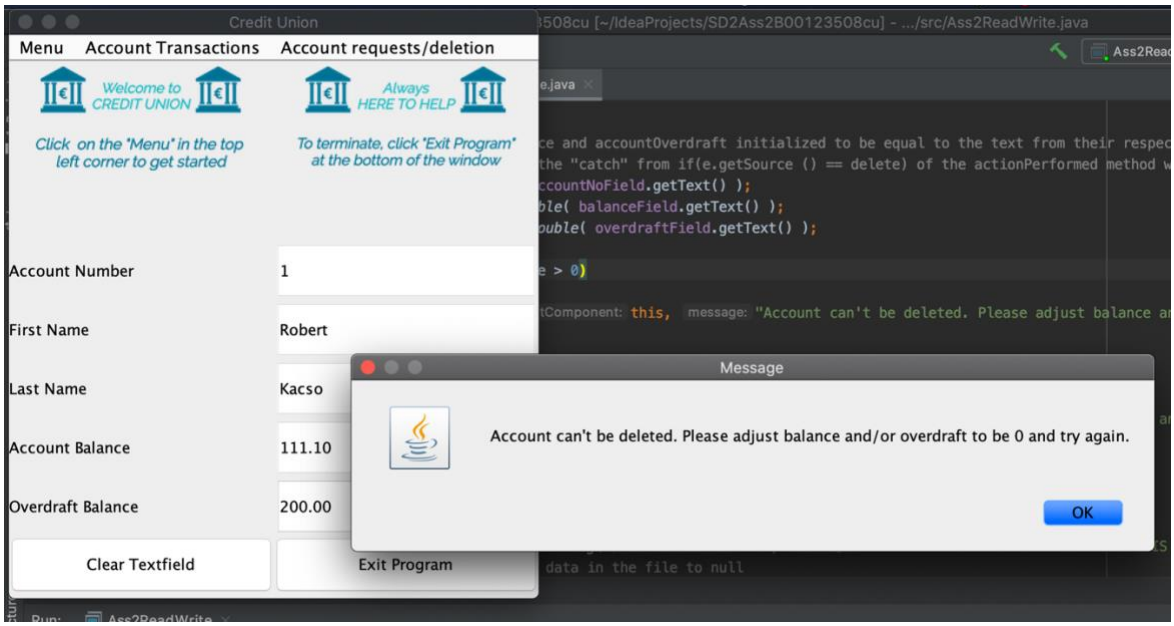
Fixed by introducing an if statement that would require that the balance field won't be negative nor positive and that the overdraft field would be zero too, in order to delete an account.

Let's have a look at the "if" statement in the code first:

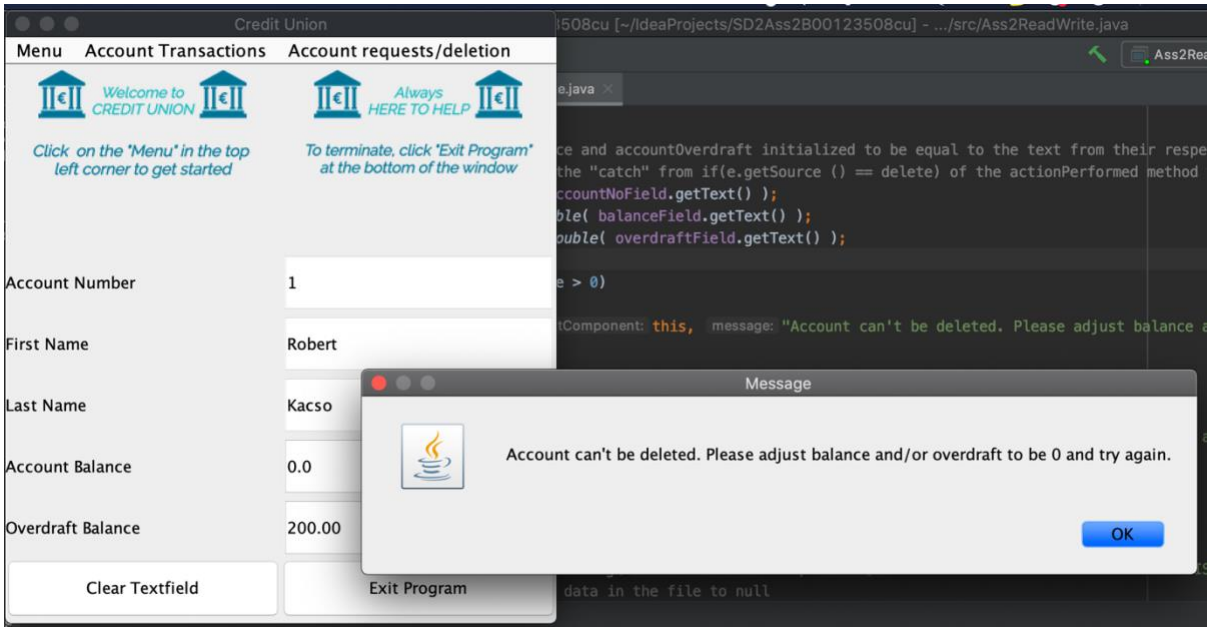
```
//==========================================================================================
// delete method to delete account info and set balances to 0
public void delete()
{
    try
    {
        // variable accountNumber, accountBalance and accountOverdraft initialized to be equal to the text from their respective fields
        // If the any of the fields are empty, the "catch" from if(e.getSource () == delete) of the actionPerformed method will come in effect.
        int accountNumber = Integer.parseInt( accountNoField.getText() );
        double accountBalance = Double.parseDouble( balanceField.getText() );
        double accountOverdraft = Double.parseDouble( overdraftField.getText() );

        if (accountBalance < 0 && accountBalance > 0)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Account can't be deleted. Please adjust balance and/or overdraft to be 0 a
        }

        else if (accountOverdraft > 0)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Account can't be deleted. Please adjust balance and/or overdraft to be 0 a
        }

        else
        {
            // message prompting user to confirm deletion of the account
            int accept = JOptionPane.showConfirmDialog( parentComponent: this,  message: "ARE YOU SURE YOU WANT TO DELETE THIS ACCOUNT?",  title: "Warning
            // if option is accepted, reset all data in the file to null
            if (accept == 0)
            {
                data.setAccountNo( 0 );
                data.setFirstName( null );
                data.setLastName( null );
```

All the rest of the existing previous code has been put in the "else" statement.

Now let's see if the problem was solved:

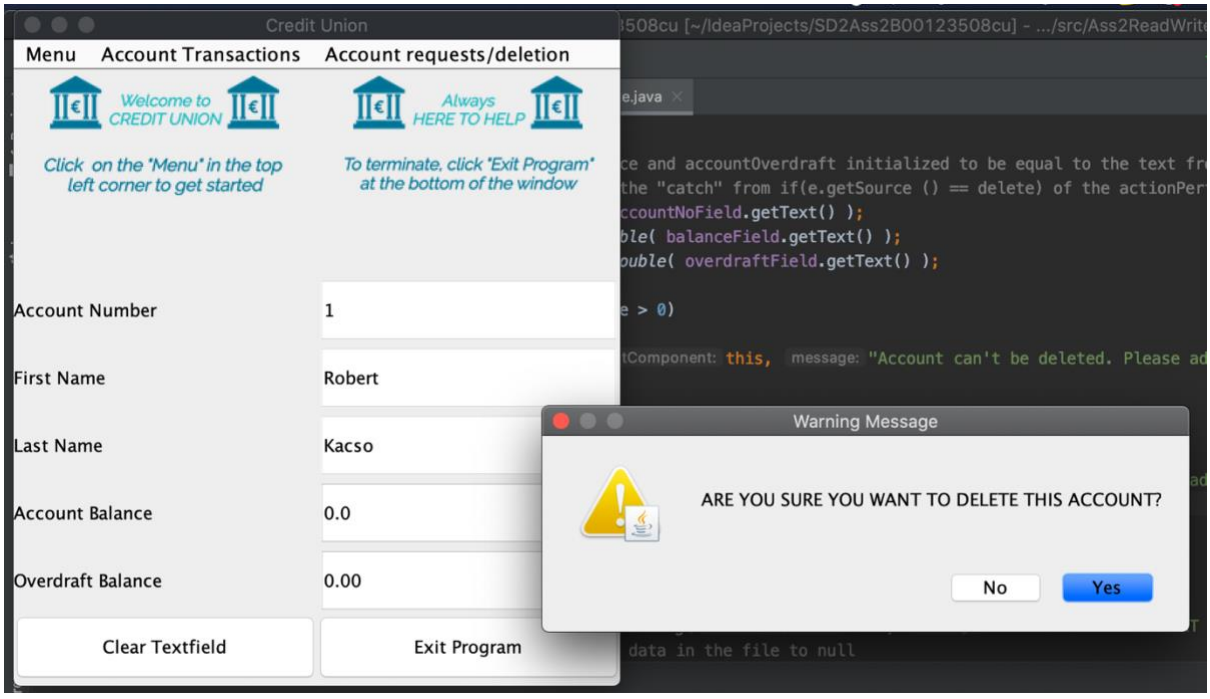


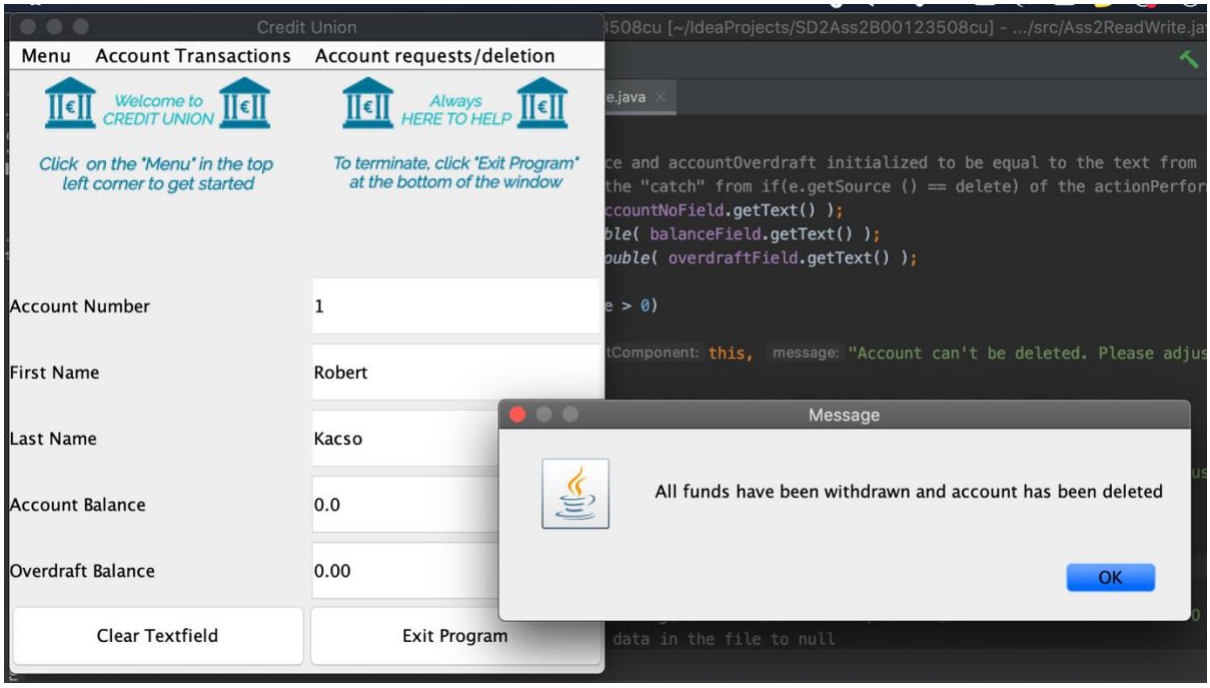
And if I set only the balance to zero:

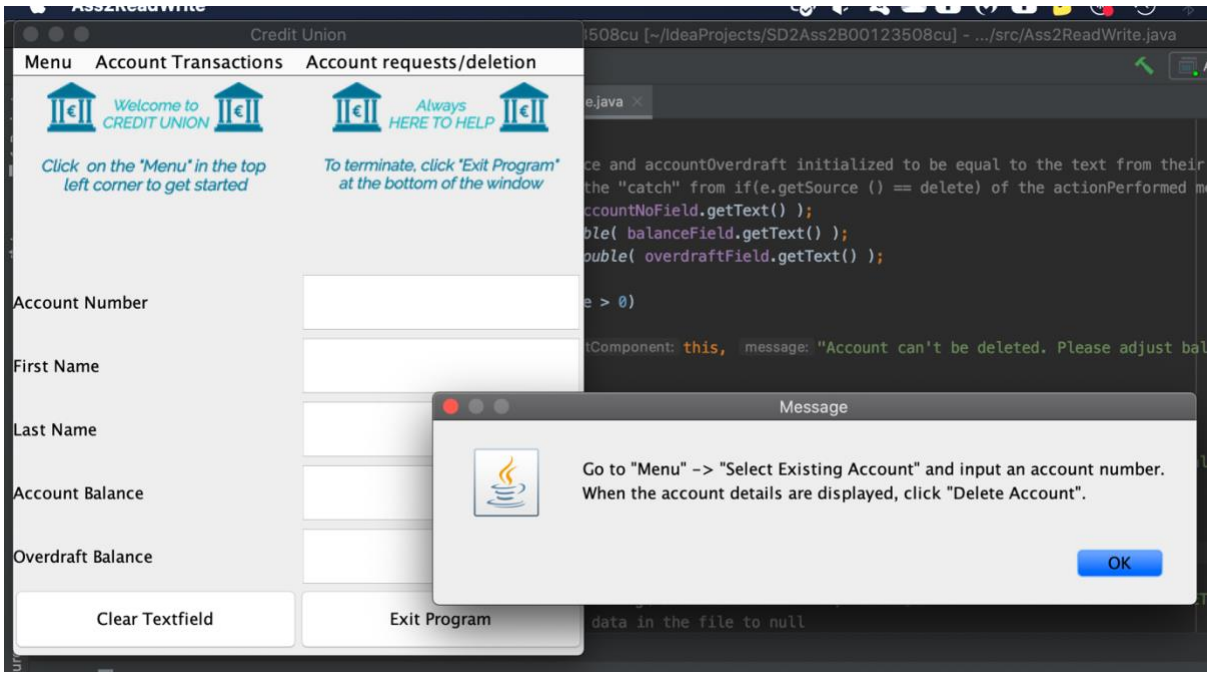


With only the balance field being zero, user is still not allowed to delete the account.

What if the overdraft field is zero too?







After the balance and overdraft were adjusted to be zero, we got a prompt asking if we really want to delete the account, after that another prompt saying that the funds have been withdrawn and account deleted, and a third prompt telling us that if we want to delete another account, go to “Menu” -> “Select Existing Account” and continue from there.

There were my problems and I’m happy were sorted.

Methods

Ass2Record class Methods

public void read(RandomAccessFile file) method

This method reads file from the record.

```java
//==========================================================================================
// method to read a record with a input/output exception
public void read( RandomAccessFile file ) throws IOException
{
    accountNo = file.readInt(); // initialize variable accountNo and read it in

    //-----------------------------------------
    char[] first = new char[15]; // array for the first name of max 15 characters

    for ( int i=0; i < first.length; i++ ) // "for" loop to read in all characters
        first[ i ] = file.readChar();

    firstName = new String (first); // initialize variable firstName and read it in

    //-----------------------------------------
    char[] last = new char[15]; // array for the last name of max 15 characters

    for (int i =0; i<last.length; i++) // "for" loop to read in all characters
        last[i] = file.readChar();

    lastName = new String (last); // initialize variable lastName and read it in

    //-----------------------------------------
    balance = file.readDouble(); // initialize variable balance and read it in

    //-----------------------------------------
    overdraft = file.readDouble(); // initialize variable overdraft and read it in
}
```

public void write(RandomAccessFile file) method

This method writes to each file on the record.

```java
//==========================================================================================
// method to write to a record with a input/output exception
public void write( RandomAccessFile file) throws IOException
{
    StringBuffer buf; // declaring StringBuffer "buf" to append, reverse, replace, concatenate and manipulate Strings or sequence of characters

    file.writeInt( accountNo ); // write account number inputted to file in the record

    //-----------------------------------------
    // if/else statement to help append, reverse, replace, concatenate and manipulate variable firstName type Strings of max 15 characters
    if (firstName != null)
        buf = new StringBuffer( firstName);
    else
        buf = new StringBuffer( capacity: 15 );

    buf.setLength( 15 ); // buffer length set to 15 characters

    file.writeChars( buf.toString() ); // write firstName inputted to file in the record

    //-----------------------------------------
    // if/else statement to help append, reverse, replace, concatenate and manipulate variable lastName type Strings of max 15 characters
    if ( lastName != null )
        buf = new StringBuffer( lastName );
    else
        buf = new StringBuffer( capacity: 15);

    buf.setLength( 15 ); // buffer length set to 15 characters

    file.writeChars( buf.toString() ); // write lastName inputted to file in the record

    //-----------------------------------------
    file.writeDouble( balance ); // write balance inputted to file in the record

    //-----------------------------------------
    file.writeDouble( overdraft ); // write overdraft inputted to file in the record
}
```

Methods to set and get details from the text field

These methods will set the data and get the data from the text filed, for 1 file from the record. These methods are followed by a size() method, that will determine the size of each file in the record.

```
//==================================================================
// methods to set and get details from the record
public void setAccountNo(int a ) { accountNo = a; }
public int getAccountNo() { return accountNo;}

public void setFirstName( String f) {firstName = f;}
public String getFirstName() { return firstName; }

public void setLastName ( String l) { lastName = l; }
public String getLastName() { return lastName; }

public void setBalance( double b) { balance = b; }
public double getBalance() { return balance; }

public void setOverdraft( double o) { overdraft = o; }
public double getOverdraft() { return overdraft; }
```

public static in size() method

```
//-----------------------------------------
//determines size (bytes) of each file
public static int size() { return 80;}
```

Ass2CreateRandomFile class Methods

public Ass2CreateRandomFile () constructor

Used to create, in our case, 100 blank files in the record.

```
// constructor
public Ass2CreateRandomFile()
{
    Ass2Record blankRec = new Ass2Record(); //instance of variable "blankRec"

    try
    {
        RandomAccessFile randAccFile = new RandomAccessFile( name: "creditUnion.dat",  mode: "rw"); // instance of the randAccFile variable
        randAccFile.seek( pos: 0); // start seeking for empty file in the record at position 0

        for (int i=0; i<100; i++) // "for" loop to create 100 empty files onto the record
            blankRec.write(randAccFile);
    } // end try

    catch(IOException e)
    {
        System.err.println("File not opened properly\n" + e.toString() );
        System.exit(  status: 1 );
    } // end catch
}
```

Ass2ReadWrite class Methods

Ass2ReadWrite() constructor

First part will declare the instances for the Ass2Record and RandomAccessFile

```
//==========================================================================================
// constructor - initialises the Frame
public Ass2ReadWrite()
{
    super( title: "Credit Union"); // title of the frame

    // open file
    data = new Ass2Record(); //instance of variable "data"

    // access record
    try
    {
        randAccFile = new RandomAccessFile( name: "creditUnion.dat",  mode: "rw"); // instance of variable randAccFile
        System.out.println("In first try block");
    } // end try

    catch ( IOException e)
    {
        System.err.println( e.toString() );
        System.exit( status: 1);
    } // end catch
```

Then it creates the GUI:

- frame and menu bar

```
//==========================================================================================
// dropdown menu learned it from https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html
JFrame f = new JFrame( title: "Credit Union"); // declare and initialize JFrame "f"
f.getContentPane().setLayout(new GridLayout( rows: 9, cols: 2)); // set layout of the frame

JMenuBar mb = new JMenuBar(); // declare and initialize JMenubar
f.setJMenuBar(mb); // set the menu bar to the frame
```

- Menu and its 2 menu items, create and select. “Create” for creating new accounts and “Select” for selecting existing accounts.

```
//----------------------------------------
// menu and its submenus (new account and existing account)
JMenu menu = new JMenu( s: "Menu"); // declare and initialize JMenu
mb.add(menu); // add menu to the menu bar

create=new JMenuItem( text: "Create New Account"); // declare and initialize JMenuItem
create.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, InputEvent.ALT_DOWN_MASK)); // key short cut alt+N
create.addActionListener( l: this); // add action listener to "create"
menu.add(create); // add create to menu

select=new JMenuItem( text: "Select an Existing Account"); // declare and initialize JMenuItem
select.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E, InputEvent.ALT_DOWN_MASK)); // key short cut alt+E
select.addActionListener( l: this); // add action listener to "view"
menu.add(select); // add view to menu
```

- Transaction menu and its 2 menu items, lodge and withdraw. “Lodge” to make a lodgement and “Withdraw” to withdraw an amount from the account.

```
//----------------------------------------
// lodge/withdraw and its submenus (lodge and withdraw)
JMenu transactions = new JMenu( s: "Account Transactions"); // declare and initialize JMenu
mb.add(transactions); // add transactions to the menu bar

lodge=new JMenuItem( text: "Lodgement"); // declare and initialize JMenuItem
lodge.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L, InputEvent.ALT_DOWN_MASK)); // key short cut alt+L
lodge.addActionListener( l: this); // add action listener to "lodge"
transactions.add(lodge); // add lodge to transactions

withdraw=new JMenuItem( text: "Withdrawal"); // declare and initialize JMenuItem
withdraw.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W, InputEvent.ALT_DOWN_MASK)); // key short cut alt+W
withdraw.addActionListener( l: this); // add action listener to "withdraw"
transactions.add(withdraw); // add withdraw to transactions
```

- Account Requests/Delete and its 2 menu items, overdraft request and delete account. “Overdraft request” to adjust the amount of overdraft and “Delete” to delete an account.

```
//----------------------------------------
// overdraftReq/delete and its submenus (overdraft request and delete account)
JMenu reqDel = new JMenu( s: "Account requests/deletion"); // declare and initialize JMenu
mb.add(reqDel); // add reqDel to the menu bar

overdraftReq=new JMenuItem( text: "Overdraft Adjustment"); // declare and initialize JMenuItem
overdraftReq.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, InputEvent.ALT_DOWN_MASK)); // key short cut alt+O
overdraftReq.addActionListener( l: this); // add action listener to "overdraftReq"
reqDel.add(overdraftReq); // add overdraftReq to reqDel

delete=new JMenuItem( text: "Delete Account"); // declare and initialize JMenuItem
delete.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_BACK_SPACE, InputEvent.ALT_DOWN_MASK)); // key short cut alt+BACK_SPACE
delete.addActionListener( l: this); // add action listener to "delete"
reqDel.add(delete); // add delete to reqDel
```

- Labels that contain images at the top of the pane.

```
//----------------------------------------
// labels that include image icons added to the frame
// all images created by me freely online at https://logomakr.com/
f.getContentPane().add( new JLabel(new ImageIcon( filename: "creditUnionWelcome.png") ));
f.getContentPane().add(new JLabel(new ImageIcon( filename: "creditUnionHereToHelp.png") ));
f.getContentPane().add( new JLabel(new ImageIcon( filename: "creditUnionMenu.png") ));
f.getContentPane().add(new JLabel(new ImageIcon( filename: "creditUnionExit.png") ));
```

- Labels and text fields for the account details, starting with 2 empty labels used for spacing/design.

```
//----------------------------------------
// text fields and its corresponding labels added to the JFrame "f"
f.getContentPane().add( new JLabel()); // empty JLabel for spacing
f.getContentPane().add( new JLabel()); // empty JLabel for spacing

f.getContentPane().add( new JLabel( text: "Account Number") );
accountNoField = new JTextField();
accountNoField.setEditable(false);
f.getContentPane().add( accountNoField );

f.getContentPane().add( new JLabel( text: "First Name") );
firstNameField = new JTextField();
firstNameField.setEditable(false);
f.getContentPane().add( firstNameField );

f.getContentPane().add( new JLabel( text: "Last Name") );
lastNameField = new JTextField();
lastNameField.setEditable(false);
f.getContentPane().add( lastNameField );

f.getContentPane().add( new JLabel( text: "Account Balance") );
balanceField = new JTextField();
balanceField.setEditable(false);
f.getContentPane().add( balanceField );

f.getContentPane().add( new JLabel( text: "Overdraft Balance") );
overdraftField = new JTextField();
overdraftField.setEditable(false);
f.getContentPane().add( overdraftField );
```

- Buttons at the bottom of the pane, one for clearing the text fields and one for terminating the program.

```
//----------------------------------------
// buttons at the bottom of the frame to clear the text fields and to close/terminate the program
clear = new JButton ( text: "Clear Textfield");
clear.addActionListener( l: this);
f.getContentPane().add (clear);

done = new JButton( text: "Exit Program");
done.addActionListener( l: this);
f.getContentPane().add (done);
```

- At the end of the constructor we determine the frame size, set it visible and a closing method, to be able to “x” out our frame window and terminate the program in the same time.

```
    //-----------------------------------------
    f.setSize( width: 450, height: 500); // frame size
    f.setVisible(true); // frame set to be visible
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // close the program by "X"-ing out the frame window
} // end constructor
```

public double paneInput (String tmpInput) method

Created this method to take the input from a pop up (JOptionPane) window, and return it.

```
//==========================================================================
// take input from JOptionPane
public double paneInput (String tmpInput)
{
    double input = Double.parseDouble(JOptionPane.showInputDialog( parentComponent: this, tmpInput,  title: "Input", JOptionPane.QUESTION_MESSAGE));

    try
    {
        // "if" statement to give an error message if a negative number is inputted
        if (input < 0)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Invalid negative input. Enter a number higher than 0",  title: "Error message", JOption
        }
    } // end try

    catch (Exception e)
    {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Invalid negative input. Enter a number higher than 0",  title: "Error message", JOptionPane
    } // end catch

    return input;
} // end paneInput method
```

public void accountDetails(DecimalFormat df2) method

Used in another method to display the text of a file from the record, to the text fields.

```
//==========================================================================================
// method to read in the account details from the text fields and make the text fields uneditable
public void accountDetails(DecimalFormat df2)
{
    accountNoField.setText(String.valueOf(data.getAccountNo()));
    accountNoField.setEditable(false);

    firstNameField.setText(data.getFirstName());
    firstNameField.setEditable(false);

    lastNameField.setText(data.getLastName());
    lastNameField.setEditable(false);

    balanceField.setText(String.valueOf(df2.format(data.getBalance())));
    balanceField.setEditable(false);

    overdraftField.setText(String.valueOf(df2.format(data.getOverdraft())));
    overdraftField.setEditable(false);
} // end accountDetails method
```

public void updateAccount() method

Used at the end of add record, lodge, withdraw, overdraft request and delete methods, to update the record file with the new modified details.

```java
public void updatedAccount()
{
    // declaring and initializing temporary variables which then could be seeked and modified
    int tmpAccount = Integer.parseInt( accountNoField.getText() );
    double tmpBalance = Double.parseDouble( balanceField.getText() );
    double tmpOverdraft = Double.parseDouble( overdraftField.getText() );

    try
    {
        // call in method randomAccessFile from the record class and initialize it to be equal to variable randAccFile
        randAccFile = data.randomAccessFile();
        //data.read( randAccFile ); // can be used too, just wanted to practice extra methods

        // writing out the new inputted data to the text fields
        data.setAccountNo( tmpAccount );
        data.setFirstName( firstNameField.getText() );
        data.setLastName( lastNameField.getText() );
        data.setBalance( tmpBalance );
        data.setOverdraft( tmpOverdraft );

        // seek for the requested file in the record
        randAccFile.seek( pos: (long) ( tmpAccount-1 ) * Ass2Record.size() );
        // write data to the seeked file
        data.write( randAccFile );
    } // end try

    catch (IOException e ) // input/output exception catch
    {
        System.err.println("Error writing record\n " + e.toString() );
        System.exit( status: 1 );
    } // end catch

} // end updateAccount method
```

public void addRecord() method

This method adds the details to the blank record files.

```java
public void addRecord()
{
    editTextFields();
    accountNoField.getText();
    {
        try
        {
            int tmpAccount = Integer.parseInt( accountNoField.getText() );

            System.out.println("In second try block");

            if (tmpAccount < 1 || tmpAccount > 100)  //validate account number is in range
            {
                JOptionPane.showMessageDialog( parentComponent: this, message: "Account number must be between 1 & 100", title: "Error message", JOptionPane.ERROR
            }

            else
            {
                //read file to check if account number already exists.
                randAccFile.seek( pos: (long) (tmpAccount - 1) * Ass2Record.size());
                data.read(randAccFile);

                if (data.getAccountNo() == tmpAccount)  //if account exists, display dialog box to user
                {
                    JOptionPane.showMessageDialog( parentComponent: this, message: "Account Number already exists! Please try a different Account Number");
                    accountNoField.setText("");   // clear account textfield
                }
                else   //once conditions are met, data is written to file.
                {
                    updatedAccount(); // method called in to update the account with the new input
                    JOptionPane.showMessageDialog( parentComponent: this, message: "Account created and details saved");
                }
            }

            resetTextFields(); // method called in to reset/clear textfield

        } // end try
```

```
        catch (NumberFormatException nfe ) // number format exception catch
        {
            JOptionPane.showMessageDialog( parentComponent: this, message: "Please input the details of the new account and click \"Create New Account\" again. " +
                    "\nIf necessary, reset the text fields by clicking on \"Clear Textfield\".");
        } // end catch

        catch (IOException io) // input/output exception catch
        {
            System.err.println("error during write to file\n" + io.toString() );
        } // end catch

    }//end initial if statement
} // end addRecord method
```

public void readRecordByInput() method

It prompt user to inputting into paneInput() method an account number, and it retrieves the details from that specific file.

```
// create method to read the record by inputting the account number used in delete, lodge, withdraw and overdraft methods
public void readRecordByInput()
{
    try
    {
        int accountNumber = (int) paneInput( tmpInput: "ENTER ACCOUNT NUMBER");

        if (accountNumber < 1 || accountNumber > 100)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "WRONG INPUT!!! Input a number between 1 and 100.");
        }

        else
            {
                // seek for the requested file in the record
                randAccFile.seek( pos: (accountNumber - 1) * Ass2Record.size());
                // read data from the seeked file
                data.read(randAccFile);

                if (data.getAccountNo() == 0) // "if" statement to reset/clear accountNoFiled if account doesn't exist
                {
                    JOptionPane.showMessageDialog( parentComponent: this,  message: "Account does not exist");
                    resetTextFields();
                }
                else
                {
                    accountDetails(df2); // if account exists output the account details
                }
            }

    }//end try
    catch (EOFException eof )
    {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Account does not exist");
    } // end catch

    catch (IOException e )
    {
        System.err.println("Error during read from file\n " + e.toString() );
        System.exit(  status: 1 );
    } // end catch
} // end readRecordByInput method method
```

public void lodgement() method

To execute a lodgement, this method is being called. It basically reads the balance field for the existing balance and adds to it the new inputted lodgement. The result of the addition is being outputted to the balance field and then the file is being update by the updateDetails()

method. Before all this it verifies that the inputted number is not negative. If it is the balance is not updated and a pop up window will come up saying to input a number above zero.

```
//==========================================================================================
// method to lodge money
public void lodgement()
{
    double lodgeAmount, oldBalance, newBalance;

    // variable oldBalance initialized to be equal to the current text from the balanceField
    // If the lodgement field is empty, the "catch" from if(e.getSource () == lodge) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // prompting user to input new lodgement
    lodgeAmount = paneInput( tmpInput: "Enter amount you want to lodge: ");

    // "if" statement to don't let the balance update with a negative number
    if (lodgeAmount > 0)
    {
        // arithmetic calculation of the new balance
        newBalance = lodgeAmount + oldBalance;
        // set balanceField to show the new balance
        balanceField.setText(String.valueOf(df2.format(newBalance)));
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Balance updated");
    }

    updatedAccount(); // method called in to update the account with the new input
} // end lodgement method
```

public void withdrawal() method

For withdrawals, this method is being called in. First it verifies that the number inputted to be withdrawn is not negative, then it checks if the withdrawn amount is less then or equal to the balance. If it is it updates the balance by deducting the withdrawn amount from the balance. It also checks that the amount withdrawn is more than the balance but not reaching the max overdraft. Prompts a message asking the user if they want to go into overdraft. And it checks if the amount withdrawn passes the limit of the account balance plus the overdraft. If it is, it tells the user that there are not enough funds for the transaction.

```
//==========================================================================================
// method for withdrawal()
public void withdrawal()
{
    int accept;
    double withdrawAmount, oldBalance, oldOverdraft;

    // variable oldBalance initialized to be equal to the current text from the balanceField
    // If the balance field is empty, the "catch" from if(e.getSource () == withdraw) of the actionPerformed method will come in effect.
    oldBalance = Double.parseDouble(balanceField.getText());

    // variable oldOverdraft initialized to be equal to the current text from the overdraftField
    oldOverdraft = Double.parseDouble(overdraftField.getText());

    // prompting user to input amount to withdraw
    withdrawAmount = paneInput( tmpInput: "Enter amount you want to withdraw: ");

    // if/else statements for various scenarios...
    if (withdrawAmount < 0) // ... for inputting negative number
    {
        return;
    }
```

```
    else
    {
        if ((oldBalance - withdrawAmount) >= 0) // ... for withdrawal of an amount less then or equal to the current balance
        {
            balanceField.setText(String.valueOf(oldBalance - withdrawAmount));
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Balance updated");
        }
        else if ((oldBalance - withdrawAmount) >= (0 - oldOverdraft)) // ... for withdrawal of an amount that puts account holder into overdraft
        {
            accept = JOptionPane.showConfirmDialog( parentComponent: this,  message: "You accept going into overdraft?",  title: "Warning Message", JOptionPane.YES_NO_
            if (accept == 0) // if user accepts to go into overdraft, the balance is updated
            {
                balanceField.setText(String.valueOf(oldBalance - withdrawAmount));
                JOptionPane.showMessageDialog( parentComponent: this,  message: "Balance updated");
            }
        }
        else if ((oldBalance - withdrawAmount) < (0 - oldOverdraft)) // ... for withdrawal of an amount greater then the max overdraft allowed
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Sorry, not enough funds. Try a lesser amount.",  title: "Error message", JOptionPane.ERROR
        }
    }

    updatedAccount(); // method called in to update the account with the new input
} // end withdrawal method
```

public void reqOverdraft() method

Very similar to the lodgement() method, only that it does not add up anything. It just replaces the old overdraft amount with the new overdraft amount.

```
//==========================================================================================
// method to request change for overdraft
public void reqOverdraft()
{
    double newOverdraft;

    // Needed here EVEN if it's result is ignored by the program.
    // If the overdraft field is empty, the "catch" from if(e.getSource () == overdraftReq) of the actionPerformed method will come in effect.
    // Without Double.parseDouble(overdraftField.getText()); from bellow, the method would ask to input a new overdraft even if the account field is empty, rer
    Double.parseDouble(overdraftField.getText());

    // prompting user to input amount of new overdraft
    newOverdraft = paneInput( tmpInput: "Enter New Overdraft Amount: ");

    if ( newOverdraft >=0 )
    {
        // set overdraftField to show the new overdraft
        overdraftField.setText(String.valueOf(df2.format(newOverdraft)));
        JOptionPane.showMessageDialog( parentComponent: this,  message: "Overdraft amount updated");
    }


    updatedAccount(); // method called in to update the account with the new input
} // end reqOverdraft method
```

public void delete() method

It deletes the selected account if the balance and the overdraft is zero. If not, it prompt the user to adjust the said fields by withdrawing the amount from the account, or lodging into the account if it is overdrawn, and to request to nullify the overdraft.

```
// delete method to delete account info and set balances to 0
public void delete()
{
    try
    {
        // variable accountNumber, accountBalance and accountOverdraft initialized to be equal to the text from their respective fields
        // If the any of the fields are empty, the "catch" from if(e.getSource () == delete) of the actionPerformed method will come in effect.
        int accountNumber = Integer.parseInt( accountNoField.getText() );
        double accountBalance = Double.parseDouble( balanceField.getText() );
        double accountOverdraft = Double.parseDouble( overdraftField.getText() );

        if (accountBalance < 0 && accountBalance > 0)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Account can't be deleted. Please adjust balance and/or overdraft to be 0 and try aga
        }

        else if (accountOverdraft > 0)
        {
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Account can't be deleted. Please adjust balance and/or overdraft to be 0 and try aga
        }

        else
        {
```

```
        else
        {
            // message prompting user to confirm deletion of the account
            int accept = JOptionPane.showConfirmDialog( parentComponent: this,  message: "ARE YOU SURE YOU WANT TO DELETE THIS ACCOUNT?",  title: "Warning Message",
            // if option is accepted, reset all data in the file to null
            if (accept == 0)
            {
                data.setAccountNo( 0 );
                data.setFirstName( null );
                data.setLastName( null);
                data.setBalance( 0) ;
                data.setOverdraft( 0 );

                // seek for the requested file in the record
                randAccFile.seek(  pos: (long) ( accountNumber-1 ) * Ass2Record.size() );
                // write new data to the seeked file
                data.write( randAccFile );

                JOptionPane.showMessageDialog( parentComponent: this,  message: "All funds have been withdrawn and account has been deleted");

                resetTextFields(); // method called in to reset/clear text fields
            }
        }
    }//end try

    catch (IOException io)
    {
        System.err.println("error during write to file\n" + io.toString() );
    } // end catch

    updatedAccount(); // method called in to update the account with the new input
} // end delete method
```

public void editTextFields() method

I made this method to set the text fields to be editable as initially are set to not be editable. It is called in at the beginning of addRecord() method, to be able to create new accounts.

```
//==================================================
// method to set text fields to be editable
public void editTextFields()
{
    accountNoField.setEditable(true);
    firstNameField.setEditable(true);
    lastNameField.setEditable(true);
    balanceField.setEditable(true);
    overdraftField.setEditable(true);
} // end resetTextFields method
```

public void resetTextFields() method

It’s used in multiple methods to reset/clear text fields .

```
//==================================================
// method to reset/empty/clear text fields
public void resetTextFields()
{
    accountNoField.setText("");
    firstNameField.setText("");
    lastNameField.setText("");
    balanceField.setText("");
    overdraftField.setText("");
} // end resetTextFields method
```

public void closeFile() method

Used to terminate program when the “Exit Program” button is being pressed

```
//==========================================================================================
// method to close file/end program
private void closeFile()
{
    try
    {
        // message to user when program is terminated
        JOptionPane.showMessageDialog( parentComponent: this, message: "Thank you for using CREDIT UNION! " +
                "\n                    GOODBYE!");
        randAccFile.close();
        System.exit( status: 0 );
    } // end try

    catch (IOException io)
    {
        System.err.println( "File not closed properly\n" + io.toString() );
    } // end catch

    setVisible(false);
} // end closeFile method
```

public void actionPerformed() method

Is the method that executes all the methods from above, when prompted by the user through pressing buttons or menu items.

```
//==========================================================================================
// add actionPerformed for the buttons
public void actionPerformed ( ActionEvent e )
{
    //----------------------------------------
    // create account
    if (e.getSource() == create) // when "create" JMenuItem is pressed
    {
        addRecord(); // method called in
    }

    //----------------------------------------
    // read account details when account number entered
    if (e.getSource() == select) // when "view" JMenuItem is pressed
    {
        try
        {
            readRecordByInput(); // method called in
        } // end try

        catch ( NumberFormatException nfe ) // number format exception catch
        {
            JOptionPane.showMessageDialog( parentComponent: this, message: "Invalid input. Please enter an integer number between 1 and 100.");
        } // end catch

    }
```

```java
//----------------------------------------
//update balance by making a lodgement
if (e.getSource() == lodge) // when "lodge" JMenuItem is pressed
{
    try
    {
        lodgement(); // method called in
    } // end try

    catch ( NumberFormatException nfe ) // number format exception catch
    {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Go to \"Menu\" -> \"Select Existing Account\" and input an account number. " +
                "\nWhen the account details are displayed, click \"Lodgement\".");
    } // end catch
}

//----------------------------------------
// method to withdraw money from the account
if (e.getSource() == withdraw) // when "withdraw" JMenuItem is pressed
{
    try
    {
        withdrawal(); // method called in
    } // end try

    catch ( NumberFormatException nfe ) // number format exception catch
    {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Go to \"Menu\" -> \"Select Existing Account\" and input an account number. " +
                "\nWhen the account details are displayed, click \"Withdrawal\".");
    } // end catch
}
```

```java
//----------------------------------------
//method to request an overdraft adjustment
if (e.getSource() == overdraftReq) // when "overdraftReq" JMenuItem is pressed
{
    try
    {
        reqOverdraft(); // method called in
    } // end try

    catch ( NumberFormatException nfe ) // number format exception catch
    {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Go to \"Menu\" -> \"Select Existing Account\" and input an account number. " +
                "\nWhen the account details are displayed, click \"Overdraft Adjustment\".");
    } // end catch
}

//----------------------------------------
//read account details when account number entered and delete account
if (e.getSource() == delete) // when "delete" JMenuItem is pressed
{
    try
    {
        delete(); // method called in
    } // end try

    catch ( NumberFormatException nfe ) // number format exception catch
    {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Go to \"Menu\" -> \"Select Existing Account\" and input an account number. " +
                "\nWhen the account details are displayed, click \"Delete Account\".");
    } // end catch

}
```

```
    //-----------------------------------------
    // clear textfield
    if (e.getSource() == clear) // when "clear" JButton is pressed
    {
        resetTextFields(); // method called in
    }

    //-----------------------------------------
    // save last input and close program
    if (e.getSource() == done) // when "done" JButton is pressed
    {
        closeFile(); // method called in
    }
}
```

Main method

This method calls in the constructor and runs the entire program.

```
    //==========================================================================
    //Instantiate a Ass2ReadWrite object and start the program by calling in the main method
    public static void main(String[] args )
    {
        Ass2ReadWrite app = new Ass2ReadWrite(); // calling in the constructor
        app.addWindowListener(new WindowAdapter(){public void windowClosing(WindowEvent e) {System.exit( status: 0);}}); // assist with closing the window
    } // end main method
}
```

Illustration of the program running

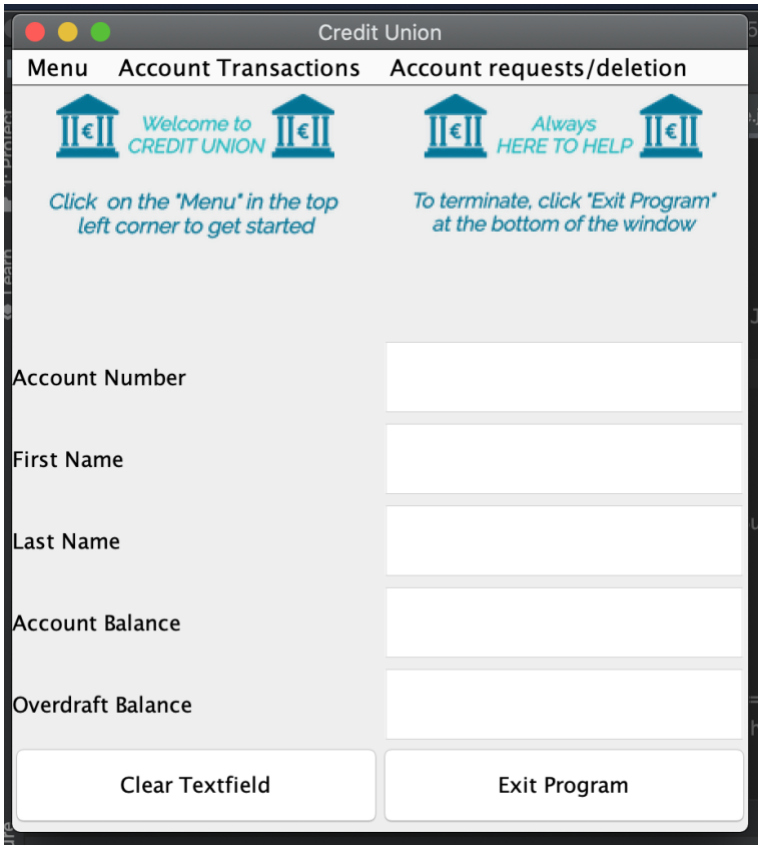
Interface

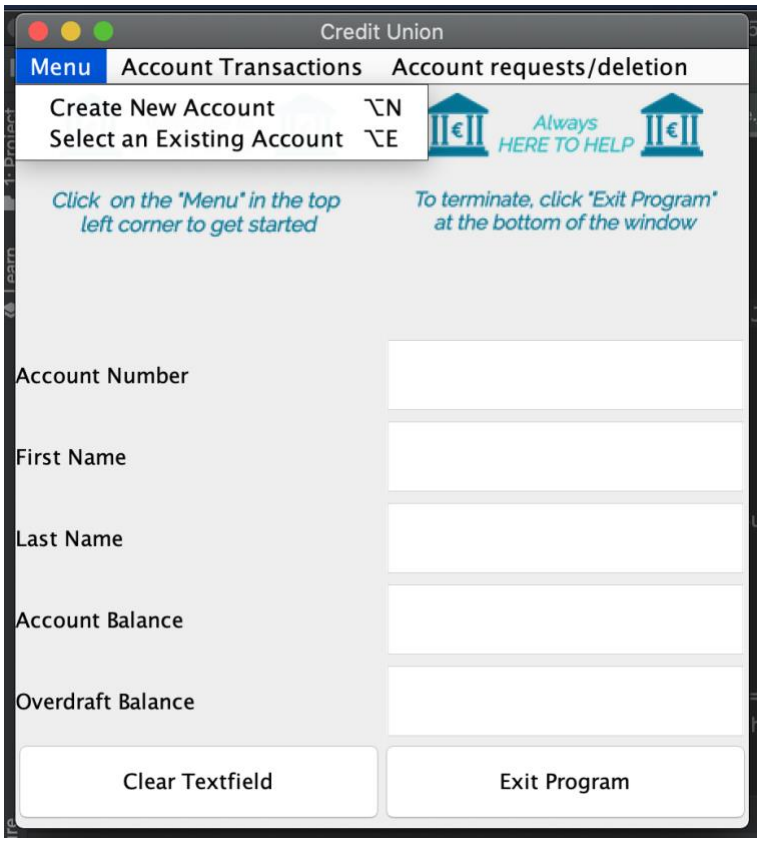
I'll present here the look and feel of the program.

First we have a welcome message at the top of the page and another one prompting the user to click the "Menu" option to get started. If we take a closer look, we will notice that at this stage the text fields are not editable. After clicking the "Menu" option, a dropdown will reveal 2 other options. One to create new accounts and the other to select existing accounts.

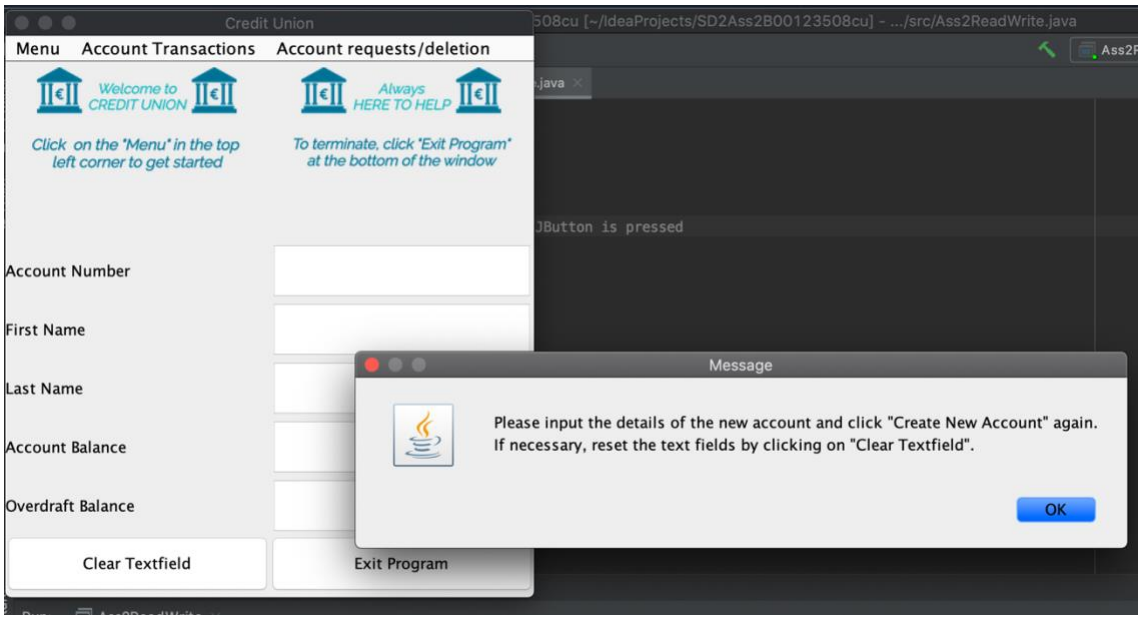
Create new account

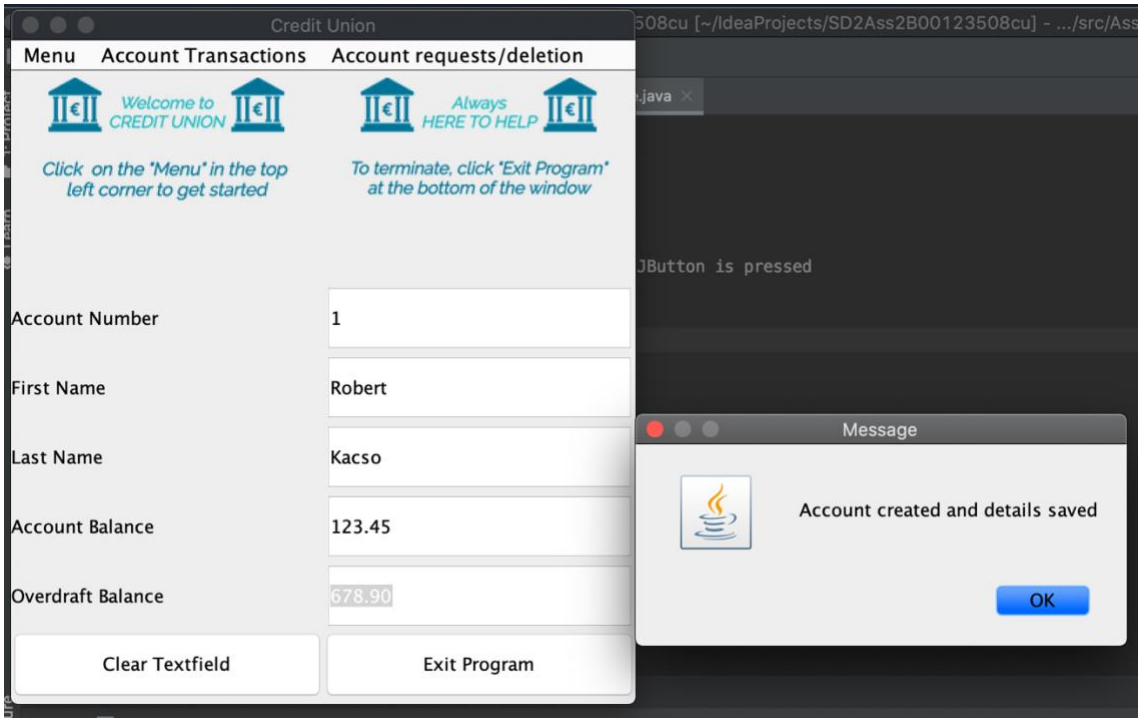
Let's have a look:


Credit Union
Menu
Account Transactions
Account requests/deletion
Welcome to CREDIT UNION
Always HERE TO HELP
Click on the "Menu" in the top left corner to get started
To terminate, click "Exit Program" at the bottom of the window
Account Number
First Name
Last Name
Account Balance
Overdraft Balance
Clear Textfield
Exit Program


Credit Union
Menu
Account Transactions
Account requests/deletion
Create New Account ⌥N
Select an Existing Account ⌥E
Always HERE TO HELP
Click on the "Menu" in the top left corner to get started
To terminate, click "Exit Program" at the bottom of the window
Account Number
First Name
Last Name
Account Balance
Overdraft Balance
Clear Textfield
Exit Program

By clicking “Create New Account” (or we can use the shortcut keys too, in this case Alt+N) we get prompted to input the details of the new account into the text fields, which now have become editable, and click “Create New Account” again to save it. Let’s do that.



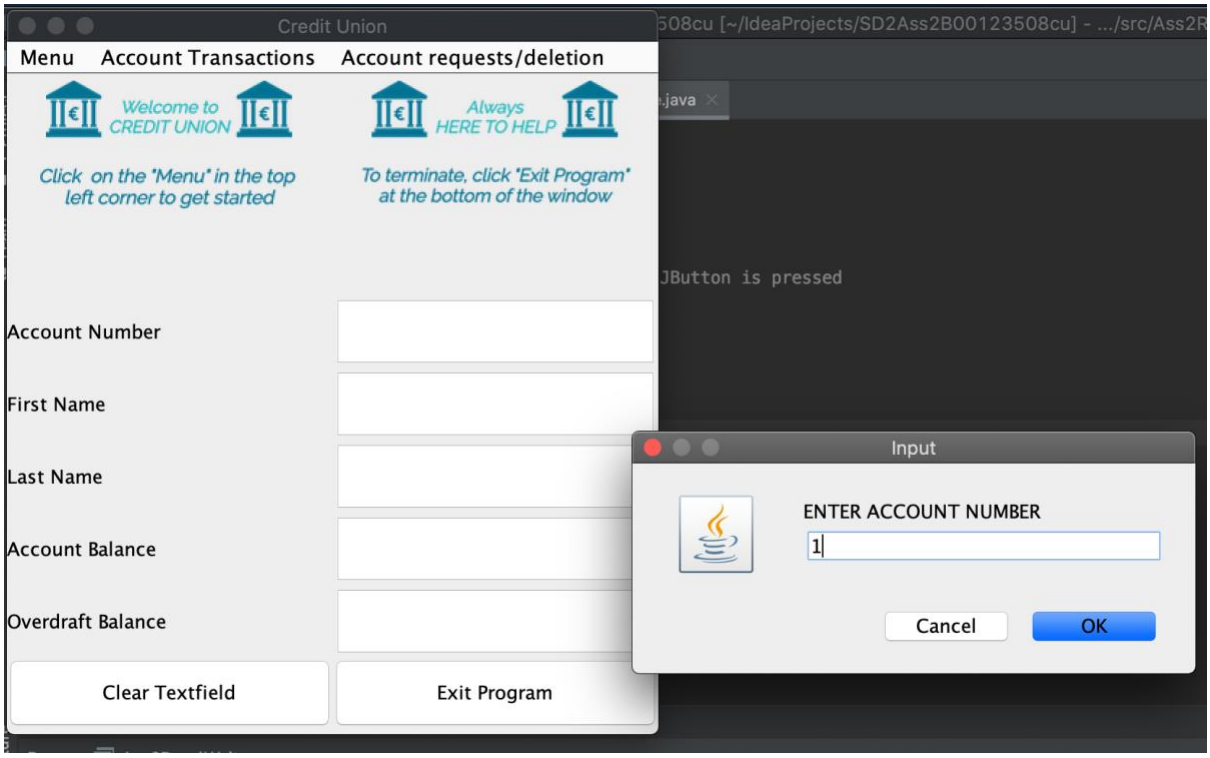


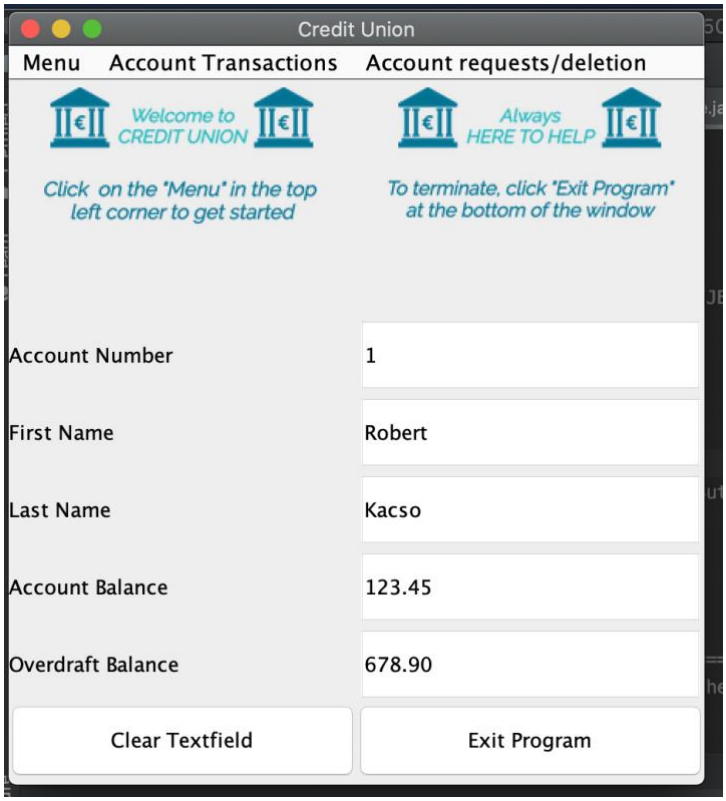
After we create the account and click OK, the text fields remain open to create another account if needed.

Select existing account

If we want to see the account we created we click on the “Menu” option once again and click “Select Existing Account”.

When we do, a pop-up message will prompt to input the account number. After that the account details of the said account number, will be displayed with the text fields NOT being editable, so details can't be changed directly in the frame window.





Next, we can make transactions or request a different overdraft amount or even delete the account. Let's start we some transactions.

Lodgement

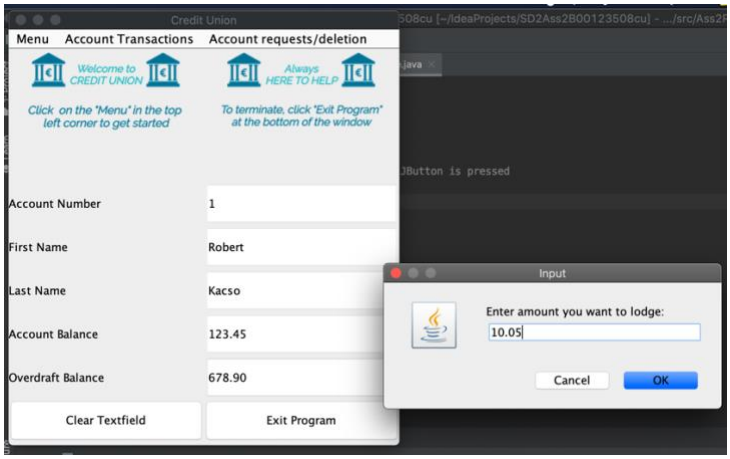
First we make a lodgement of a positive as well as a negative amount. Tellers can make mistakes, so the program should not allow to lodge a negative amount as that would actually result in a withdrawal. Same applies for withdrawals and overdraft request.
Ex:
Lodgement: 2 + (-2) = 0. Lodging a negative number will in fact deduct it. A “+” and a “-“ will make a “-“.

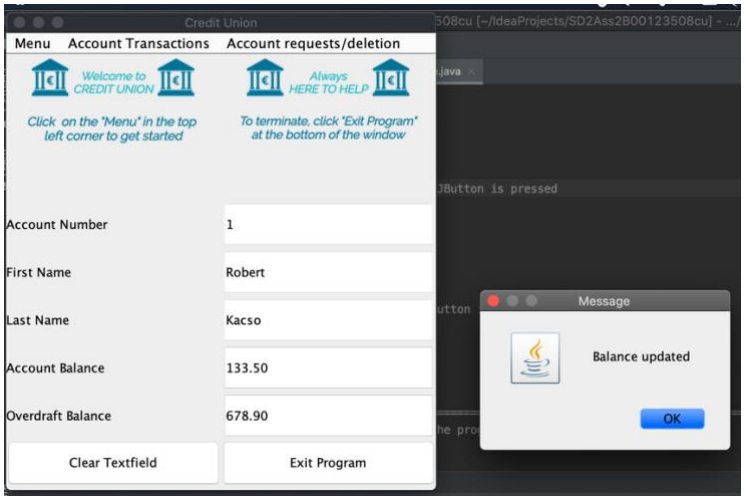
Withdrawal: 2 – (-2) = 4. Withdrawing a negative number will in fact result in addition. A “-“ and another “-“ will make a “+”.

Overdraft means that the account holder is in negative already, so if we have, let’s say an overdraft amount of €100 that means that we can take out from our account until we reach € - 100. No need to input the negative before the amount.

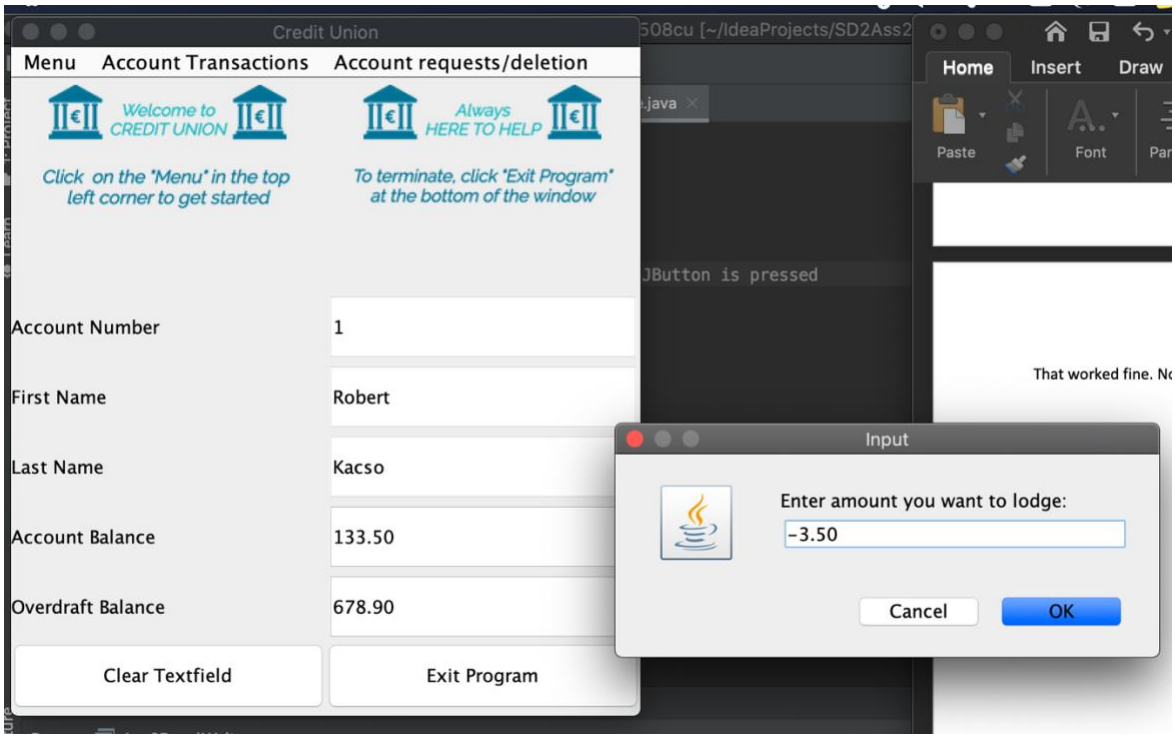
Let’s test all of this in our program. When I say our, I actually mean MY program, as I designed it ☺

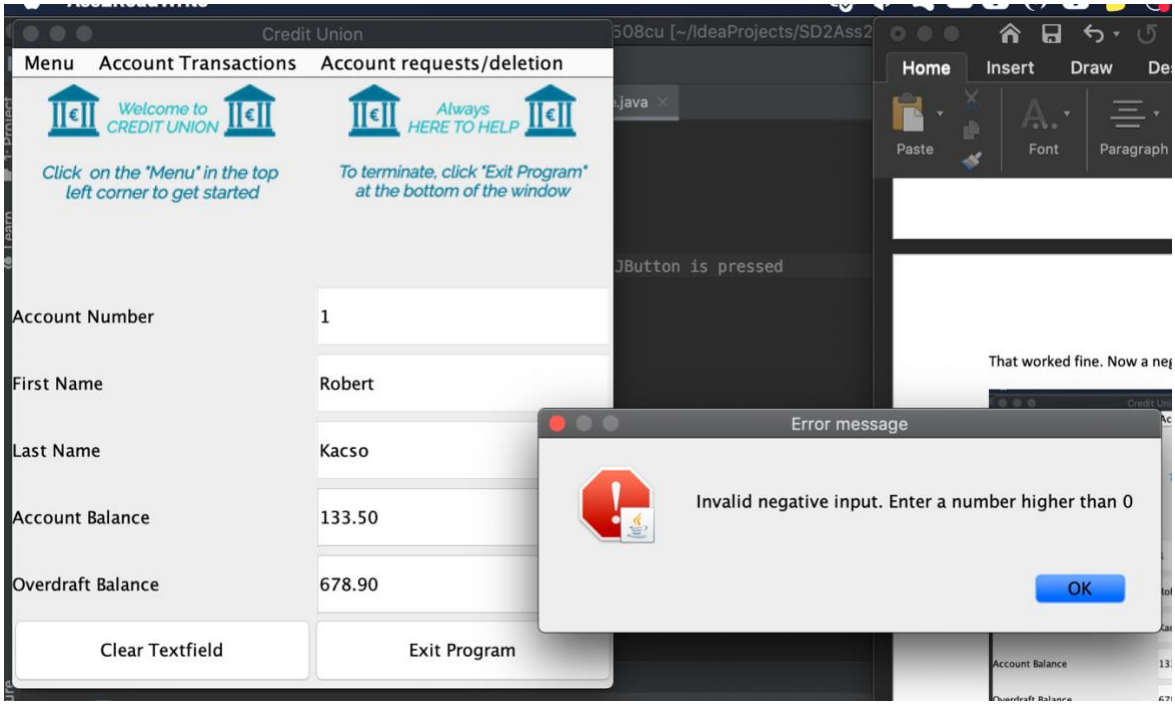
First a positive lodgement:

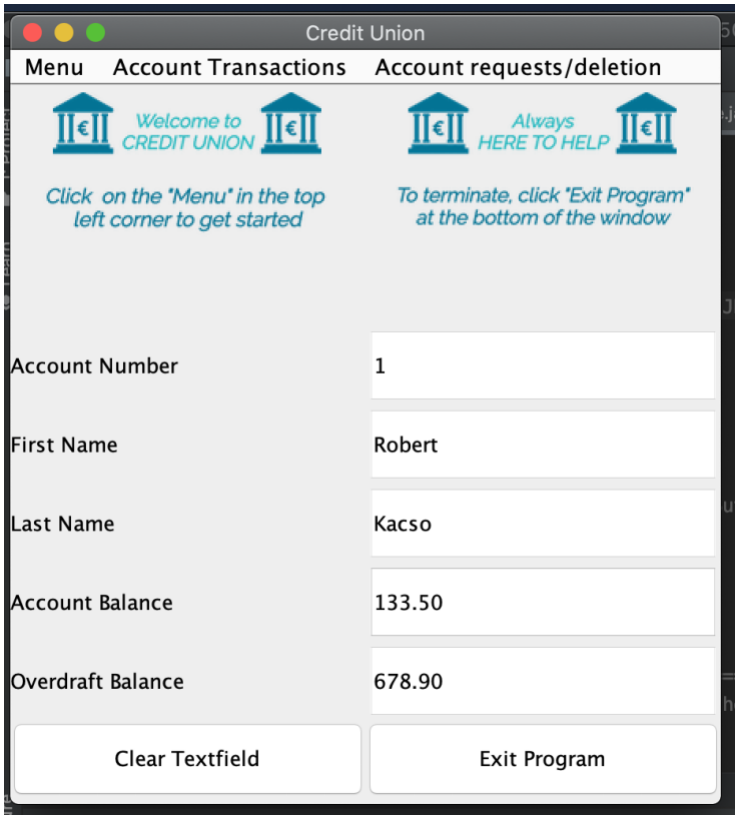




That worked fine. Now a negative number:





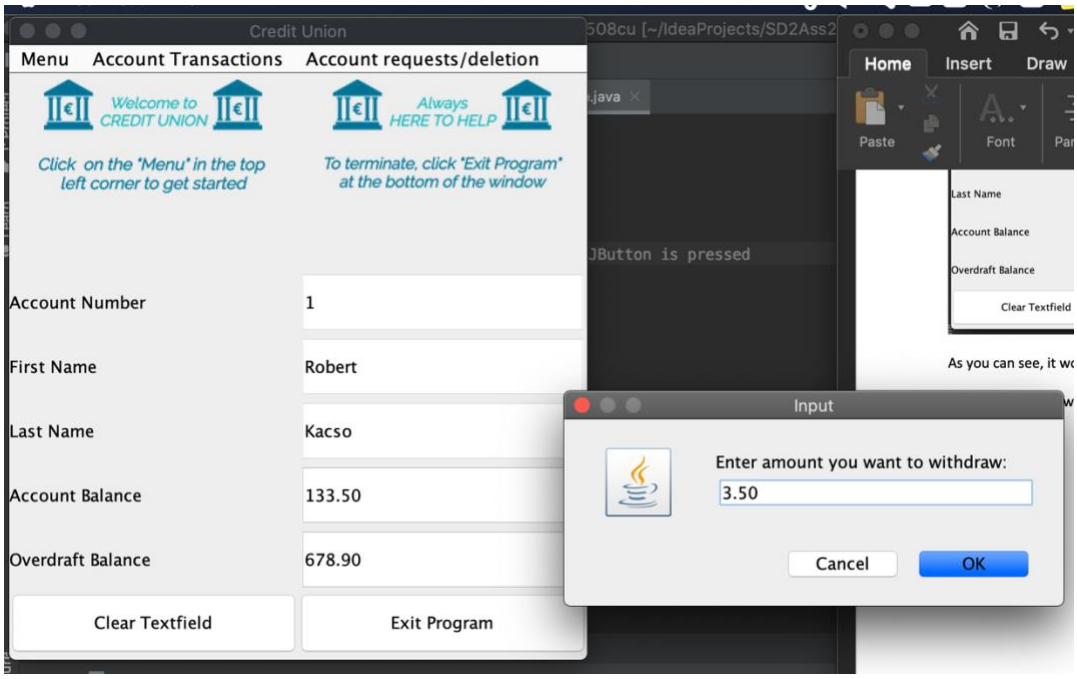


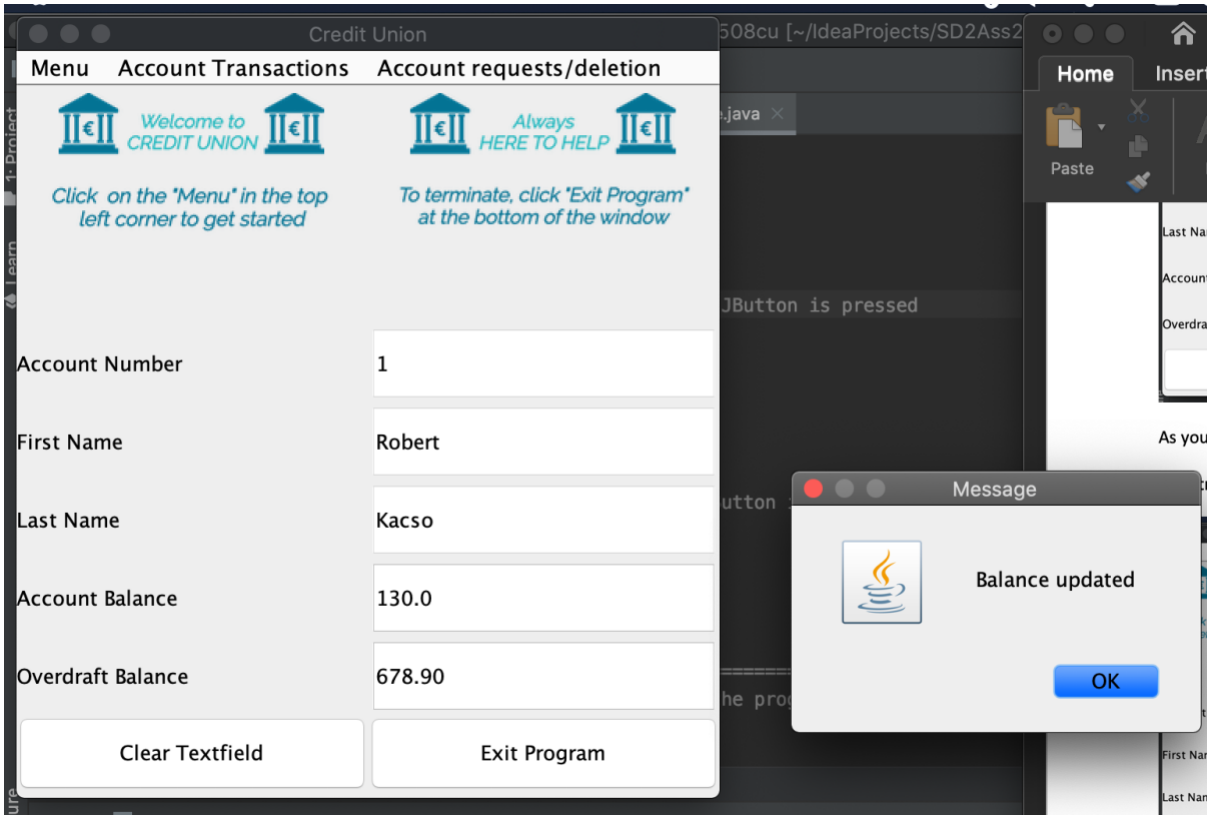
As you can see, it won't allow it and the balance doesn't get updated.

Withdrawal

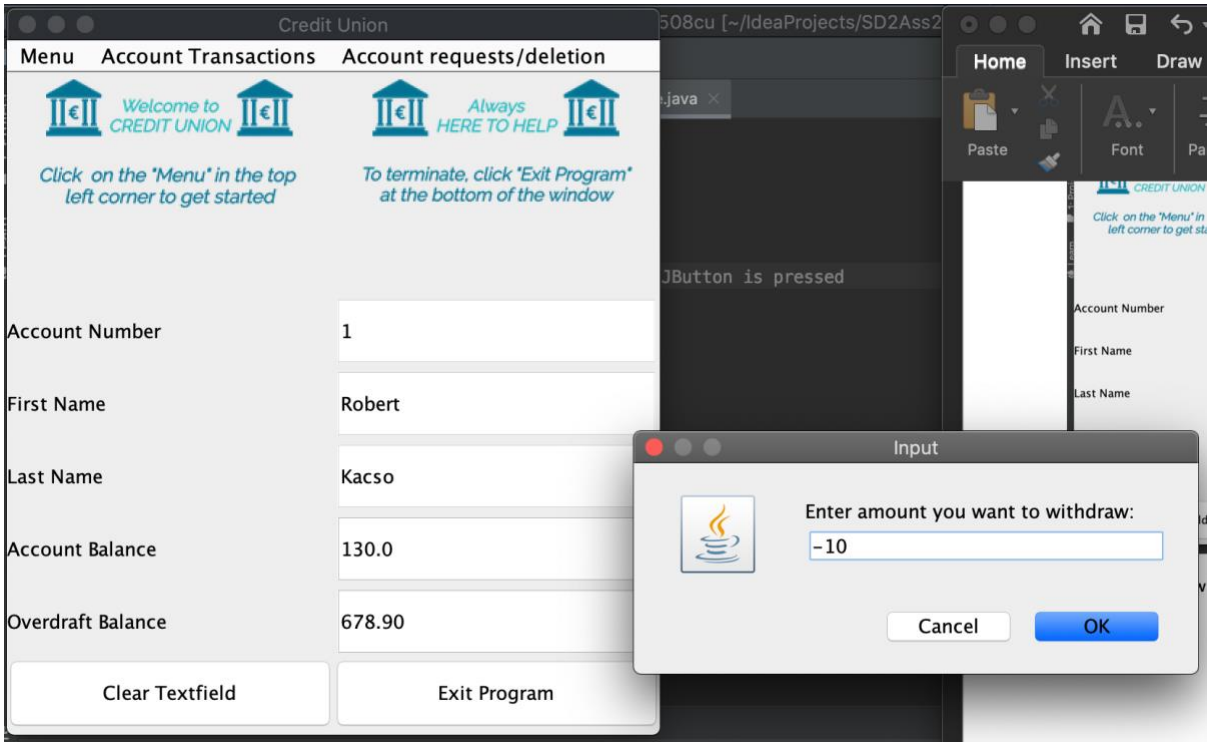
Let's try the same with withdrawal and changing the overdraft amount.

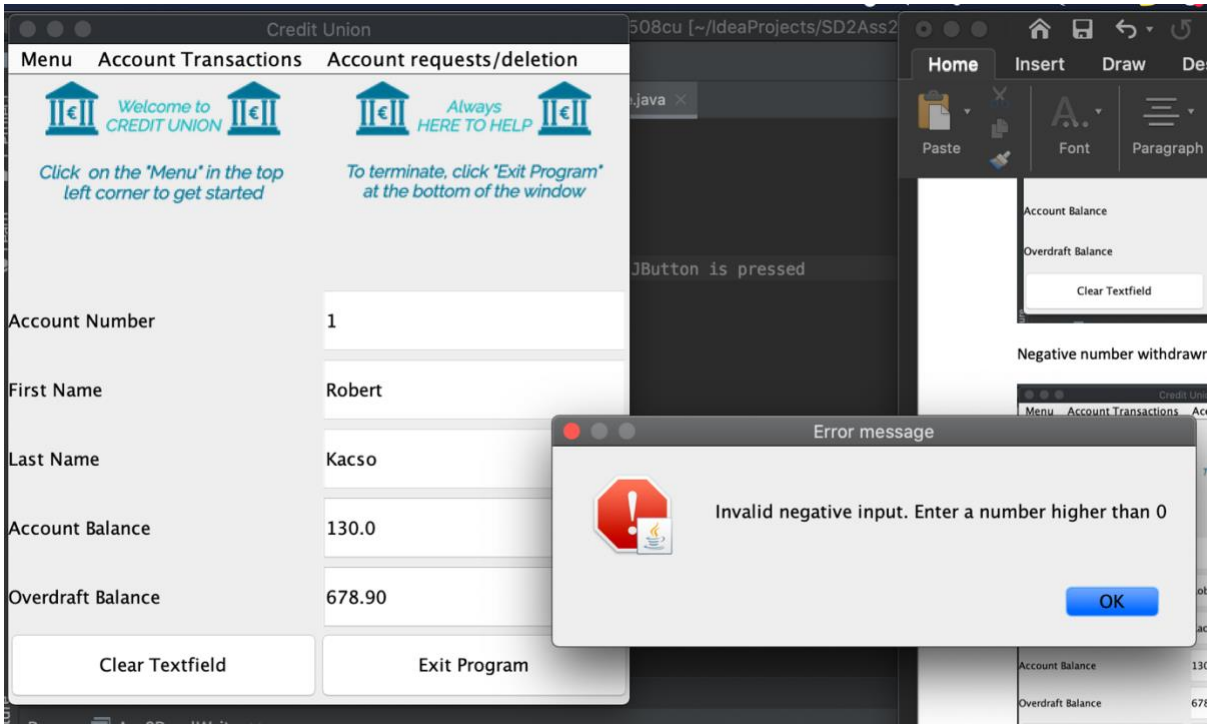
Positive number withdrawn:

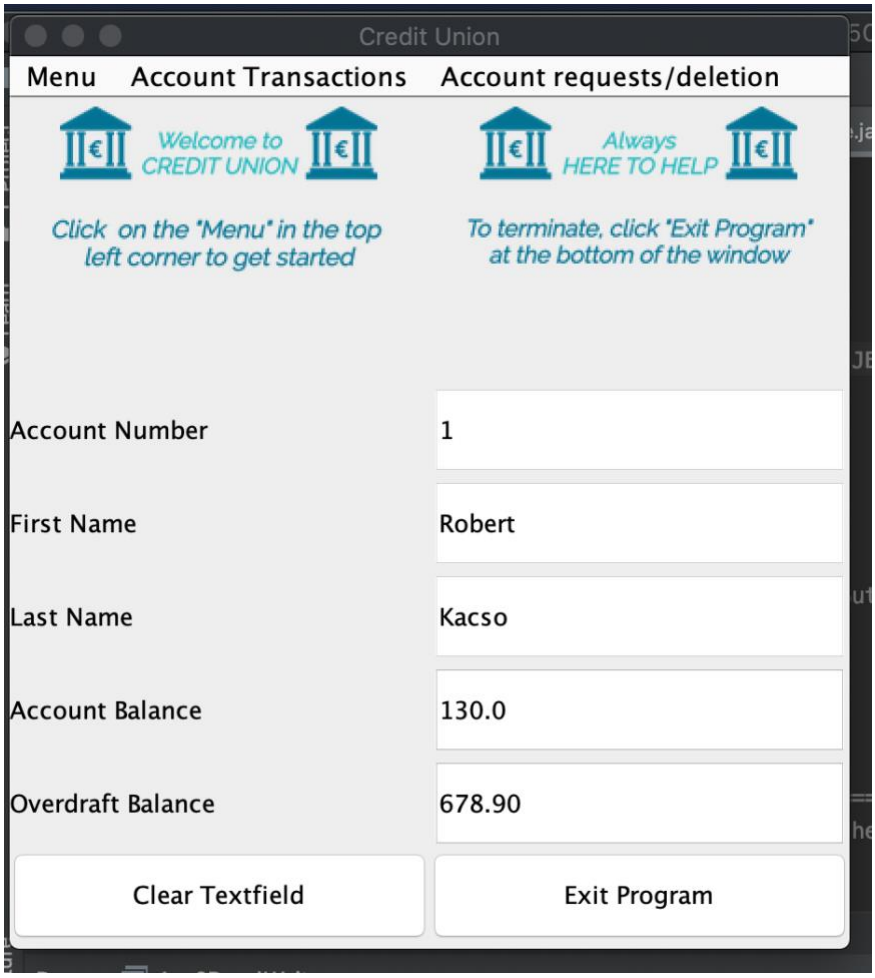




Negative number withdrawn:



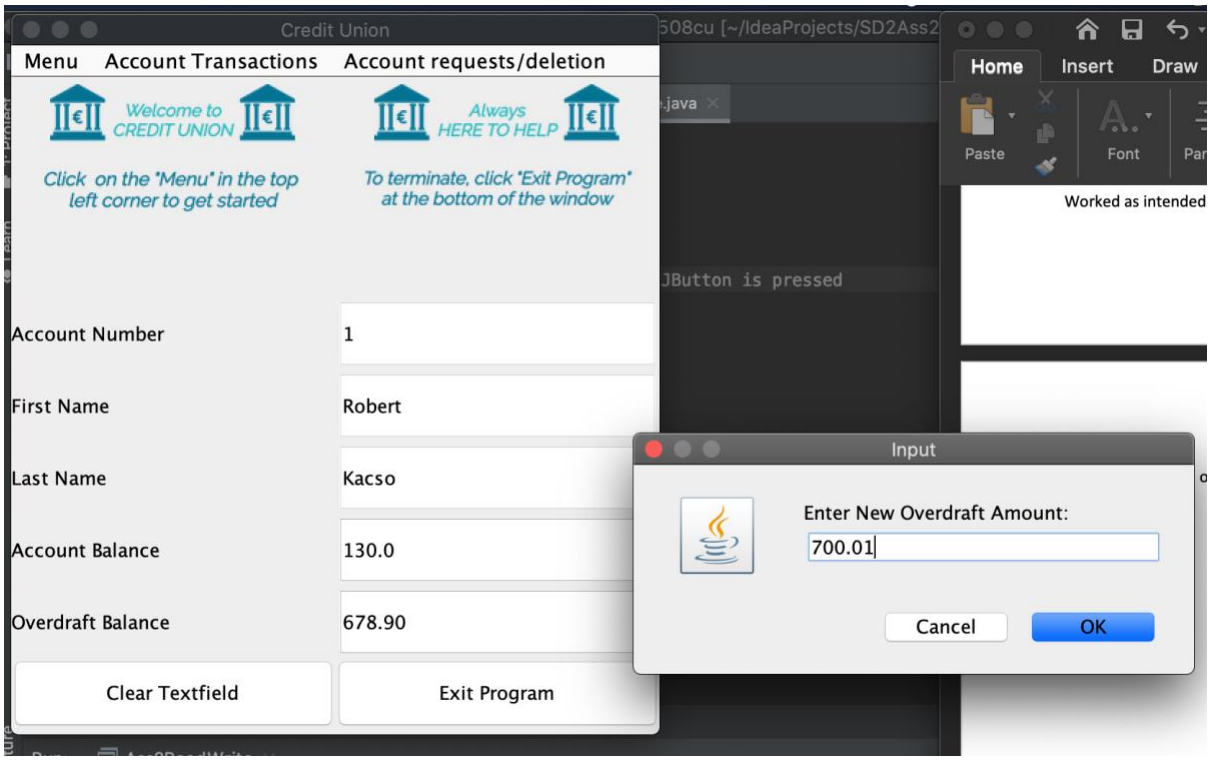


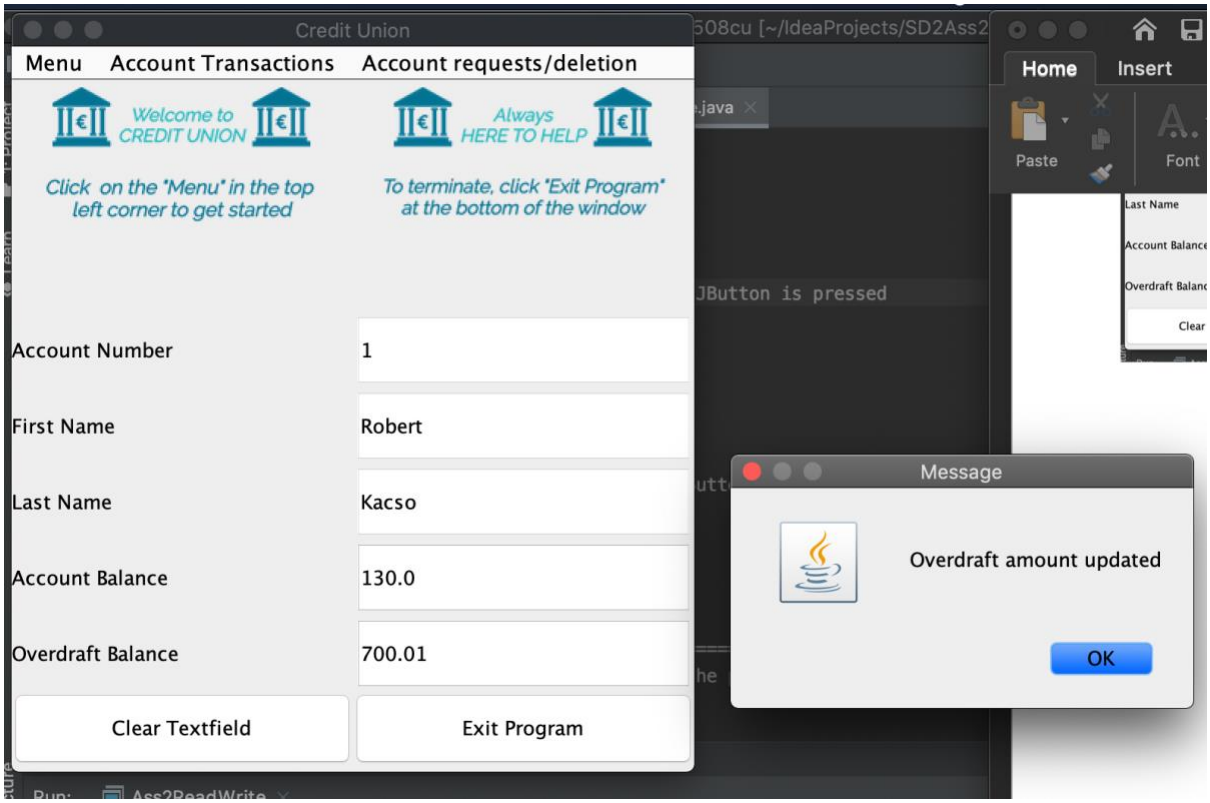


Worked as intended.

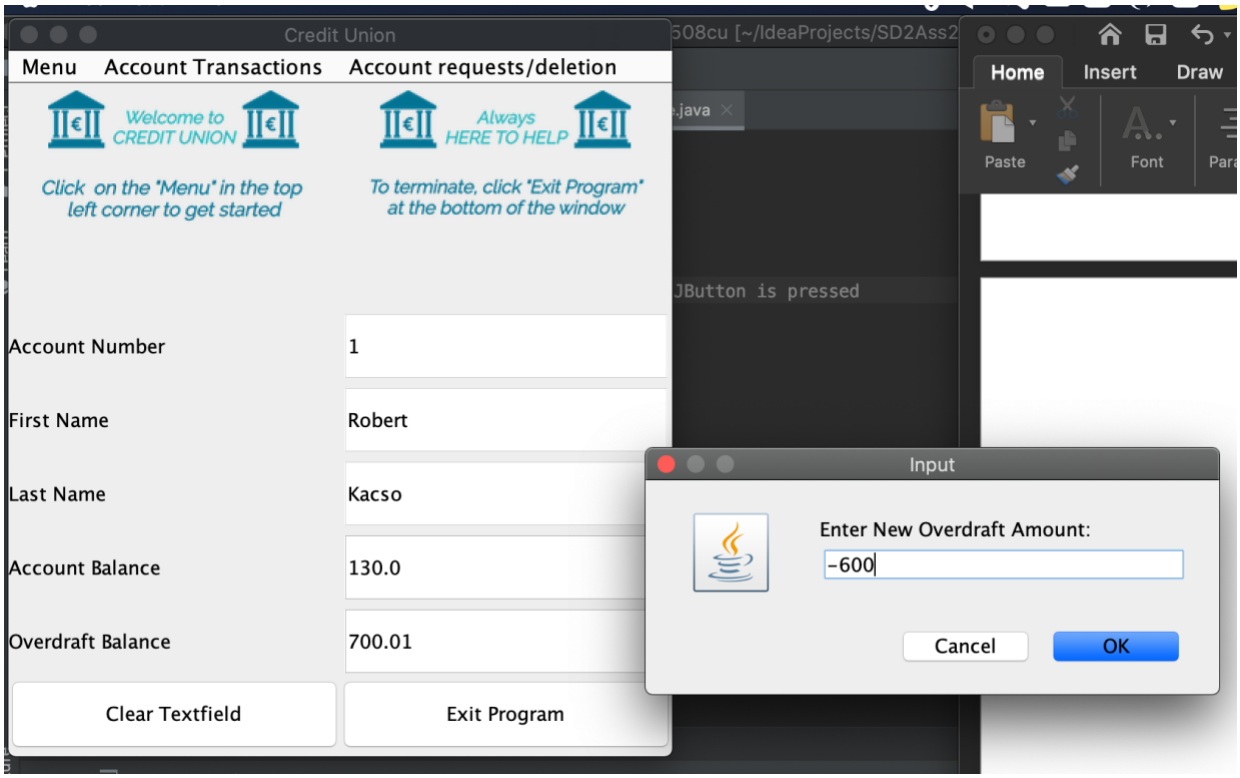
Overdraft

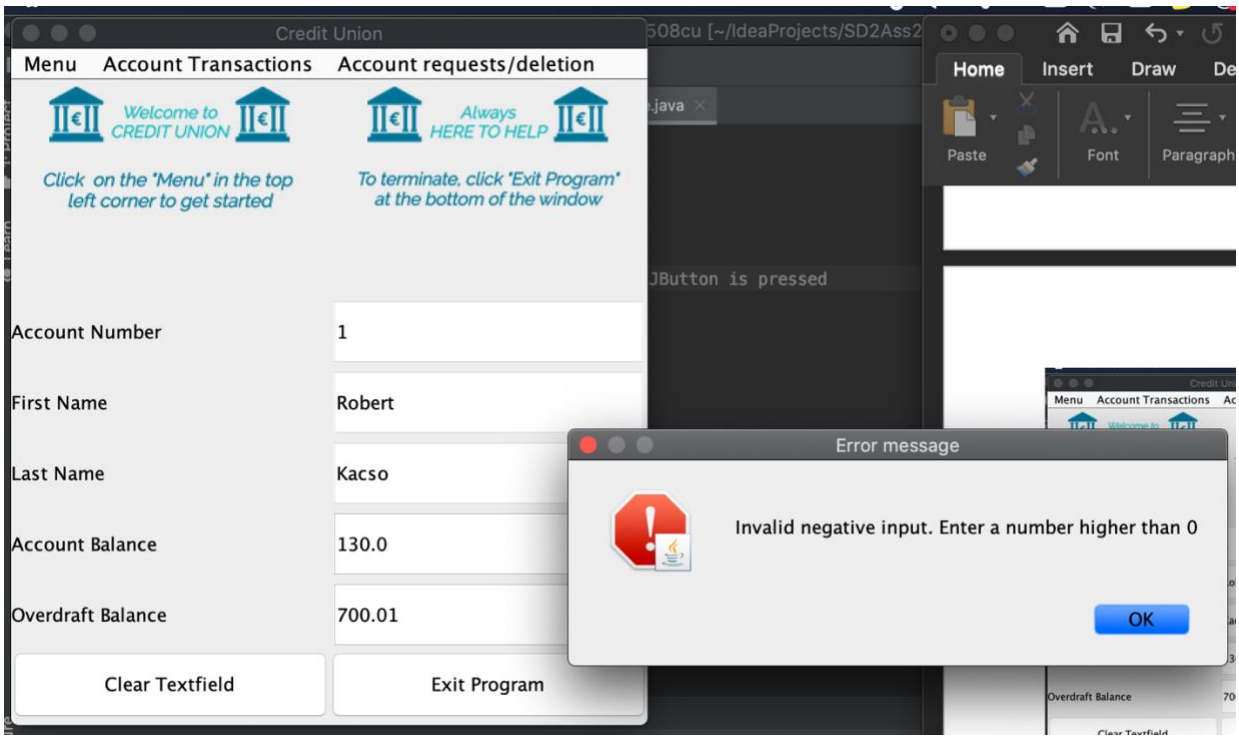
Now, changing the overdraft amount, using a positive number:

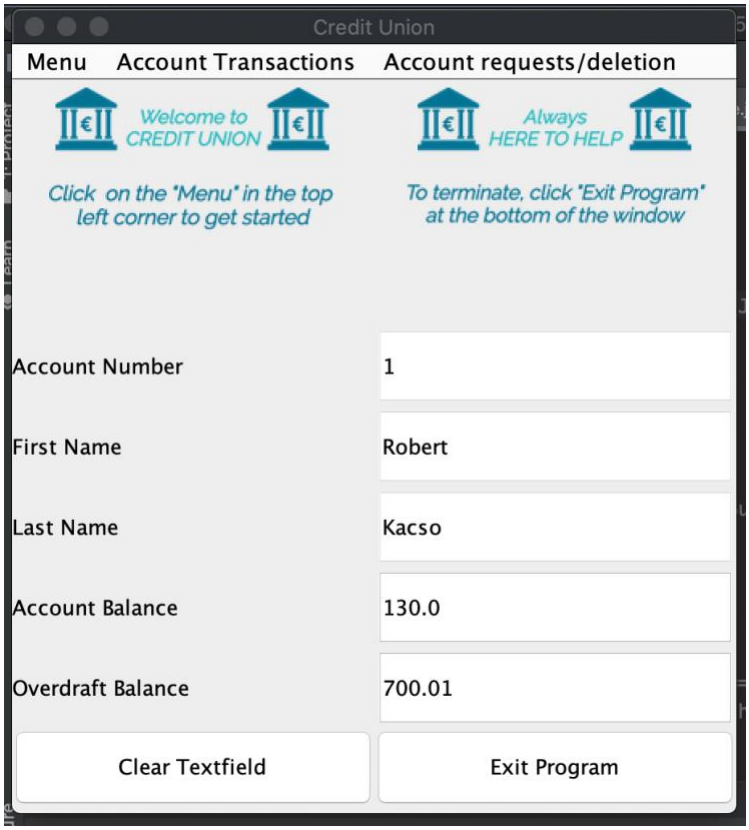




And now using a negative number:







All worked as it should.

Overdrawing and deleting an account

Next, we will try to delete and account number. In order to delete an account, the account holder has to withdraw any amount that they have in their account or to pay of the overdraft first and cancel any overdraft approved on their account.

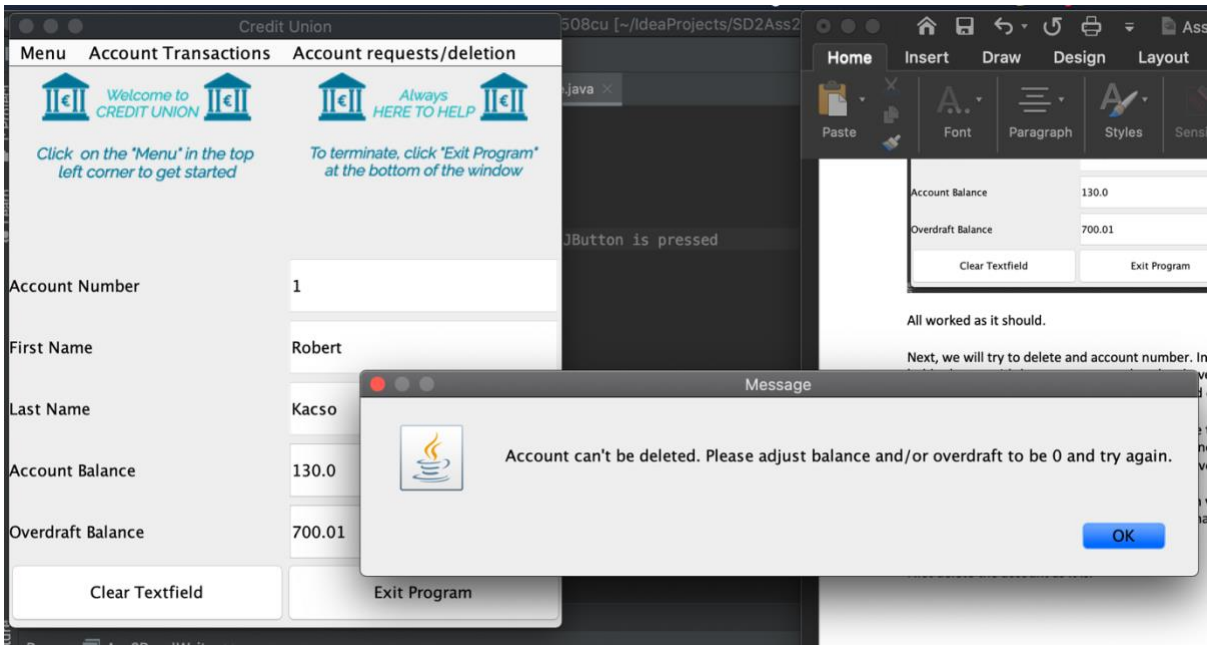
Here we will test multiple things. Trying to delete the account with money still in it, without money in it but still an overdraft approved, with negative balance, meaning that the account holder is in overdraft and with the balance and overdraft zero.

In the same time we will see what happens when we withdraw more than our available balance and I'm going to try to withdraw more than all available funds including balance and overdraft.

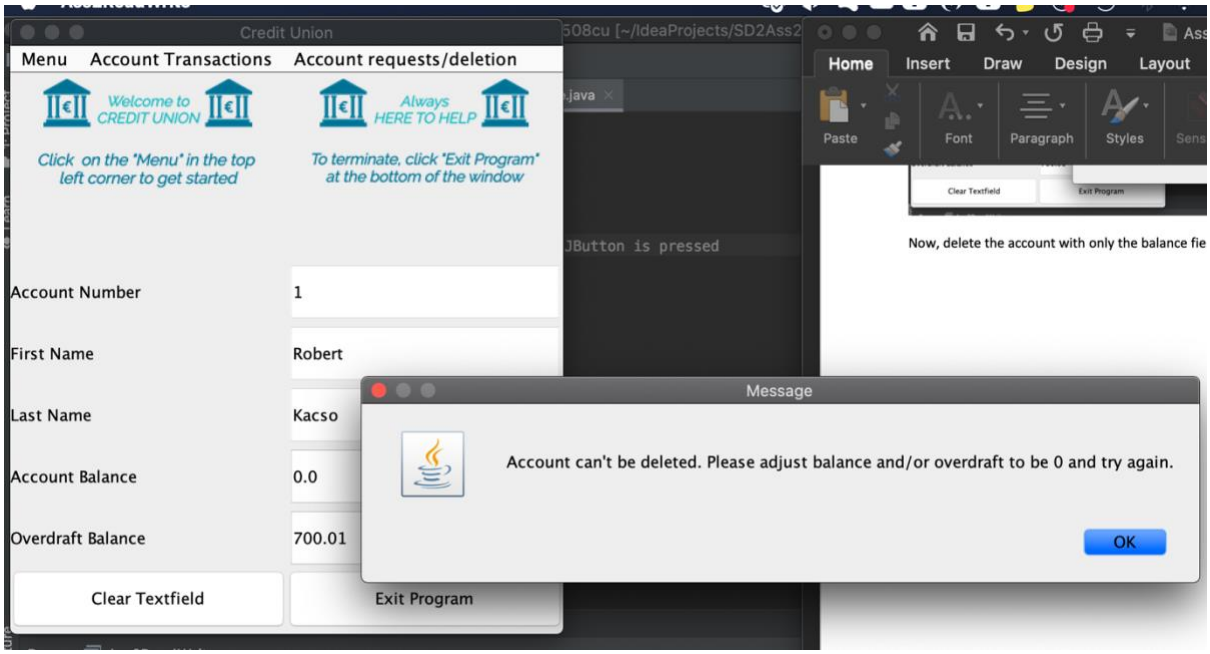
In order to do all these tests, I will lodge and withdraw amounts sometimes, without showing it. For example, further down I will withdraw the entire balance to make it zero, without adjusting the overdraft to zero, before I will show that the account can't be deleted without the overdraft being adjusted to zero too. Same, I will lodge money back to adjust the balance back to zero from a negative balance, before finally deleting the account.

Let's get started.

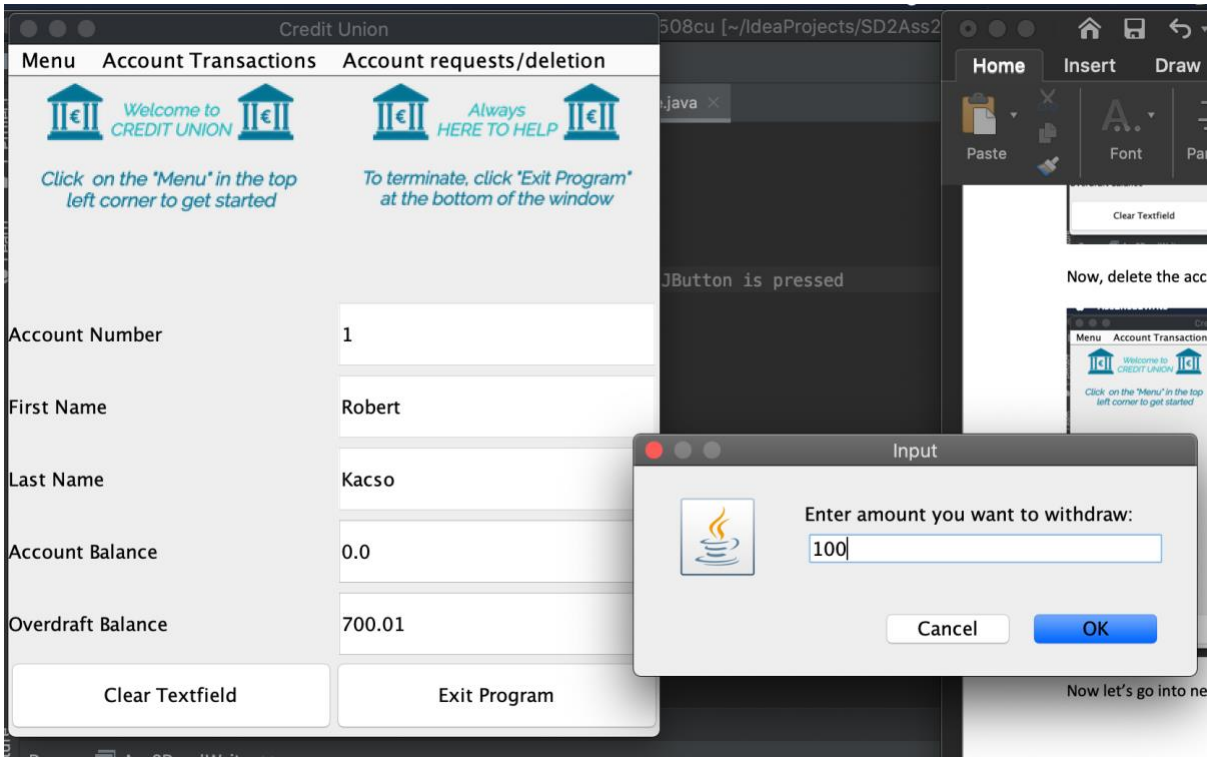
First delete the account as it is:

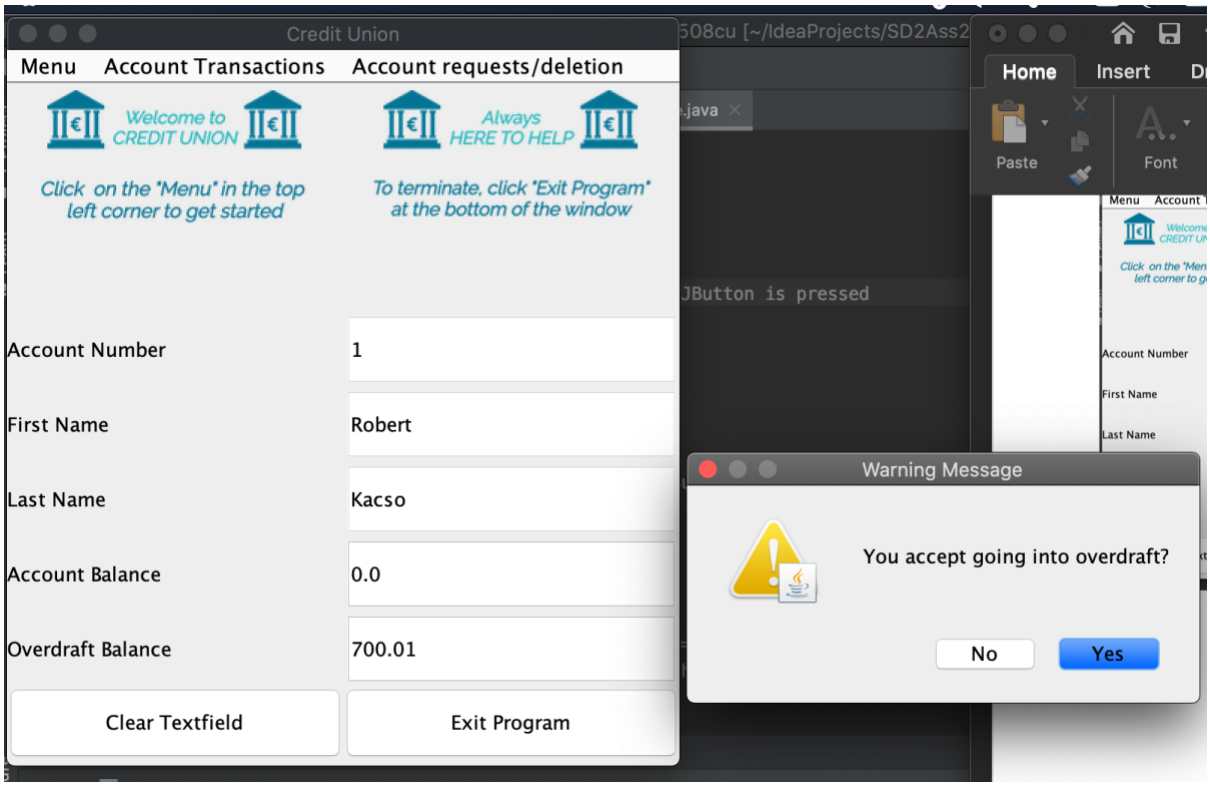


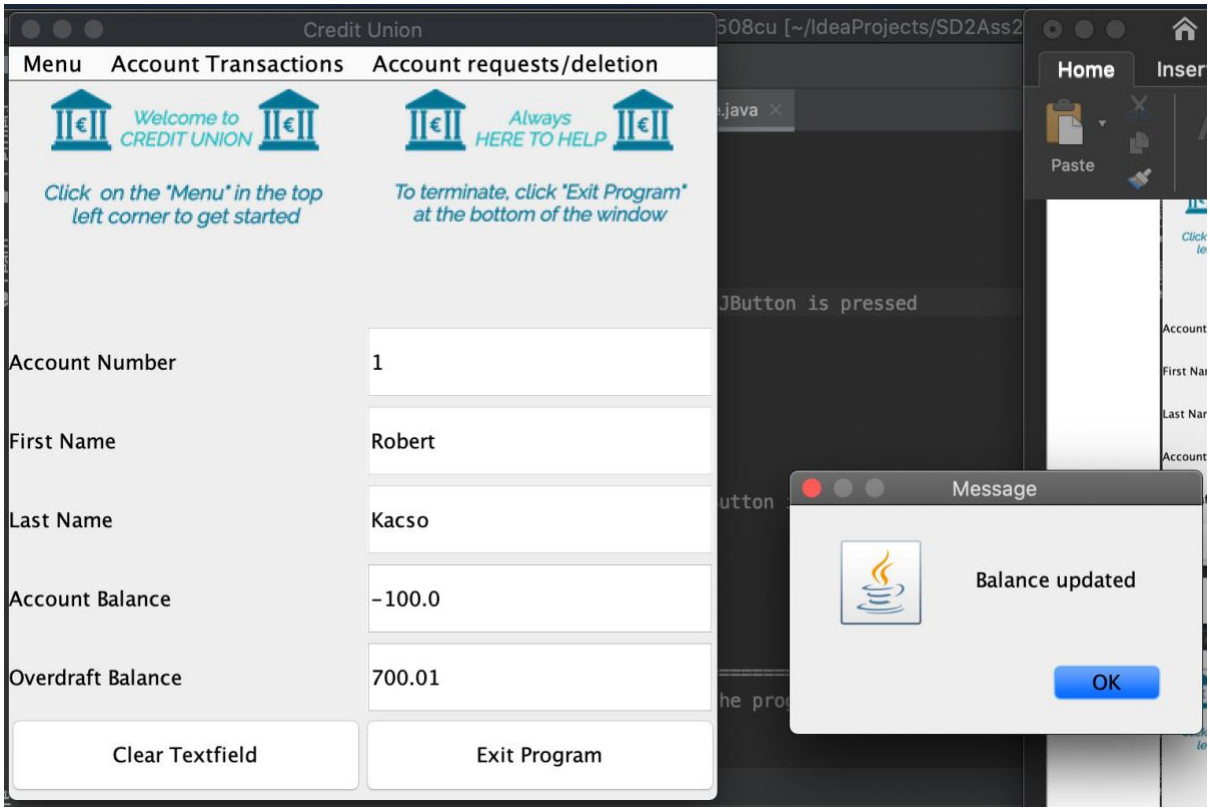
Now, delete the account with only the balance field being zero:

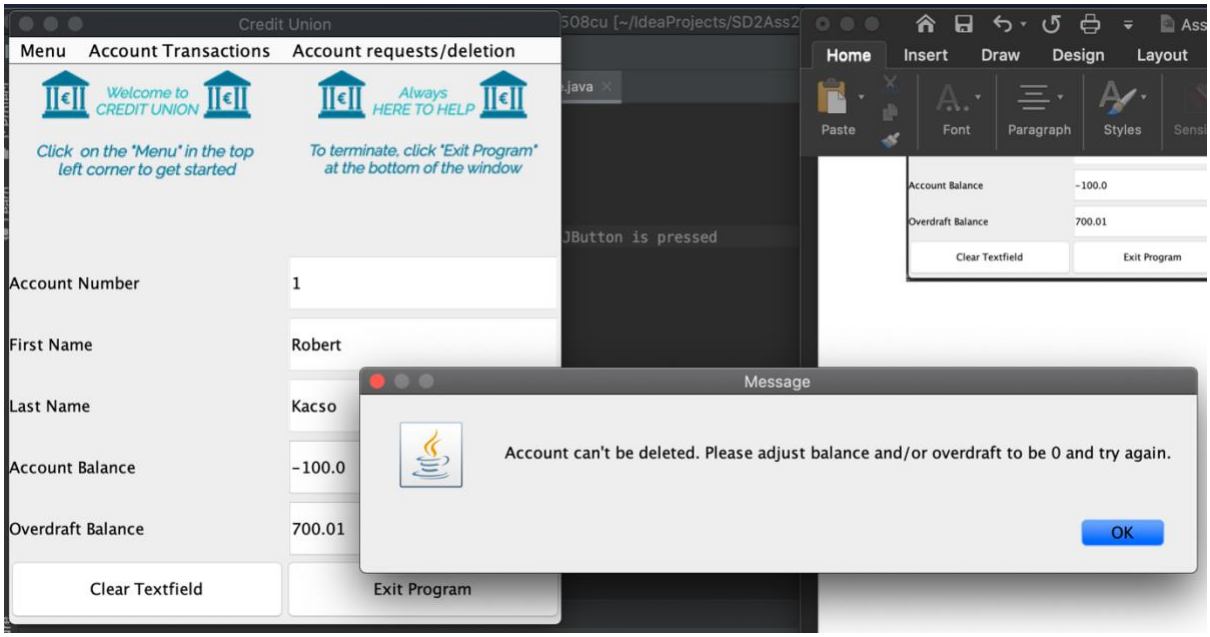


Now let's go into negative balance an try to delete the account:

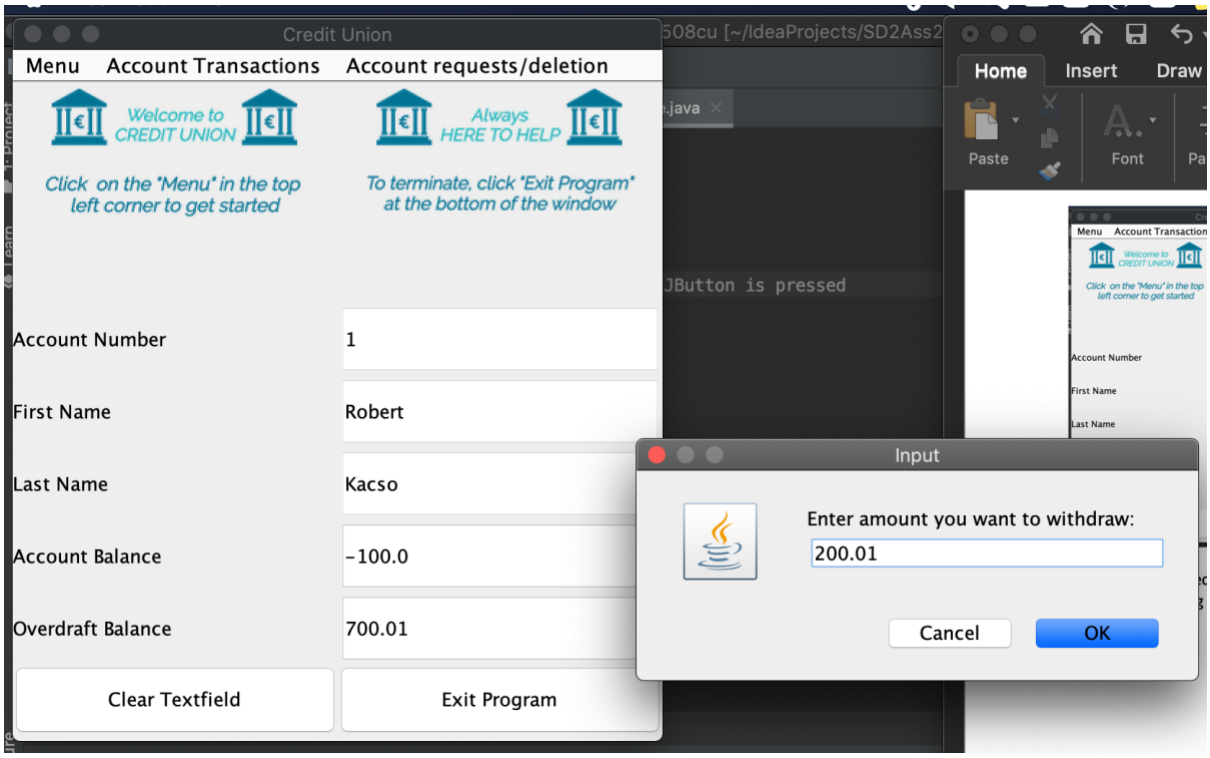

Credit Union
Menu
Account Transactions
Account requests/deletion
Welcome to CREDIT UNION
Always HERE TO HELP
Click on the "Menu" in the top left corner to get started
To terminate, click "Exit Program" at the bottom of the window
Account Number
1
First Name
Robert
Last Name
Kacso
Account Balance
0.0
Overdraft Balance
700.01
Clear Textfield
Exit Program
JButton is pressed
Input
Enter amount you want to withdraw:
100
Cancel
OK
Home
Insert
Draw
Paste
Font
Now, delete the acc
Now let's go into ne

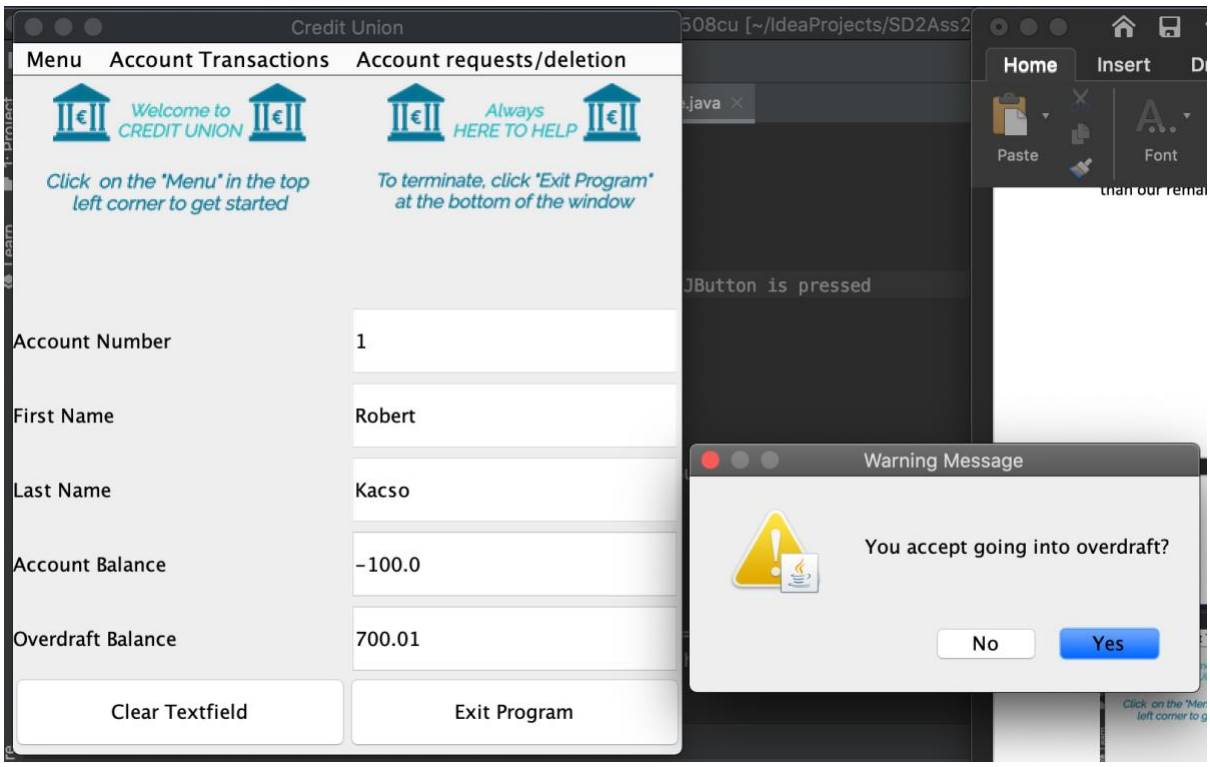

Credit Union
Menu
Account Transactions
Account requests/deletion
Welcome to CREDIT UNION
Always HERE TO HELP
Click on the "Menu" in the top left corner to get started
To terminate, click "Exit Program" at the bottom of the window
Account Number
1
First Name
Robert
Last Name
Kacso
Account Balance
0.0
Overdraft Balance
700.01
Clear Textfield
Exit Program
JButton is pressed
Warning Message
You accept going into overdraft?
No
Yes
Home
Insert
Paste
Font

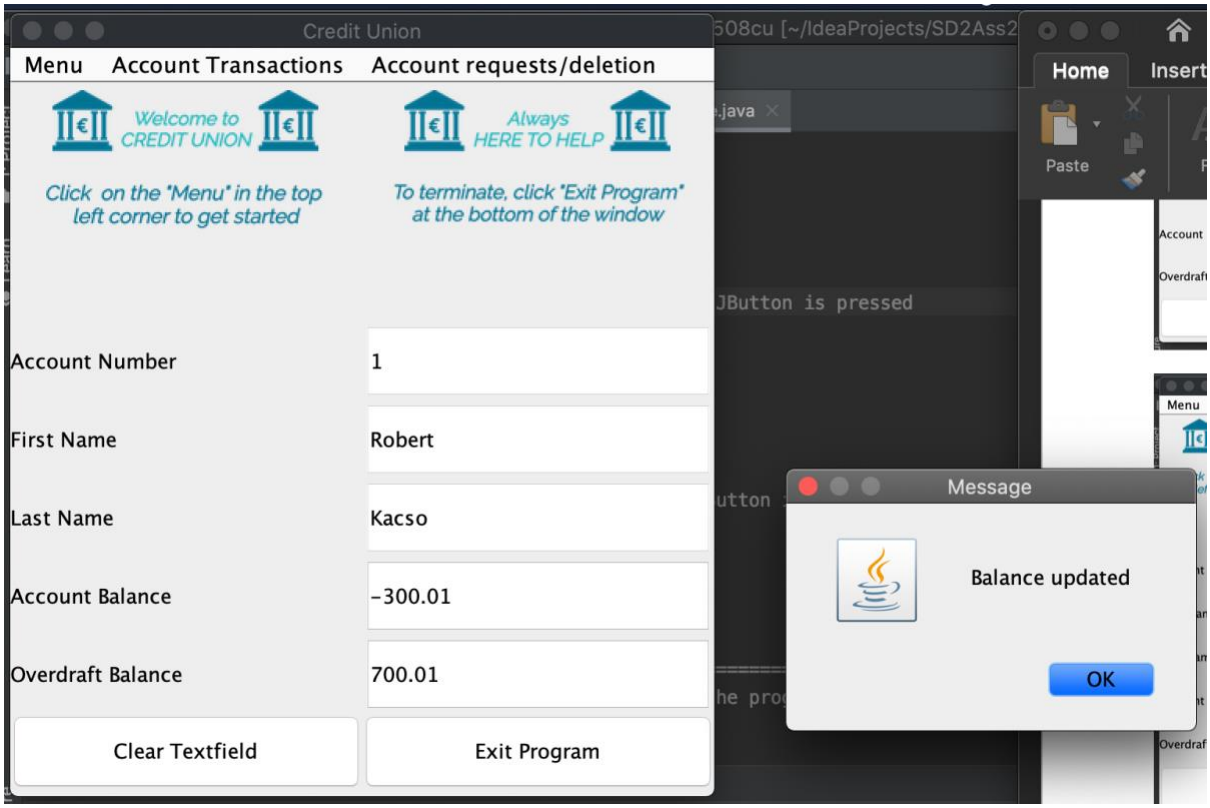




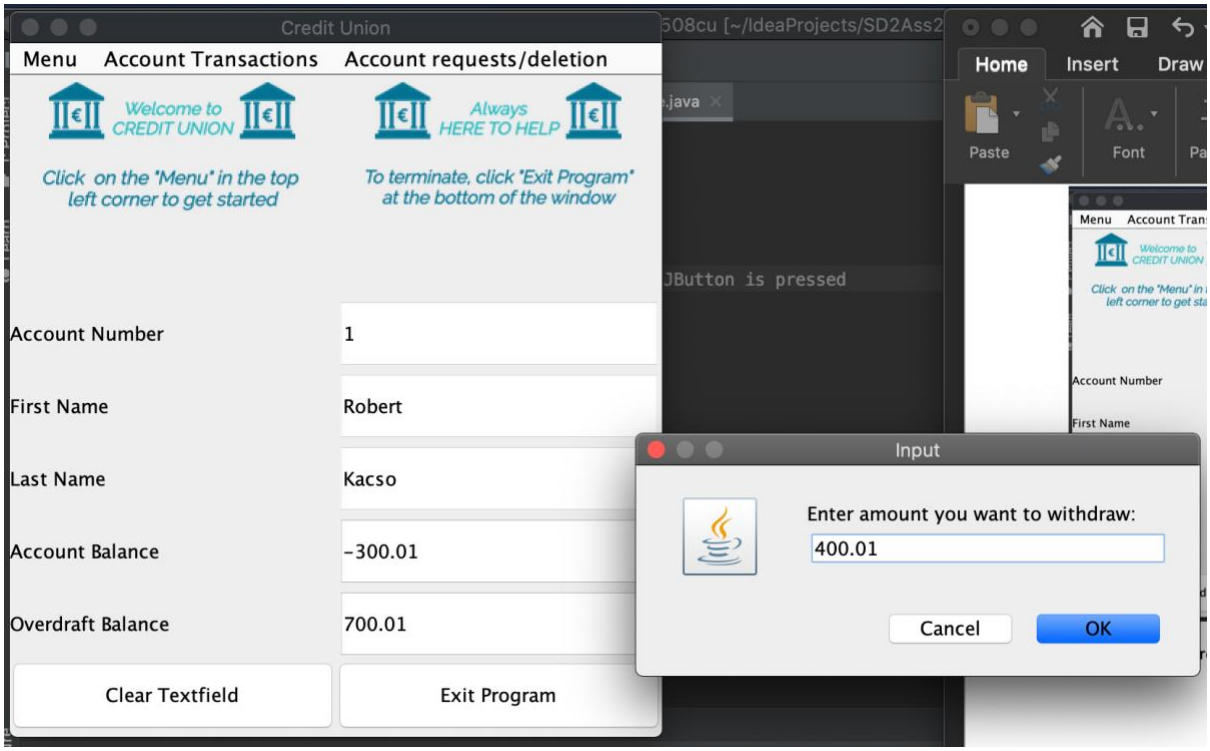
It works as intended. Let's try to go further into overdraft and then try and take out more than our remaining funds.

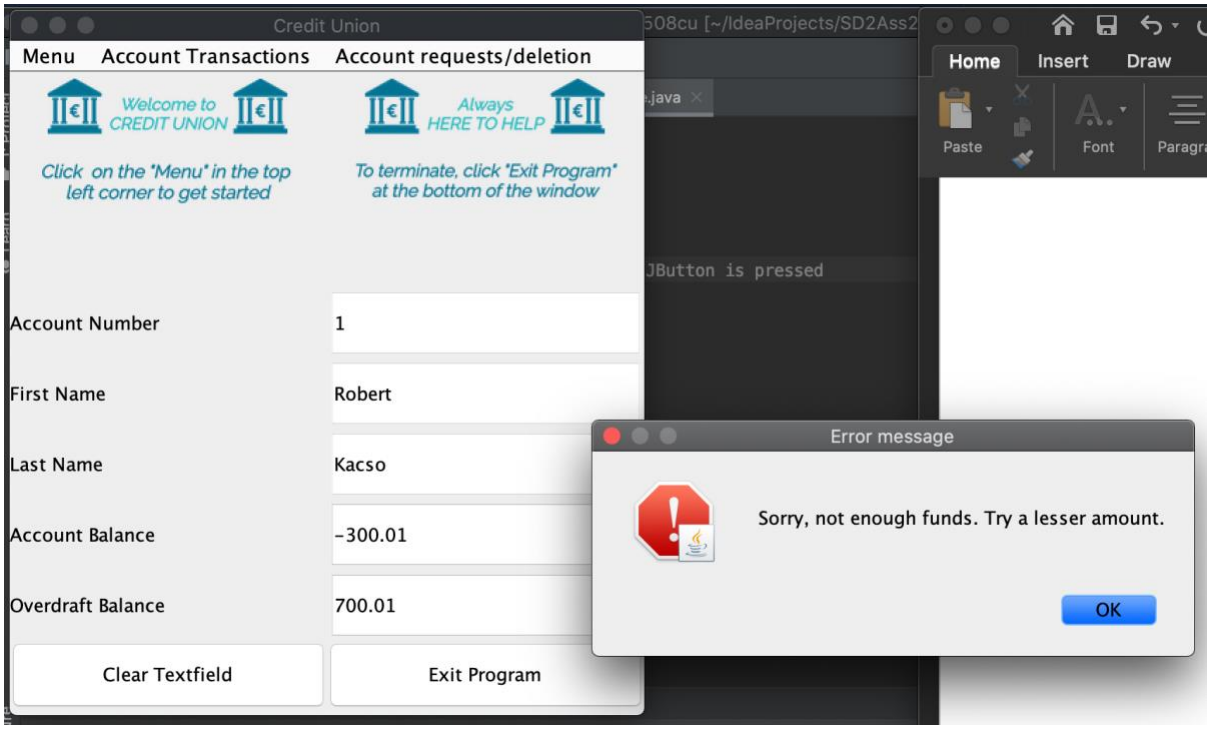

Credit Union
Menu
Account Transactions
Account requests/deletion
Welcome to CREDIT UNION
Always HERE TO HELP
Click on the "Menu" in the top left corner to get started
To terminate, click "Exit Program" at the bottom of the window
Account Number
1
First Name
Robert
Last Name
Kacso
Account Balance
-100.0
Overdraft Balance
700.01
Clear Textfield
Exit Program
JButton is pressed
Input
Enter amount you want to withdraw:
200.01
Cancel
OK
Home
Insert
Draw
Paste
Font

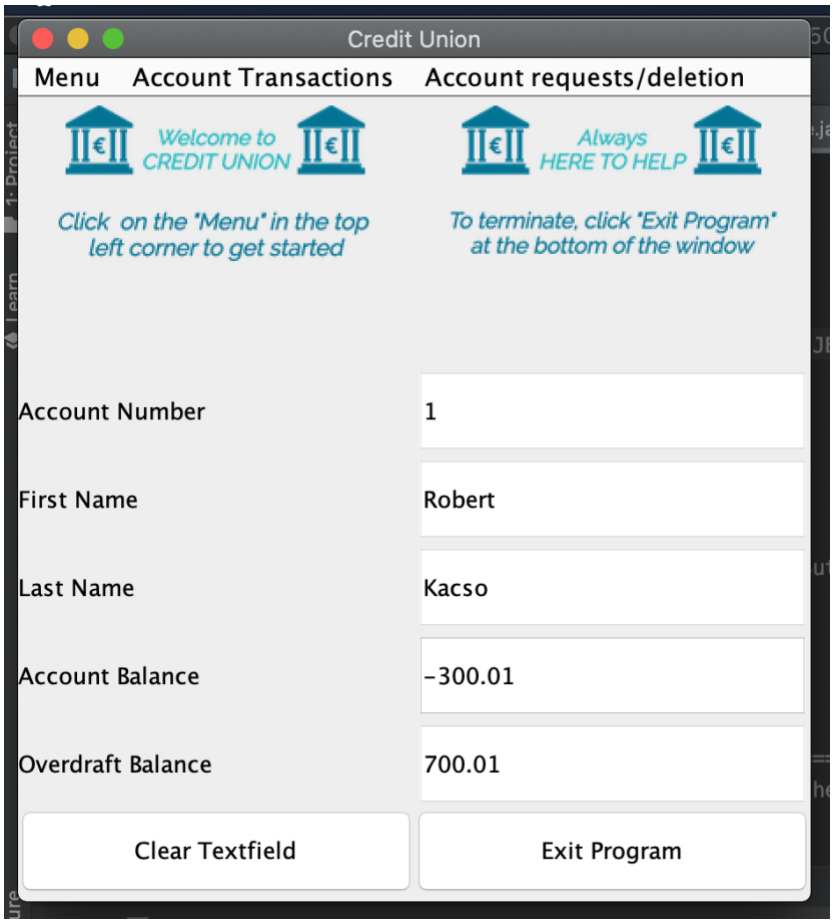

Credit Union
Menu
Account Transactions
Account requests/deletion
Welcome to CREDIT UNION
Always HERE TO HELP
Click on the "Menu" in the top left corner to get started
To terminate, click "Exit Program" at the bottom of the window
Account Number
1
First Name
Robert
Last Name
Kacso
Account Balance
-100.0
Overdraft Balance
700.01
Clear Textfield
Exit Program
JButton is pressed
Warning Message
You accept going into overdraft?
No
Yes
Home
Insert
Paste
Font



Now withdraw more than our left overdraft fund.

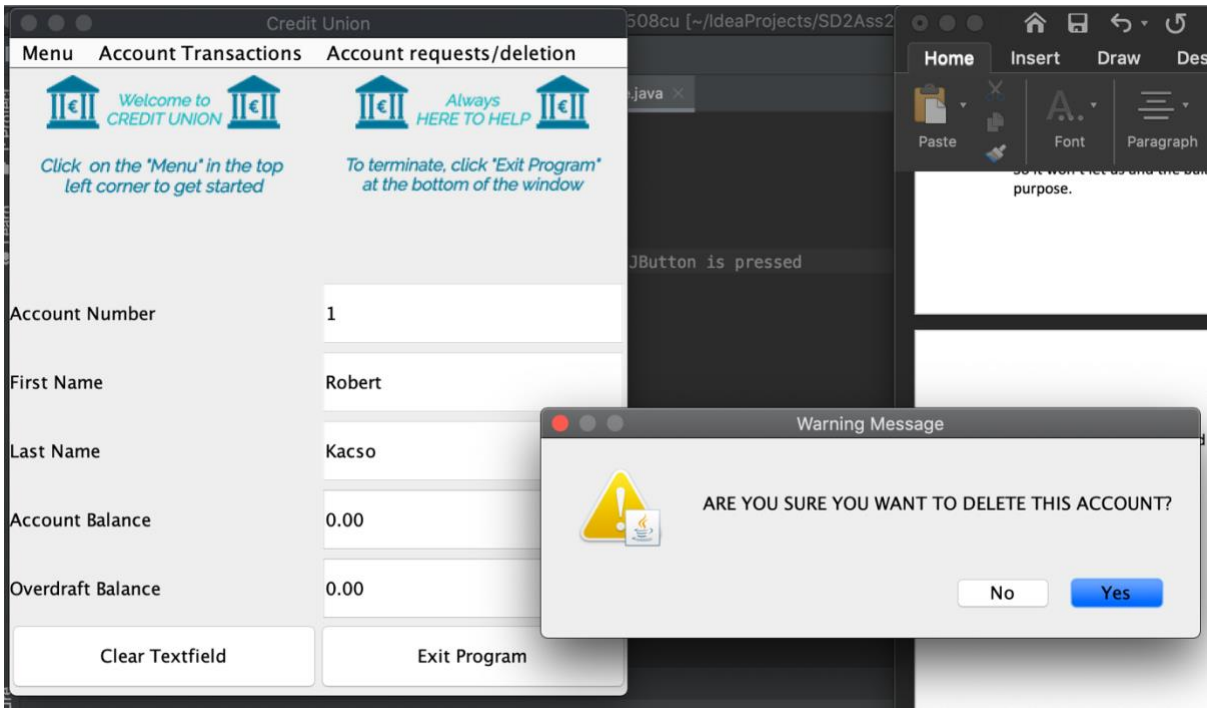


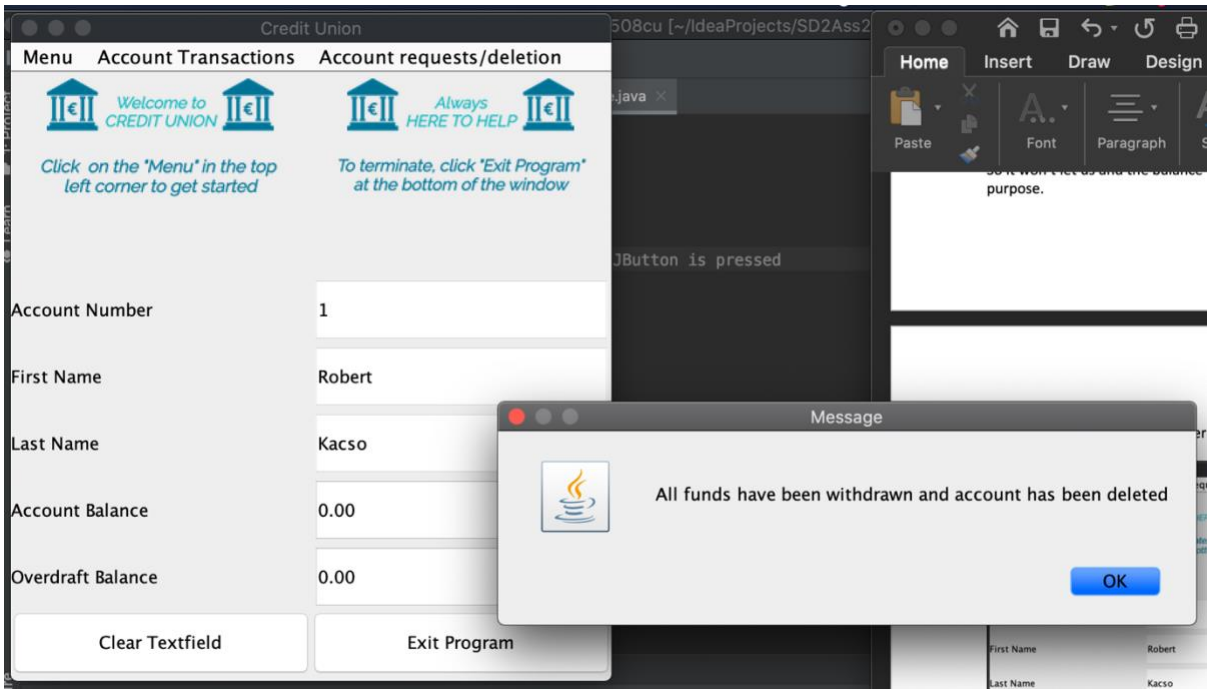


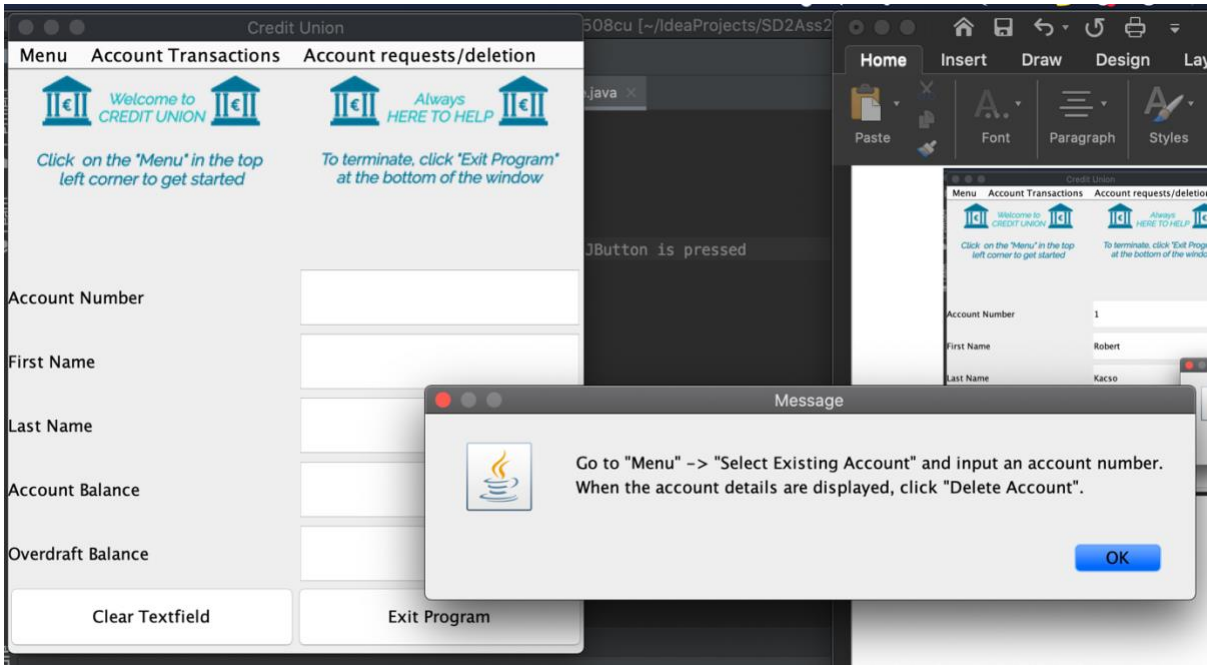


So it won't let us and the balance hasn't been updated either, which is the intended purpose.

Now let’s adjust balance and overdraft to zero and delete the account.

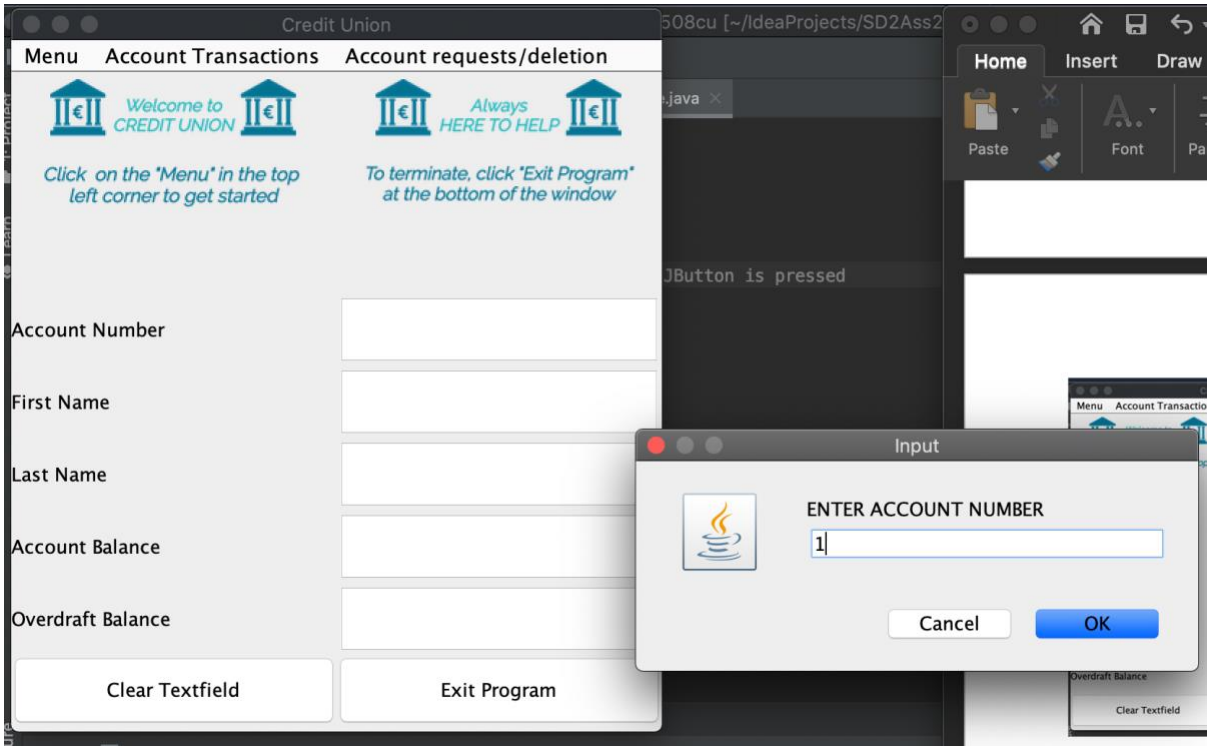


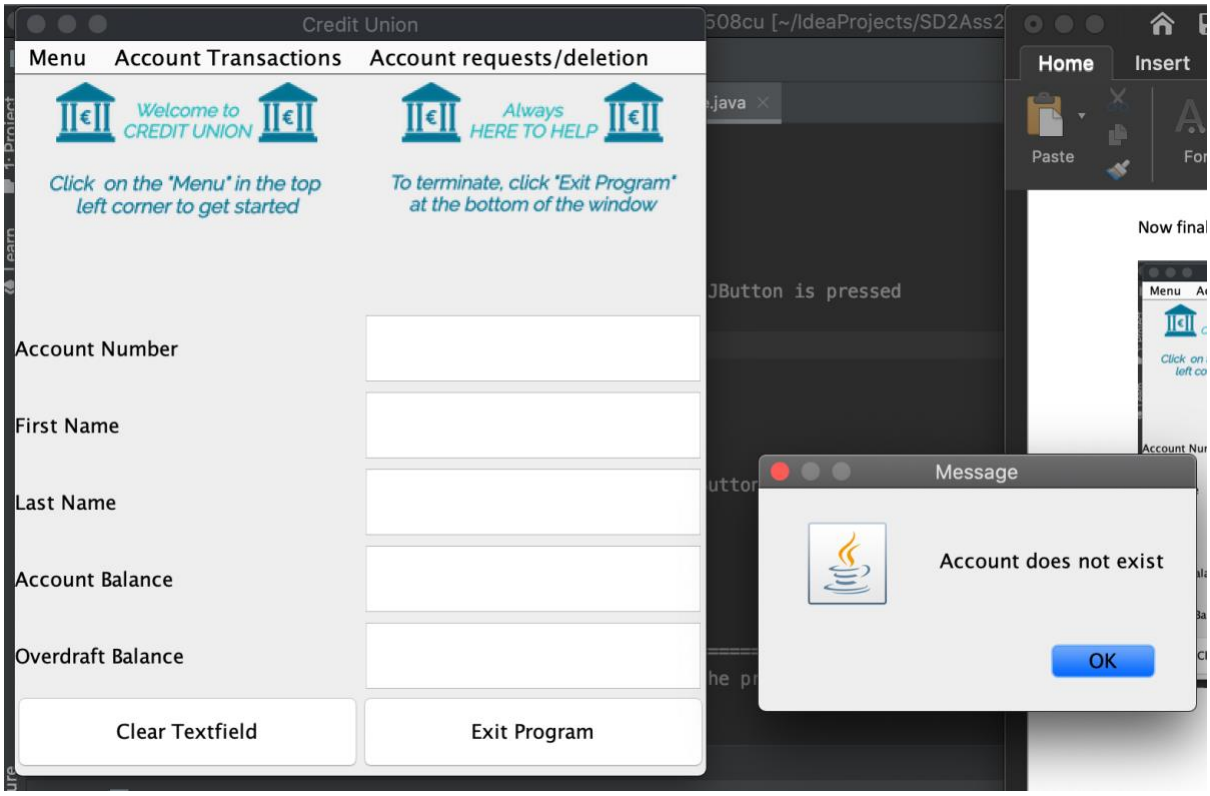




The account has been deleted, but not before a reconfirmation of the user to do so. After the account has been deleted, the text fields have been reset to clear and a pop-up message tells the user that in order to delete another account, select one first.

Now finally let's check if the account has been deleted, by searching for it.



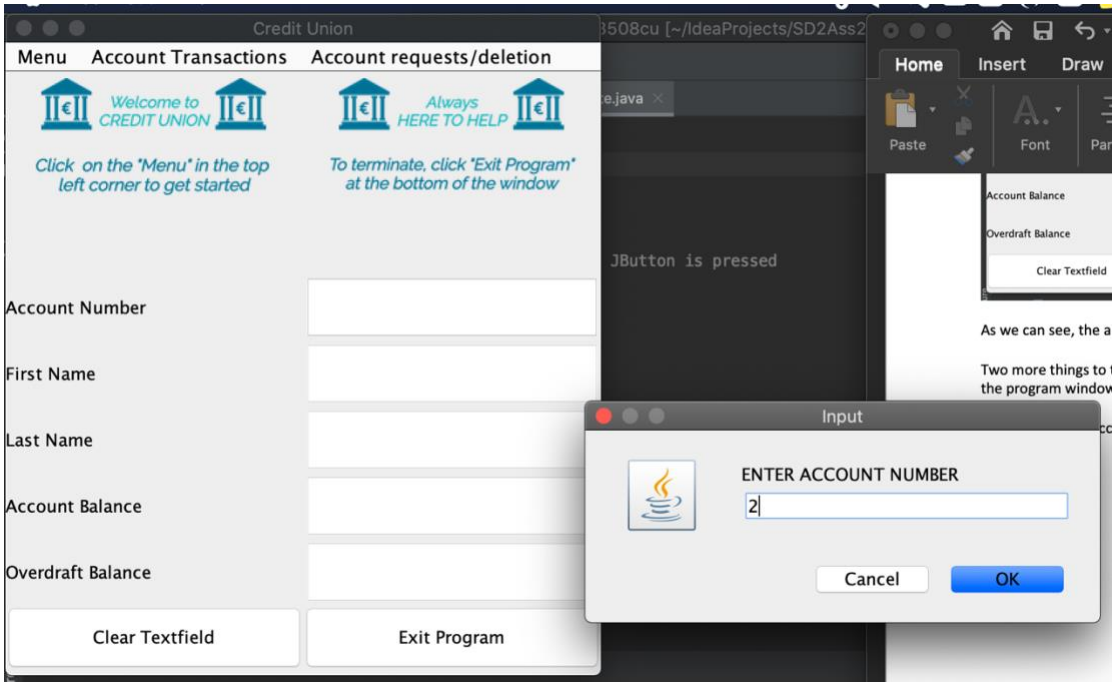


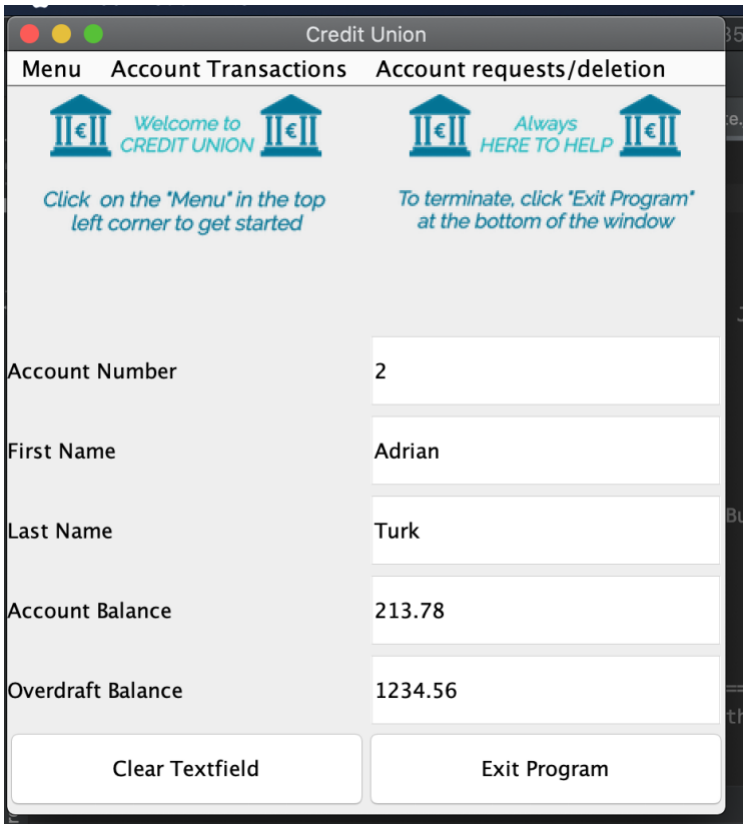
As we can see, the account has been deleted.

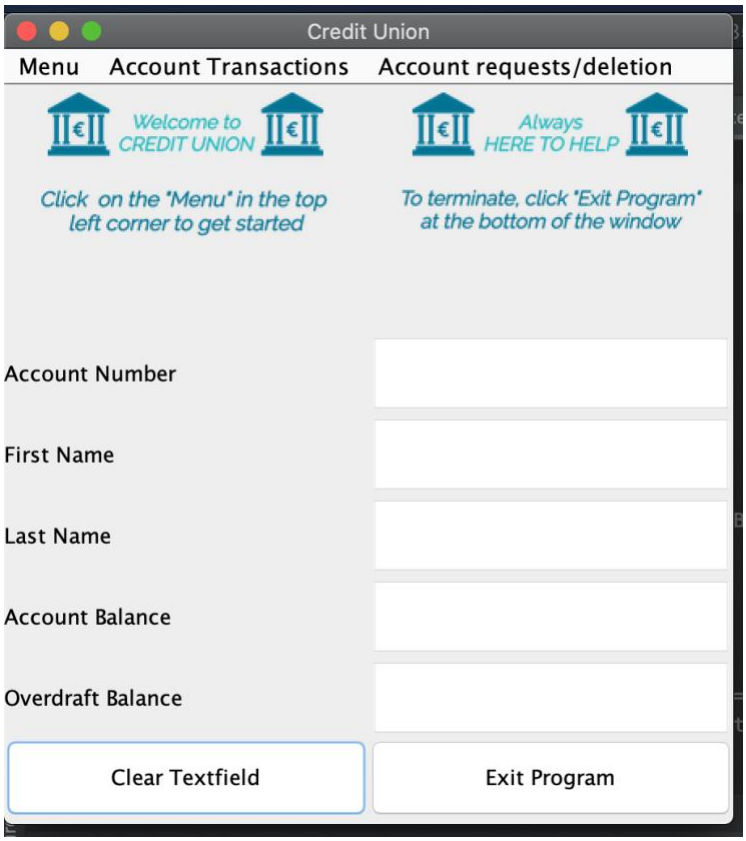
Two more things to test… The “Clear Textfield” and “Exit Program” buttons at the bottom of the program window.

Clear Textfiled

Let’s bring up an account and then clear the text filed.



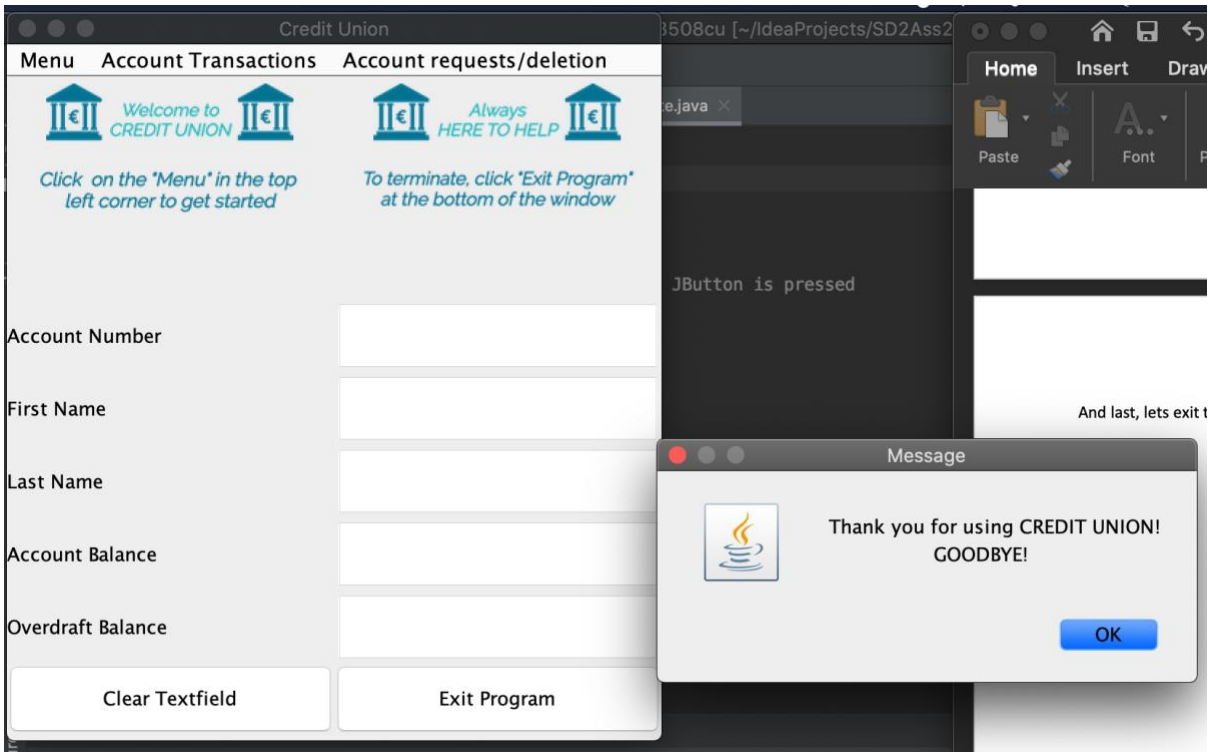




As you can see, the text field button has been pressed and worked as intended.

Exit program

And last, lets exit the program:



It worked and a “Goodbye” message sends us on our way.

The program can be terminated by pressing the “x” at the top corner of the window too, but no message will pop-up in that case.

Conclusion

It was a challenge building this program, with more problems encountered as ever before, but I think I definitely learned a lot, and I feel a sensation of a big achievement, that makes me wonder how good it’ll make me feel when I finish college altogether. I’m grateful for everything I learned during this year and this program and I’ll make sure to practice over the summer holidays, maybe even learn a few other things too.

Thank you for the opportunity.