1. Introduction
2. Use Cases/UP
3. Sequence Diagram/UML
4. Patterns/Code
5. White/Black box testing with use cases examples

**Introduction**

The group consisting of Kevin Clyne, Cody Heard, Christopher Jones, Gary Sims, and Lamar Sims designed and implemented a Banking interface where an account would be created inside of the interface for two different banks. With the creation of those accounts was the type of accounts that were available with the banks: a checking account, saving account, or both. With the finish creation of the Account or Accounts banking operations such as, Deposit and Withdraw can be activated.

**What Everyone Did**

1. Kevin Clyne
   1. Use Cases
   2. Black box
   3. Singleton
2. Lamar Sims
   1. The whole damn project/ UMLs
   2. Template Method
   3. GUI
   4. Testing/debugging
3. Cody
   1. GUI
   2. Testing
4. Gary Sims
   1. Strategy pattern
   2. White box testing
5. Chris Jones
   1. Sequence Diagram
   2. Factory Pattern
   3. Encouraged Cody as he made the GUI

**Flow of Events for Individual Use Cases**

Start-up Case – When the application is run the user will be prompted to create an account in order to access the banking interface.

Account Creation Case – The user will then reach a form that asks for the user’s name, age, SSN, address, password, bank, and the type of account he/she wishes to create(Savings/Checking). The information is then printed and displayed on the console.

Account Balance Input Case – The User will be asked to enter the balance for whichever account is selected. A pop-up message will then display all of the user’s account information.

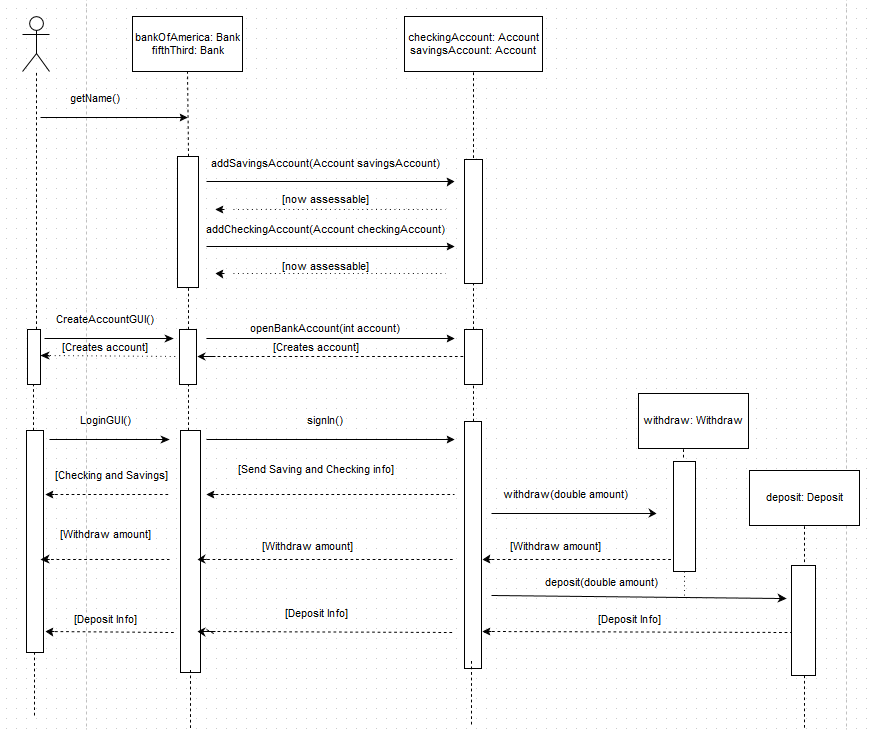
Account Log-in Case – The user will user must then access the banking interface by returning to the start-up screen, and entering their access ID and password. A pop-up will ask the user to verify their password for a 2nd time.

Banking interface Case- Once logged in, the user will be able to view their account balance, account ID, and make withdrawals and deposits.

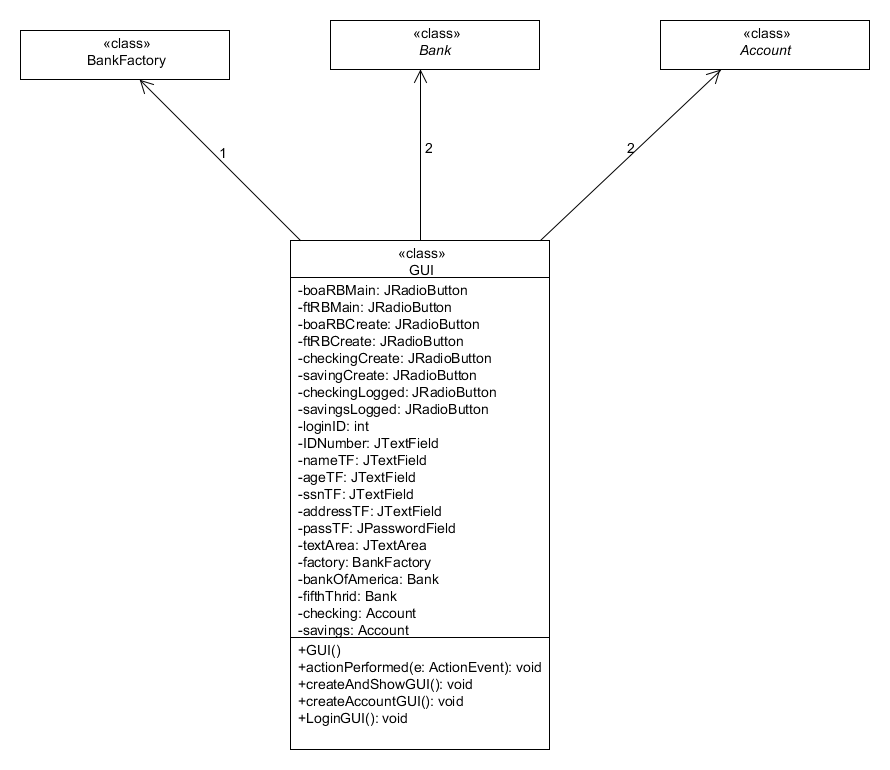
Account Deposit Case – The user can make a deposit to their checking/savings account by clicking on the “Deposit” button, then entering the amount they wish to deposit. They will then see their updated account balance on the banking interface screen.

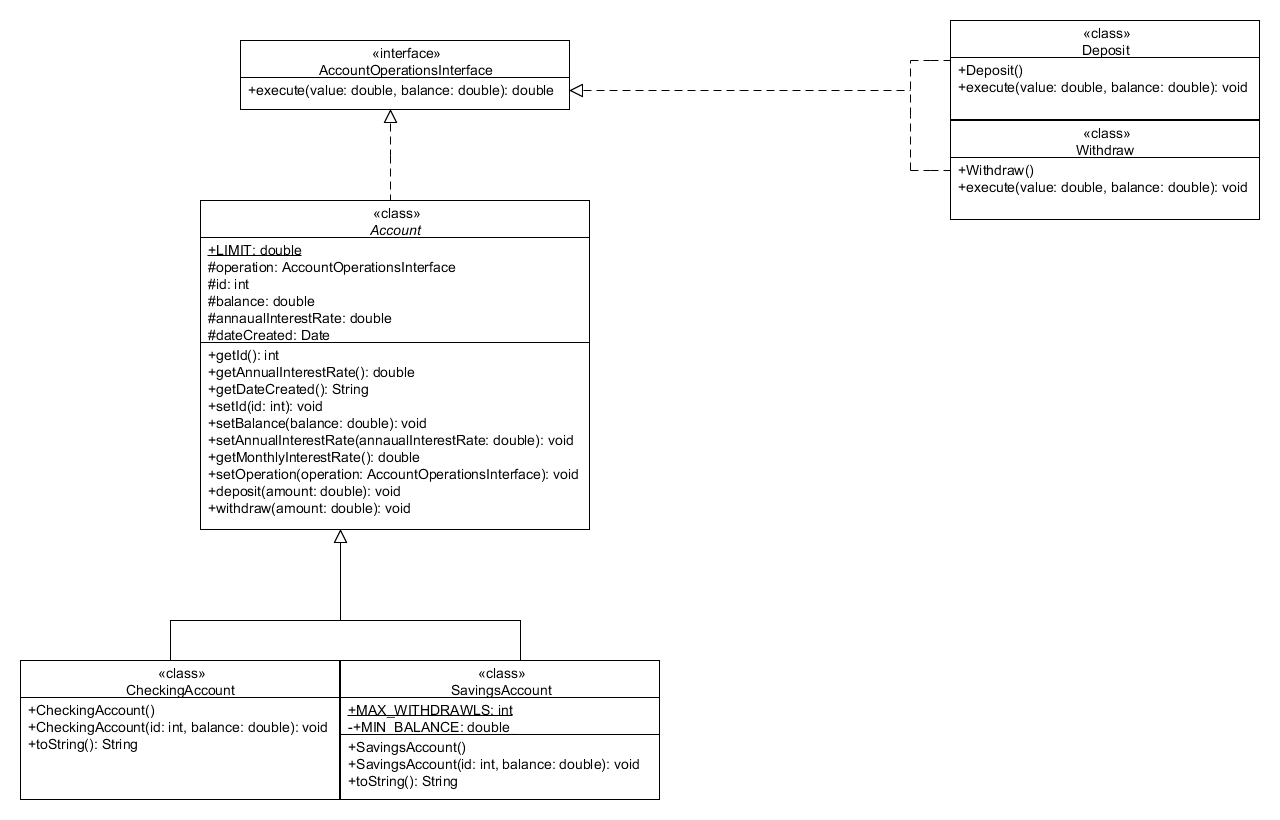
Account withdrawal Case – The user can make a withdrawal to their checking/savings account by clicking on the “withdrawal” button, then entering the amount they wish to withdraw. They will then see their updated account balance on the banking interface screen.

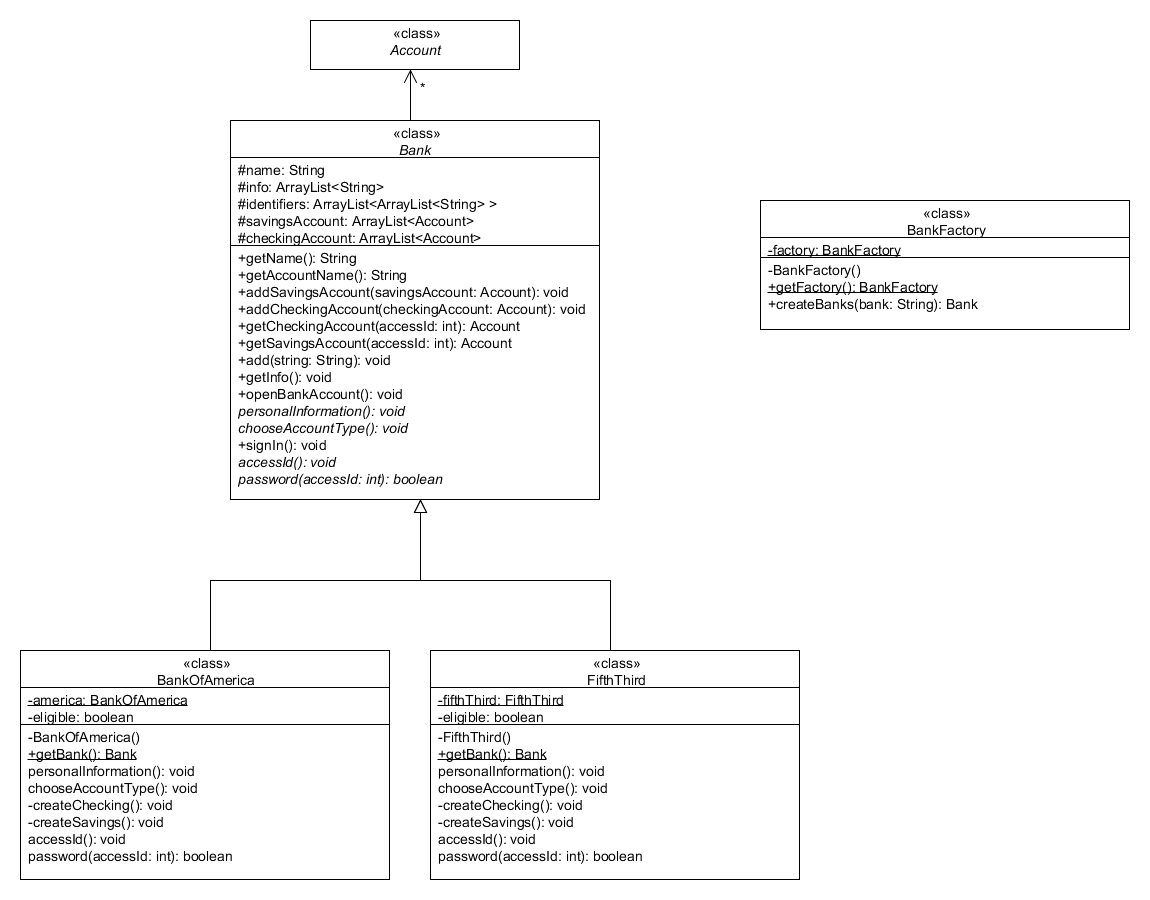
Invalid input Case – Any invalid characters will not be accepted, and the application will notify the user via pop-up message.

**Sequence Diagram** 

**UML Diagrams**







**Patterns and Code**

public class Main {

  public static void main(String[] args) {

GUI banking = new GUI();

banking.createAndShowGUI();

  }

}

import javax.swing.\*;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class GUI extends JPanel implements ActionListener{

private JRadioButton boaRBMain;

private JRadioButton ftRBMain;

private JRadioButton boaRBCreate = new JRadioButton("Bank Of America");

private JRadioButton ftRBCreate = new JRadioButton("Fifth & Third");

private JRadioButton checkingCreate = new JRadioButton("Checking");

private JRadioButton savingCreate = new JRadioButton("Saving");

private JRadioButton checkingLogged = new JRadioButton("Checking");

private JRadioButton savingsLogged = new JRadioButton("Savings");

private int loginID;

private JTextField IDNumber;

private JTextField nameTF = new JTextField(10);

private JTextField ageTF = new JTextField(10);

private JTextField ssnTF = new JTextField(10);

private JTextField addressTF = new JTextField(10);

private JPasswordField passTF = new JPasswordField(15);

private JTextArea textArea = new JTextArea();

private BankFactory factory = BankFactory.getFactory();

private Bank bankOfAmerica = factory.createBanks("Bank Of America");

private Bank fifthThird = factory.createBanks("Fifth Third");

private Account checking;

private Account savings;

public GUI() {

  super(new GridLayout());

  JTabbedPane tabbedPane = new JTabbedPane();

// ImageIcon icon = createImageIcon("images/middle.gif");

  JComponent panel\_1a = new JPanel();

  panel\_1a.setLayout(null);

  panel\_1a.setPreferredSize(new Dimension(300,200));

  tabbedPane.addTab("Sign In", null, panel\_1a,

          "Does nothing");

  JLabel Number = new JLabel("ID Number");

  Number.setBounds(20,50,100,20);

  IDNumber = new JTextField(40);

  IDNumber.setBounds(20, 70, 100, 20);

  JButton Login = new JButton("Login");

  Login.setBounds(158, 60, 80, 40);

  Login.addActionListener(this);

  Login.addActionListener(new ActionListener(){

      public void actionPerformed(ActionEvent arg0) {

     try{

     loginID = Integer.parseInt(IDNumber.getText());

     }catch(Exception ex){

     }

      }

  });

  JButton CreateAccount = new JButton("Create Account");

  CreateAccount.setBounds(90, 150, 130, 50);

  CreateAccount.addActionListener(this);

  boaRBMain = new JRadioButton("Bank Of America");

  boaRBMain.setSelected(true);

  boaRBMain.setBounds(10, 10, 130, 20);

  ftRBMain = new JRadioButton("Fifth & Third");

  ftRBMain.setActionCommand("Fifth & Third");

  ftRBMain.setBounds(150, 10, 150, 20);

  ButtonGroup group = new ButtonGroup();

  group.add(boaRBMain);

  group.add(ftRBMain);

  panel\_1a.add(IDNumber);

  panel\_1a.add(Login);

  panel\_1a.add(CreateAccount);

  panel\_1a.add(Number);

  panel\_1a.add(boaRBMain);

  panel\_1a.add(ftRBMain);

  //Add the tabbed pane to this panel.

  add(tabbedPane);

}

public void actionPerformed(ActionEvent e) {

    if(e.getActionCommand() == "Login") {

        if(boaRBMain.isSelected() && bankOfAmerica.signIn(loginID)){

            LoginGUI();

        }

        if(ftRBMain.isSelected() && fifthThird.signIn(loginID)){

            LoginGUI();

        }

    }else if(e.getActionCommand() == "Create Account"){

   nameTF.setText("");

        ageTF.setText("");

        ssnTF.setText("");

        addressTF.setText("");

        passTF.setText("");

        CreateAccountGUI();

    }

 }

public static void createAndShowGUI() {

  JFrame frame = new JFrame("");

  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

  frame.add(new GUI(), BorderLayout.CENTER);

  frame.pack();

  frame.setResizable(false);

  frame.setVisible(true);

}

public void CreateAccountGUI(){

  JPanel p = new JPanel(new GridLayout(8,2));

  JLabel[] labels = { new JLabel("Name:") , new JLabel("Age"), new JLabel("SSN"), new JLabel("Address"), new JLabel("Password")};

  JButton confirm = new JButton("Confirm");

  checkingCreate.setActionCommand("Checking");

  checkingCreate.setSelected(true);

  p.add(checkingCreate);

  savingCreate.setActionCommand("Savings");

  savingCreate.setSelected(false);

  p.add(savingCreate);

  p.add(labels[0]);

  p.add(nameTF);

  nameTF.setText("First Last");

  p.add(labels[1]);

  p.add(ageTF);

  ageTF.setText("###");

  p.add(labels[2]);

  p.add(ssnTF);

  ssnTF.setText("###-##-####");

  p.add(labels[3]);

  p.add(addressTF);

  addressTF.setText("Street address only");

  p.add(labels[4]);

  p.add(passTF);

  ButtonGroup group = new ButtonGroup();

  group.add(boaRBCreate);

  group.add(ftRBCreate);

  boaRBCreate.setActionCommand("Bank Of America");

  boaRBCreate.setSelected(true);

  p.add(boaRBCreate);

  ftRBCreate.setActionCommand("Fifth & Third");

  p.add(ftRBCreate);

  p.add(new JLabel());

  p.add(confirm);

  confirm.addActionListener(new ActionListener(){

      public void actionPerformed(ActionEvent arg0) {

     boolean testPassed = true;

     if(boaRBCreate.isSelected()){

     if(nameTF.getText().isEmpty() || ageTF.getText().isEmpty() || ssnTF.getText().isEmpty() || addressTF.getText().isEmpty() || passTF.getPassword().length < 7){

     testPassed = false;

     }

     if(testPassed){

     try{

     Integer.parseInt(ageTF.getText());

     } catch (Exception ex){

     JOptionPane.showMessageDialog(null, "The age was not valid");

     testPassed = false;

     }

     }

          if(testPassed){

         bankOfAmerica.add(nameTF.getText());

         bankOfAmerica.add(ageTF.getText());

         bankOfAmerica.add(ssnTF.getText());

         bankOfAmerica.add(addressTF.getText());

         bankOfAmerica.add(String.valueOf(passTF.getPassword()));

         int account = 0;

         if(checkingCreate.isSelected() && savingCreate.isSelected()){

                  account = 2;

              }

              else if(checkingCreate.isSelected()){

                  account = 0;

              }

              else if(savingCreate.isSelected()){

                  account = 1;

              }

            bankOfAmerica.openBankAccount(account);

          }

          } else if(ftRBCreate.isSelected()){

         if(nameTF.getText().isEmpty() || ageTF.getText().isEmpty() || ssnTF.getText().isEmpty() || addressTF.getText().isEmpty() || passTF.getPassword().length < 7){

     testPassed = false;

     }

     if(testPassed){

     try{

     Integer.parseInt(ageTF.getText());

     } catch (Exception ex){

     JOptionPane.showMessageDialog(null, "The age was not valid");

     testPassed = false;

     }

     }

       if(testPassed){

     int account = 0;

     fifthThird.add(nameTF.getText());

      fifthThird.add(ageTF.getText());

      fifthThird.add(ssnTF.getText());

      fifthThird.add(addressTF.getText());

      fifthThird.add(String.valueOf(passTF.getPassword()));

      if(checkingCreate.isSelected() && savingCreate.isSelected()){

                  account = 2;

              }

              else if(checkingCreate.isSelected()){

                  account = 0;

              }

              else if(savingCreate.isSelected()){

                  account = 1;

              }

             fifthThird.openBankAccount(account);

              }

       }

     testPassed = true;

          }

  });

  JFrame frame = new JFrame("Create Account");

  p.setOpaque(true);

  frame.setContentPane(p);

  frame.pack();

  frame.setVisible(true);

}

public void LoginGUI(){

  JPanel p = new JPanel();

  if(boaRBMain.isSelected()){

     textArea = new JTextArea("Welcome \n" + bankOfAmerica.getAccountName(loginID) ,11,20);

  }

  if(ftRBMain.isSelected()){

     textArea = new JTextArea("Welcome \n" + fifthThird.getAccountName(loginID) ,11,20);

  }

  textArea.setEditable(false);

  p.add(textArea);

  JPanel p2 = new JPanel(new GridLayout(2,2,0,30));

  JButton deposit = new JButton("Deposit");

  JButton withdraw = new JButton("Withdraw");

  ButtonGroup group = new ButtonGroup();

  group.add(checkingLogged);

  group.add(savingsLogged);

  checkingLogged.setSelected(true);

  p2.add(checkingLogged);

  p2.add(savingsLogged);

  p2.add(deposit);

  p2.add(withdraw);

  if(boaRBMain.isSelected()){

      checking = bankOfAmerica.getCheckingAccount(loginID);

      savings = bankOfAmerica.getSavingsAccount(loginID);

      if(checking != null && savings != null){

          textArea.insert(checking.toString(), 0);

          textArea.insert(savings.toString(), 0);

      } else if(checking != null){

              textArea.insert(checking.toString(), 0);

      } else if(savings != null){

         textArea.insert(savings.toString(), 0);

      }

  } else if(ftRBMain.isSelected()){

      checking = fifthThird.getCheckingAccount(loginID);

      savings = fifthThird.getSavingsAccount(loginID);

      if(checking != null && savings != null){

          textArea.insert(checking.toString(), 0);

          textArea.insert(savings.toString(), 0);

      } else if(checking != null){

              textArea.insert(checking.toString(), 0);

      } else  textArea.insert(savings.toString(), 0);

  }

  deposit.addActionListener(new ActionListener(){

      public void actionPerformed(ActionEvent e) {

     Deposit depo = new Deposit();

          if(checkingLogged.isSelected()){

              if(e.getActionCommand() == "Deposit" && checking != null){

                  checking.setOperation(depo);

                  try{

                 checking.deposit(Double.parseDouble(JOptionPane.showInputDialog(null, "Enter the amount you want to deposit into checking:")));

                  } catch (Exception ex){

                 JOptionPane.showMessageDialog(null, "Please enter a valid monetary amount.");

                  }

                  textArea.insert(checking.toString(), 0);

              }

          } else if(savingsLogged.isSelected()){

                   if(e.getActionCommand() == "Deposit" && savings != null){

                       savings.setOperation(depo);

                       try{

                      savings.withdraw(Double.parseDouble(JOptionPane.showInputDialog(null, "Enter the amount you want to deposit into savings:")));

                       } catch (Exception ex){

                      JOptionPane.showMessageDialog(null, "Please enter a valid monetary amount.");

                       }

                       textArea.insert(savings.toString(), 0);

             }

         }

      }

   });

  withdraw.addActionListener(new ActionListener(){

      public void actionPerformed(ActionEvent e) {

     Withdraw withdraw = new Withdraw();

          if(checkingLogged.isSelected()){

              if(e.getActionCommand() == "Withdraw" && checking != null){

                  checking.setOperation(withdraw);

                  try{

                 checking.withdraw(Double.parseDouble(JOptionPane.showInputDialog(null, "Enter the amount you want to withdraw from checking:")));

                  } catch (Exception ex){

                 JOptionPane.showMessageDialog(null, "Please enter a valid monetary amount.");

              }

                  textArea.insert(checking.toString(), 0);

          }

          }

              if(savingsLogged.isSelected()){

                if(e.getActionCommand() == "Withdraw" && savings != null){

                    savings.setOperation(withdraw);

                    try{

                    savings.withdraw(Double.parseDouble(JOptionPane.showInputDialog(null, "Enter the amount you want to withdraw from savings:")));

                     } catch (Exception ex){

                    JOptionPane.showMessageDialog(null, "Please enter a valid monetary amount.");

                     }

                    textArea.insert(savings.toString(), 0);

              }

          }

      }

      });

  //Create and set up the window.

  JFrame frame = new JFrame("Account");

  //Set up the content pane.

  p.setOpaque(true);  //content panes must be opaque

  frame.add(p, BorderLayout.WEST);

  frame.add(p2, BorderLayout.EAST);

  //Display the window.

  frame.pack();

  frame.setVisible(true);

  frame.setSize(430,210);

  frame.setResizable(false);

}

}

import java.util.ArrayList;

import javax.swing.JOptionPane;

import javax.swing.JPasswordField;

/\*\*Singleton Pattern

\* Class to represent Fifth Third bank \*/

public class FifthThird extends Bank{

//Declare and initialize variables

private static FifthThird fifthThird;

private boolean eligible = false;

/\*\* no-arg constructor \*/

private FifthThird() {

    name = "Fifth Third";

    identifiers = new ArrayList<ArrayList<String>>();

    savingsAccount = new ArrayList<Account>();

    checkingAccount = new ArrayList<Account>();

}

/\*\* entry point to get the FifthThird object \*/

public synchronized static Bank getBank(){

    if (fifthThird == null){

        fifthThird = new FifthThird();

    }

    return fifthThird;

}

/\*\* evaluate personal information to determine if eligible to open a bank account\*/

void personalInformation() {

    //determine if eligible to open a bank account

      if (Integer.parseInt(info.get(1)) < 18){

          JOptionPane.showMessageDialog(null, "Sorry " + name + " but you are too young to open a bank account at " + this.name);

          return;

      }

      eligible = true; //eligible to open a bank account

      //create an access number for the bank account

      boolean valid = false;

      int accessNum = 0;

      do{

          accessNum = (int)(1000 + Math.random() \* 8999);

          valid = true;

          //make sure accessNum has not already been chosen

          for (ArrayList<String> list : identifiers){

              if (Integer.parseInt(list.get(5)) == accessNum){

                  valid = false;

                  break;

              }

          }

      }while(!valid);

      //add the information into the ArrayList for temporary storage

      info.add(String.valueOf(accessNum));

}

/\*\* choose and create the account type \*/

void chooseAccountType(int account) {

      if (!eligible) return; //if eligible is false exit method

      //create the account type

      switch(account){

      case 0: createChecking();

          break;

      case 1: createSavings();

          break;

      case 2: createChecking();

              createSavings();

      }

   //display the account holder's information

      String output = new String();

      int i = 0;

   for(String str: info){

       switch(i){

       case 0: output += "Name: " + str + "\n";

           break;

       case 1: output += "Age: " + str + "\n";

           break;

       case 2: output += "SSN: " + str + "\n";

           break;

       case 3: output += "Address: " + str + "\n";

           break;

       case 4: output += "Password: " + str + "\n";

           break;

       case 5: output += "Access Number: " + str + "\n";

       }

       i++;

   }

   JOptionPane.showMessageDialog(null, "Personal and Login Information\n" + output);

      //copy the info ArrayList into temp to be stored in the data field identifiers

      ArrayList<String> temp = new ArrayList<String>();

      for (String str: info){

          temp.add(str);

      }

      //add the information to identifiers

      identifiers.add(temp);

      info.clear(); //remove all the elements in info

      eligible = false;

}

/\*\* create a checking account \*/

private void createChecking(){

      //prompt for a balance to put in the account

   double balance = 0;

   boolean isValid = false;

   do{

   try{

   balance = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter a balance for the checking account: "));

   isValid = true;

   } catch (Exception e){

   JOptionPane.showMessageDialog(null, "Please enter in a valid amount");

   }

   } while(!isValid);

      //make sure balance is valid

      while (balance < 0){

          balance = Double.parseDouble(JOptionPane.showInputDialog(null, "The balance cannot be negative. Enter a balance for the checking account: "));

      }

      //create a new checking account, using the access number as the id, and add it to the checkingAccount ArrayList

      checkingAccount.add(new CheckingAccount(Integer.parseInt(info.get(5)), balance));

}

/\*\* create a savings account \*/

private void createSavings(){

      //prompt for a balance to put in the account

   double balance = 0;

   boolean isValid = false;

   do{

   try{

   balance = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter a balance for the savings account: "));

   isValid = true;

   } catch (Exception e){

   JOptionPane.showMessageDialog(null, "Please enter in a valid amount");

   }

   } while(!isValid);

      //make sure balance is valid

      while (balance < SavingsAccount.MIN\_BALANCE){

          balance = Double.parseDouble(JOptionPane.showInputDialog(null, "The balance must be equal to or greater than $" + SavingsAccount.MIN\_BALANCE + ". Enter a balance for the savings account "));

      }

      //create a new savings account, using the access number as the id, and add it to the checkingAccount ArrayList

      savingsAccount.add(new SavingsAccount(Integer.parseInt(info.get(5)), balance));

}

/\*\* evaluate access id for bank account \*/

boolean accessId(int accessId) {

      //evaluate the checking accounts for the specified access id, to see if access id is valid

      for (Account checking: checkingAccount){

          if (checking.getId() == accessId){

              return true;

          }

      }

      //evaluate the savings accounts for the specified access id, to see if access id is valid

      for (Account savings: savingsAccount){

          if (savings.getId() == accessId){

              return true;

          }

      }

      JOptionPane.showMessageDialog(null, "Specified access number is not valid!");

      return false;

}

/\*\* evaluate password for bank account \*/

boolean password(int accessId) {

      int i = 0; //sign in attempts

      String password = new String();

      JPasswordField pf = new JPasswordField();

      do{

          //prompt the user to provide their password for their bank account

          int option;

          if (i != 0){

              option = JOptionPane.showConfirmDialog(null, pf, "Invalid password specified for this account. Enter you password again: ",

                                                   JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

          }else{

              option = JOptionPane.showConfirmDialog(null, pf, "Enter you password: ", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

          }

          if (option == JOptionPane.OK\_OPTION){

              password = String.valueOf(pf.getPassword());

              //evaluate the identifiers ArrayList for the specified password, to see if password is valid

              //and return the checking account

              for (ArrayList<String> str: identifiers){

                  //check if the password specified is a valid password for an identifier and

                  //check to make sure the access id and password match up for the identifier

                  if (str.get(4).equals(password) && Integer.parseInt(str.get(5)) == accessId){

                      return true;

                  }

              }

          }else if (option == JOptionPane.CANCEL\_OPTION){

              return false;

          }

          i++;

      }while((i <= 4));

      JOptionPane.showMessageDialog(null, "You have reached your number of password login attempts.");

      return false;

}

}

/\*\* Representation of a SavingsAccount \*/

public class SavingsAccount extends Account{

public final static int MAX\_WITHDRAWLS = 6; //max withdrawls allowed per month

public final static double MIN\_BALANCE = 5.00; //minimum balance allowed before fees occur

/\*\* no-arg constructor \*/

public SavingsAccount(){

   id = 0;

   balance = 0;

   annualInterestRate = 0;

}

/\*\* construct SavingsAccount with specified id and balance \*/

public SavingsAccount(int id, double balance){

   this.id = id;

   this.balance = balance;

}

/\*\* string representation of a SavingsAccount object \*/

public String toString(){

    return "Savings Account:\n" +

            "Id: " + id +

            "\nBalance: " + balance + "\n";

}

}

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\* Strategy Pattern

\* A withdraw operation for an Account \*/

public class Withdraw implements AccountOperationsInterface{

  /\*\* no-arg constructor \*/

public Withdraw(){

}

/\*\* perform the withdraw operation \*/

  public double execute(double value, double balance){

   return balance - value;

  }

}

/\*\*Strategy Pattern

\* A deposit operation for an Account \*/

public class Deposit implements AccountOperationsInterface{

  /\*\* no-arg constructor \*/

public Deposit(){

}

/\*\* perform the deposit operation \*/

  public double execute(double value, double balance){

   return value + balance;

  }

}

/\*\* Representation of a CheckingAccount \*/

public class CheckingAccount extends Account{

public static double LIMIT = -30.00; //overdraft limit

/\*\* no-arg constructor \*/

public CheckingAccount(){

   id = 0;

   balance = 0;

   annualInterestRate = 0;

}

/\*\* construct CheckingAccount with specified id and balance \*/

public CheckingAccount(int id, double balance){

   this.id = id;

   this.balance = balance;

}

/\*\* string representation of a CheckingAccount object \*/

public String toString(){

    return "Checking Account:\n" +

            "Id: " + id +

            "\nBalance: " + balance + "\n";

}

}

import java.util.ArrayList;

import javax.swing.JPasswordField;

import javax.swing.JOptionPane;

/\*\*Singleton Pattern

\* Class to represent Bank of America \*/

public class BankOfAmerica extends Bank {

//Declare and initialize variables

private static BankOfAmerica america;

private boolean eligible = false;

/\*\* no-arg constructor \*/

private BankOfAmerica() {

    name = "Bank Of America";

    identifiers = new ArrayList<ArrayList<String>>();

    savingsAccount = new ArrayList<Account>();

    checkingAccount = new ArrayList<Account>();

}

/\*\* entry point to get the BankOfAmerica object \*/

public static synchronized Bank getBank(){

    if (america == null){

        america = new BankOfAmerica();

    }

    return america;

}

/\*\* evaluate personal information to determine if eligible to open a bank account\*/

void personalInformation() {

    //determine if eligible to open a bank account

      if (Integer.parseInt(info.get(1)) < 18){

          JOptionPane.showMessageDialog(null, "Sorry " + name + " but you are too young to open a bank account at " + this.name);

          return;

      }

      eligible = true; //eligible to open a bank account

      //create an access number for the bank account

      boolean valid = false;

      int accessNum = 0;

      do{

          accessNum = (int)(1000 + Math.random() \* 8999);

          valid = true;

          //make sure accessNum has not already been chosen

          for (ArrayList<String> list : identifiers){

              if (Integer.parseInt(list.get(5)) == accessNum){

                  valid = false;

                  break;

              }

          }

      }while(!valid);

      //add the information into the ArrayList for temporary storage

      info.add(String.valueOf(accessNum));

}

/\*\* choose and create the account type \*/

void chooseAccountType(int account) {

      if (!eligible) return; //if eligible is false exit method

      //create the account type

      switch(account){

      case 0: createChecking();

          break;

      case 1: createSavings();

          break;

      case 2: createChecking();

              createSavings();

      }

   //display the account holder's information

      String output = new String();

      int i = 0;

   for(String str: info){

       switch(i){

       case 0: output += "Name: " + str + "\n";

           break;

       case 1: output += "Age: " + str + "\n";

           break;

       case 2: output += "SSN: " + str + "\n";

           break;

       case 3: output += "Address: " + str + "\n";

           break;

       case 4: output += "Password: " + str + "\n";

           break;

       case 5: output += "Access Number: " + str + "\n";

       }

       i++;

   }

   JOptionPane.showMessageDialog(null, "Personal and Login Information\n" + output);

      //copy the info ArrayList into temp to be stored in the data field identifiers

      ArrayList<String> temp = new ArrayList<String>();

      for (String str: info){

          temp.add(str);

      }

      //add the information to identifiers

      identifiers.add(temp);

      info.clear(); //remove all the elements in info

      eligible = false;

}

/\*\* create a checking account \*/

private void createChecking(){

      //prompt for a balance to put in the account

   double balance = 0;

   boolean isValid = false;

   do{

   try{

   balance = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter a balance for the checking account: "));

   isValid = true;

   } catch (Exception e){

   JOptionPane.showMessageDialog(null, "Please enter in a valid amount");

   }

   } while(!isValid);

      //make sure balance is valid

      while (balance < 0){

          balance = Double.parseDouble(JOptionPane.showInputDialog(null, "The balance cannot be negative. Enter a balance for the checking account: "));

      }

      //create a new checking account, using the access number as the id, and add it to the checkingAccount ArrayList

      checkingAccount.add(new CheckingAccount(Integer.parseInt(info.get(5)), balance));

}

/\*\* create a savings account \*/

private void createSavings(){

      //prompt for a balance to put in the account

   double balance = 0;

   boolean isValid = false;

   do{

   try{

   balance = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter a balance for the savings account: "));

   isValid = true;

   } catch (Exception e){

   JOptionPane.showMessageDialog(null, "Please enter in a valid amount");

   }

   } while(!isValid);

      //make sure balance is valid

      while (balance < SavingsAccount.MIN\_BALANCE){

          balance = Double.parseDouble(JOptionPane.showInputDialog(null, "The balance must be equal to or greater than $" + SavingsAccount.MIN\_BALANCE + ". Enter a balance for the savings account "));

      }

      //create a new savings account, using the access number as the id, and add it to the checkingAccount ArrayList

      savingsAccount.add(new SavingsAccount(Integer.parseInt(info.get(5)), balance));

}

/\*\* evaluate access id for bank account \*/

boolean accessId(int accessId) {

      //evaluate the checking accounts for the specified access id, to see if access id is valid

      for (Account checking: checkingAccount){

          if (checking.getId() == accessId){

              return true;

          }

      }

      //evaluate the savings accounts for the specified access id, to see if access id is valid

      for (Account savings: savingsAccount){

          if (savings.getId() == accessId){

              return true;

          }

      }

      JOptionPane.showMessageDialog(null, "Specified access number is not valid!");

      return false;

}

/\*\* evaluate password for bank account \*/

boolean password(int accessId) {

      int i = 0; //sign in attempts

      String password = new String();

      JPasswordField pf = new JPasswordField();

      do{

          //prompt the user to provide their password for their bank account

          int option;

          if (i != 0){

              option = JOptionPane.showConfirmDialog(null, pf, "Invalid password specified for this account. Enter you password again: ",

                                                   JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

          }else{

              option = JOptionPane.showConfirmDialog(null, pf, "Enter you password: ", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);

          }

          if (option == JOptionPane.OK\_OPTION){

              password = String.valueOf(pf.getPassword());

              //evaluate the identifiers ArrayList for the specified password, to see if password is valid

              //and return the checking account

              for (ArrayList<String> str: identifiers){

                  //check if the password specified is a valid password for an identifier and

                  //check to make sure the access id and password match up for the identifier

                  if (str.get(4).equals(password) && Integer.parseInt(str.get(5)) == accessId){

                      return true;

                  }

              }

          }else if (option == JOptionPane.CANCEL\_OPTION){

              return false;

          }

          i++;

      }while((i <= 4));

      JOptionPane.showMessageDialog(null, "You have reached your number of password login attempts.");

      return false;

}

}

/\*\*Factory/Singleton Pattern

\* Factory class for create bank objects \*/

public class BankFactory {

//Declare and initialize variables

private static BankFactory factory;

/\*\* no-arg constructor \*/

private BankFactory(){

}

/\*\* get the factory \*/

public static synchronized BankFactory getFactory(){

    if (factory == null){

        factory = new BankFactory();

    }

    return factory;

}

/\*\* factory method to create banks \*/

public Bank createBanks(String bank){

    if (bank == "Bank Of America"){

        return BankOfAmerica.getBank();

    }else if (bank == "Fifth Third"){

        return FifthThird.getBank();

    }

    return null;

}

}

import java.util.ArrayList;

/\*\* Representation for a bank \*/

public abstract class Bank {

//Declare and initialize variables

protected String name = "Bank";

protected ArrayList<String> info = new ArrayList<String>();

protected ArrayList<ArrayList<String>> identifiers;

protected ArrayList<Account> savingsAccount;

protected ArrayList<Account> checkingAccount;

/\*\* return this Bank's name \*/

public String getName(){

    return name;

}

/\*\* add a savings account to this Bank \*/

public void addSavingsAccount(Account savingsAccount){

    this.savingsAccount.add(savingsAccount);

}

/\*\* add a checking account to this Bank \*/

public void addCheckingAccount(Account checkingAccount){

    this.checkingAccount.add(checkingAccount);

}

/\*\* return a checking account in this Bank \*/

public Account getCheckingAccount(int accessId){

    for (Account check: checkingAccount){

        if (check.getId() == accessId){

            return check;

        }

    }

    return null;

}

/\*\* return a savings account in this Bank \*/

public Account getSavingsAccount(int accessId){

    for (Account sav: savingsAccount){

        if (sav.getId() == accessId){

            return sav;

        }

    }

    return null;

}

/\*\* method to add an element to info \*/

public void add(String string){

      info.add(string);

}

/\*\* get return the elements stored in info \*/

public void getInfo(){

      for (String str: info){

          System.out.println(str);

      }

}

/\*\*Template method pattern

\* opens a bank account \*/

public final void openBankAccount(int account){

    personalInformation();

    chooseAccountType(account);

}

abstract void personalInformation(); /\*\* evaluate personal information to determine if eligible to open a bank account\*/

abstract void chooseAccountType(int account); /\*\* choose and create the account type \*/

/\*\*Template method pattern

\* sign into a bank account \*/

public final boolean signIn(int accessId){

    //if the access id is not zero then the access id is valid

    //and now the password needs to be evaluated

    if (accessId(accessId)){

        //evaluate password

        if (password(accessId)){

            return true;

        }

    }

    return false;

}

abstract boolean accessId(int accessId); /\*\* evaluate access id for bank account \*/

abstract boolean password(int accessId); /\*\* evaluate password for bank account \*/

public String getAccountName(int accessId){

   for (ArrayList<String> str: identifiers){

        if (Integer.parseInt(str.get(5)) == accessId){

            return str.get(0);

        }

    }

   return null;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\* Strategy Pattern

\* Interface for all account operations to implement \*/

public interface AccountOperationsInterface {

  public double execute(double value, double balance);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Date;

/\*\* Account class for a representation of a bank account \*/

public abstract class Account {

  //declare and initialize variables

  protected AccountOperationsInterface operation;

  protected int id;

protected double balance;

protected double annualInterestRate;

protected Date dateCreated;

/\*\* return the id for this Account \*/

public int getId(){

   return this.id;

}

/\*\* return the balance for this Account \*/

public double getBalance(){

   return this.balance;

}

/\*\* return the annual interest rate for this Account \*/

public double getAnnualInterestRate(){

   return this.annualInterestRate;

}

/\*\* return a string representation of the data this Account was created \*/

public String getDateCreated(){

   return this.dateCreated.toString();

}

/\*\* set the id for this Account \*/

public void setId(int id){

   this.id = id;

}

/\*\* set the balance for this Account \*/

public void setBalance(double balance){

   this.balance = balance;

}

/\*\* set the annual interest rate for this Account \*/

public void setAnnualInterestRate(double annualInterestRate){

   this.annualInterestRate = annualInterestRate;

}

/\*\* return the monthly interest rate for this Account \*/

public double getMonthlyInterestRate(){

   return this.annualInterestRate / 1200;

}

/\*\* Strategy Pattern: set the account operation to be used \*/

  public void setOperation(AccountOperationsInterface operation){

      this.operation = operation;

}

  /\*\* deposit a specified amount into this Account \*/

public void deposit(double amount){

   this.balance  = this.operation.execute(amount, this.balance);

}

/\*\* withdraw a specified amount from this Account \*/

public void withdraw(double amount){

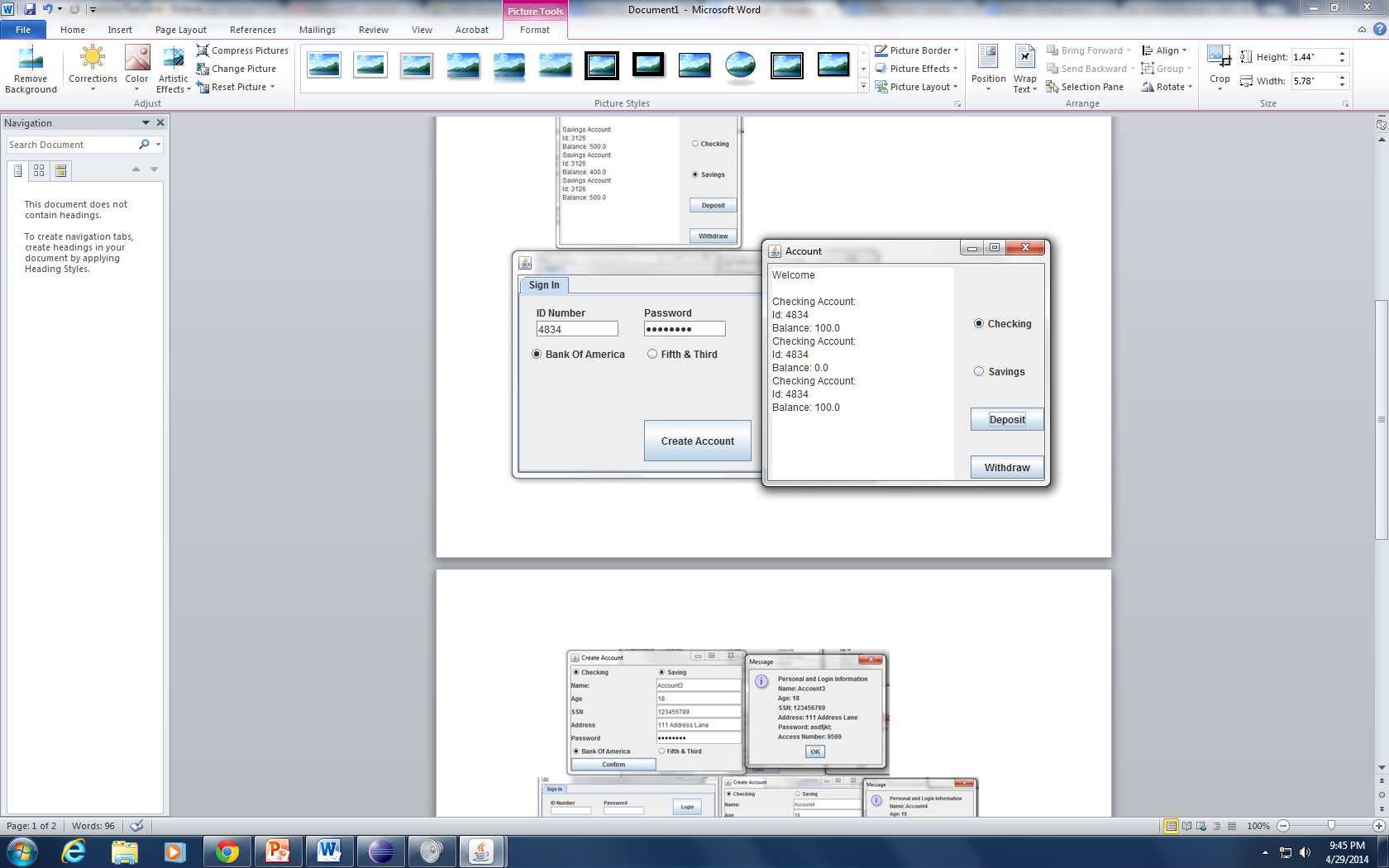
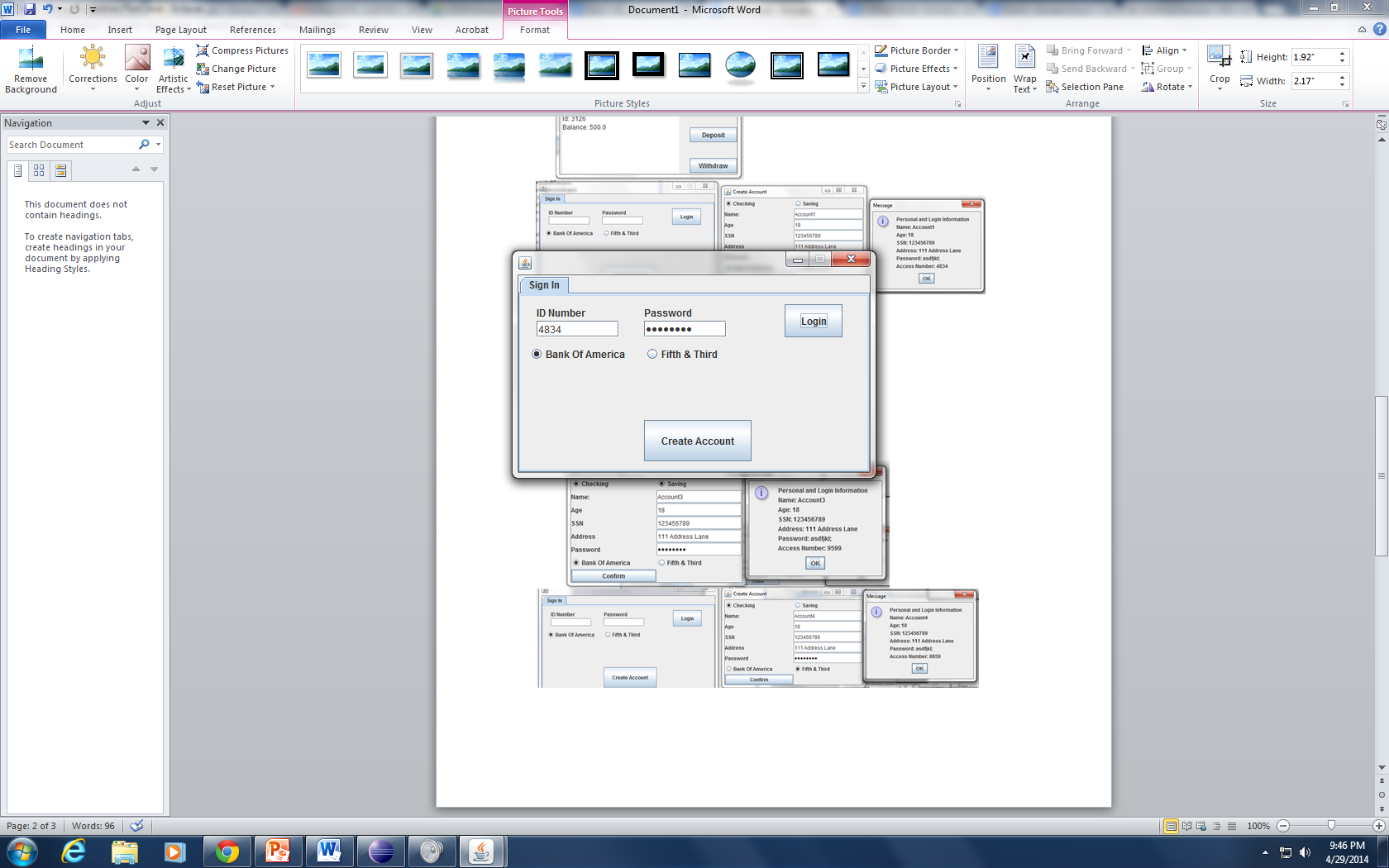
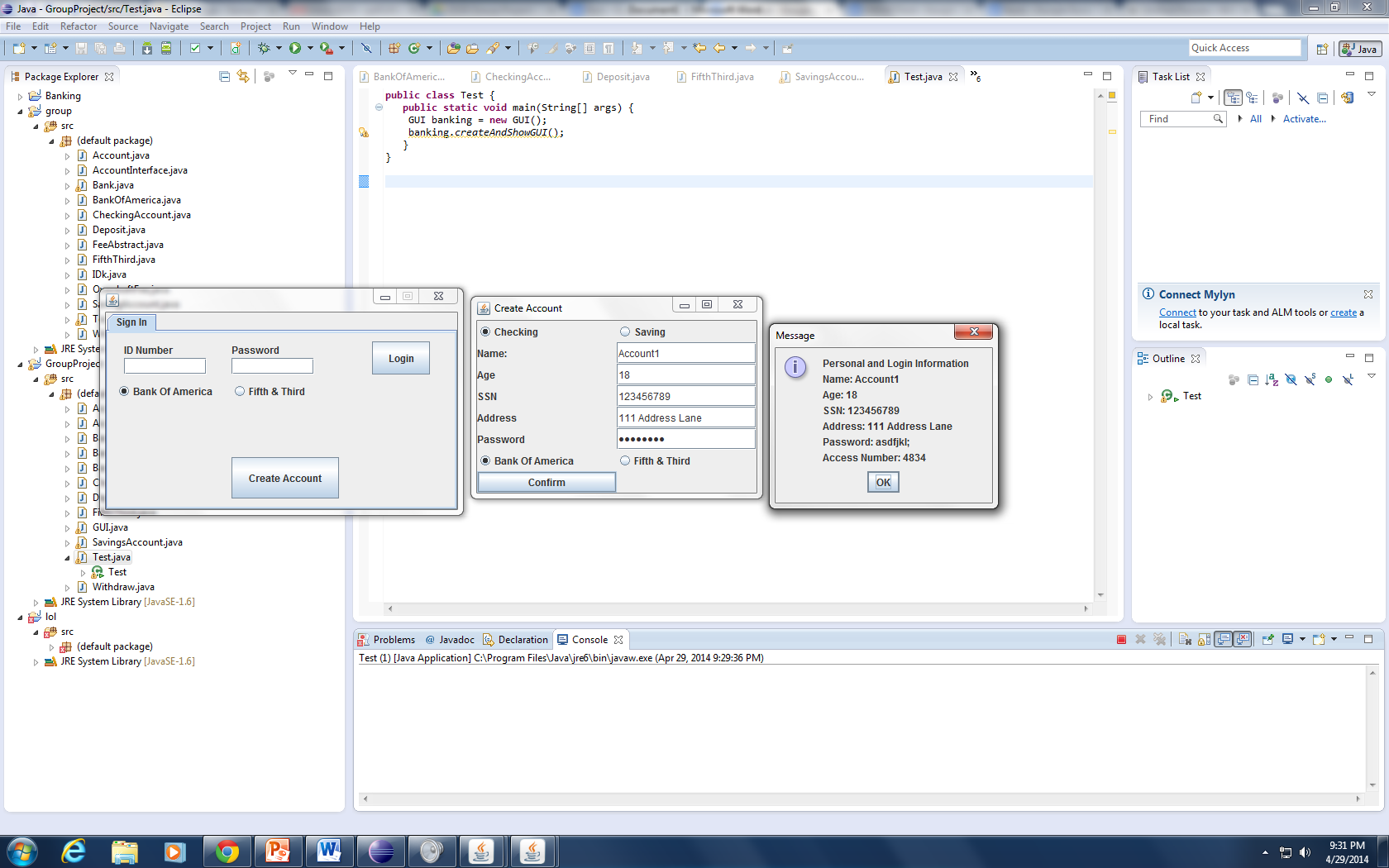
   this.balance = this.operation.execute(amount, this.balance);

}

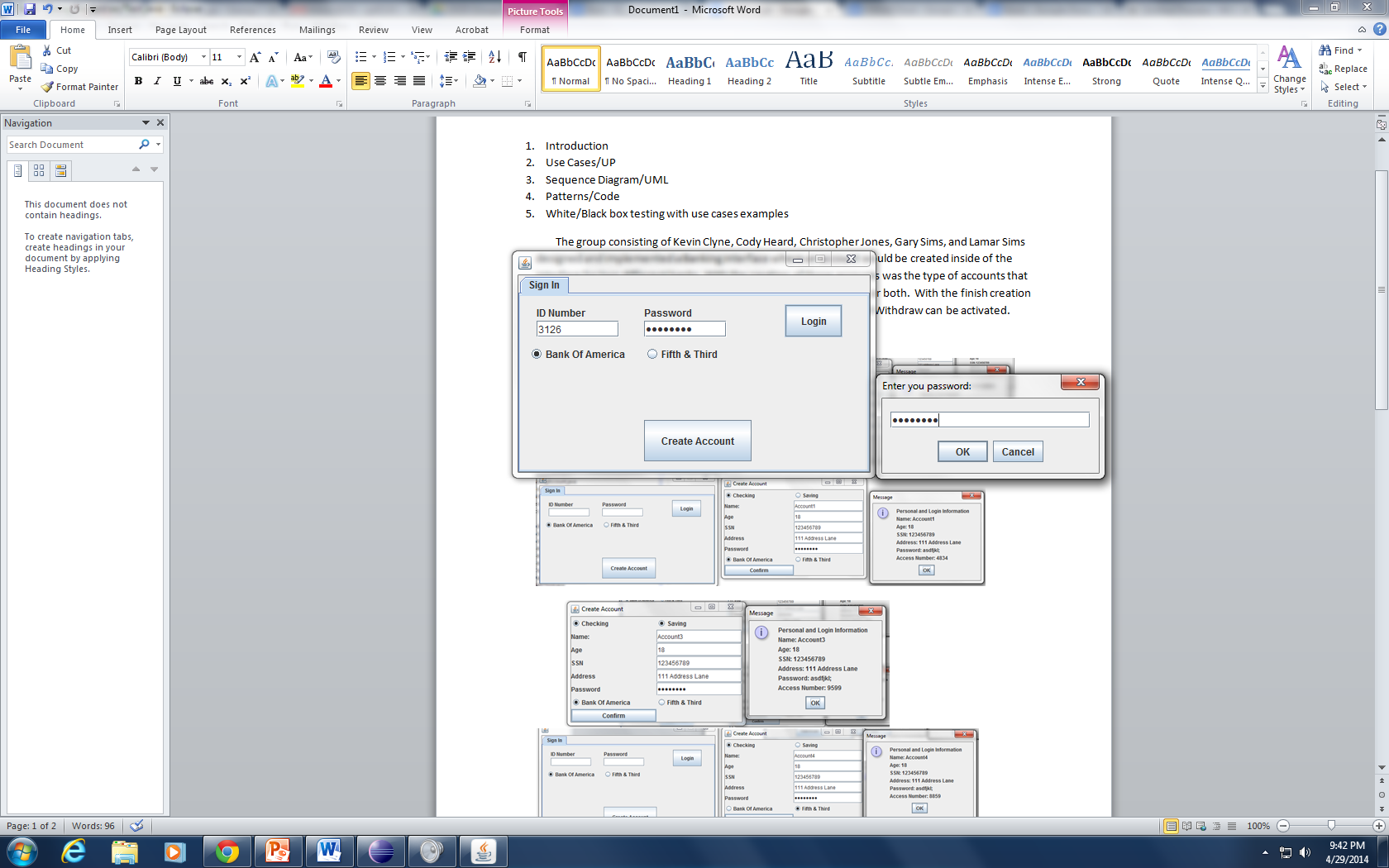
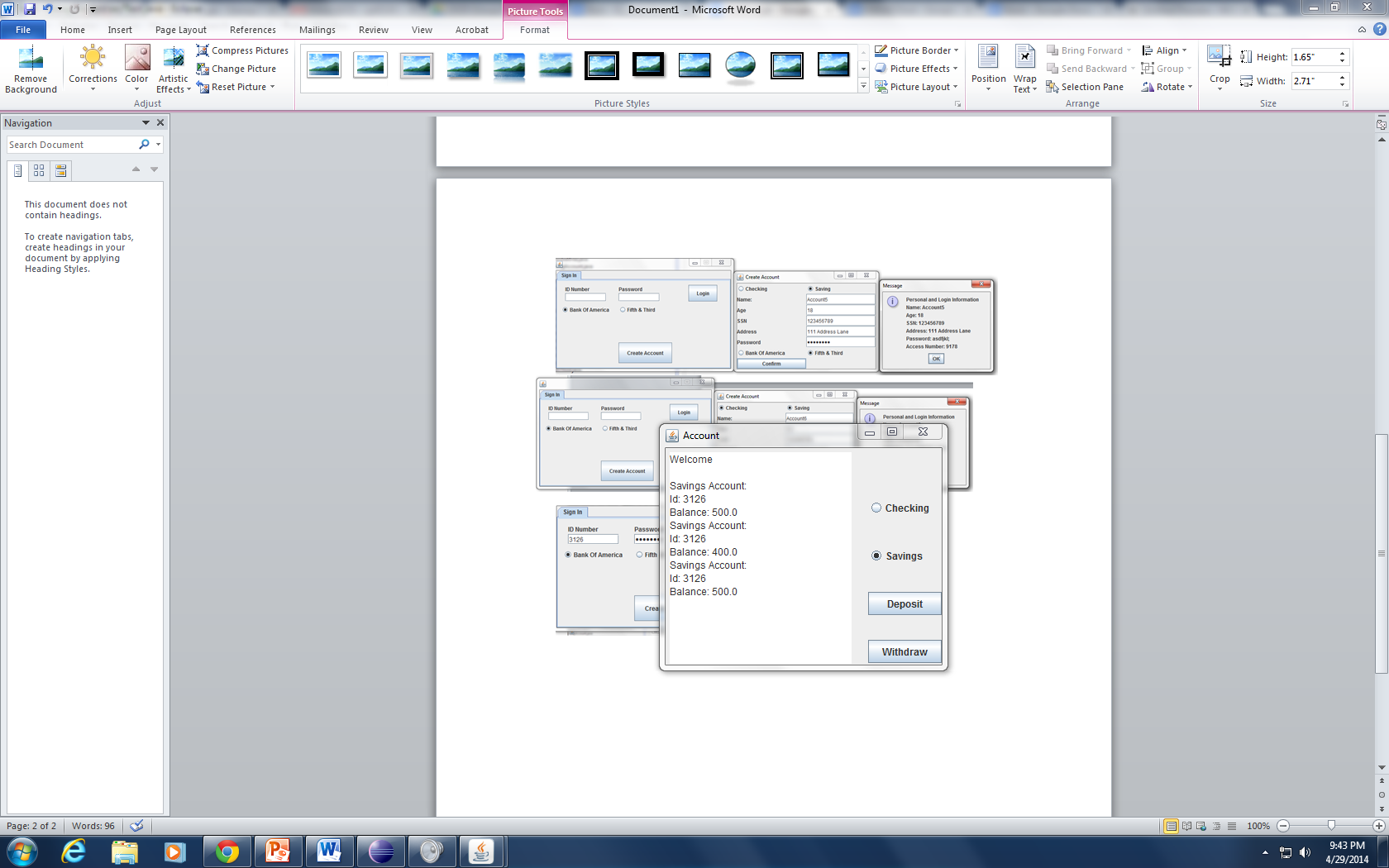
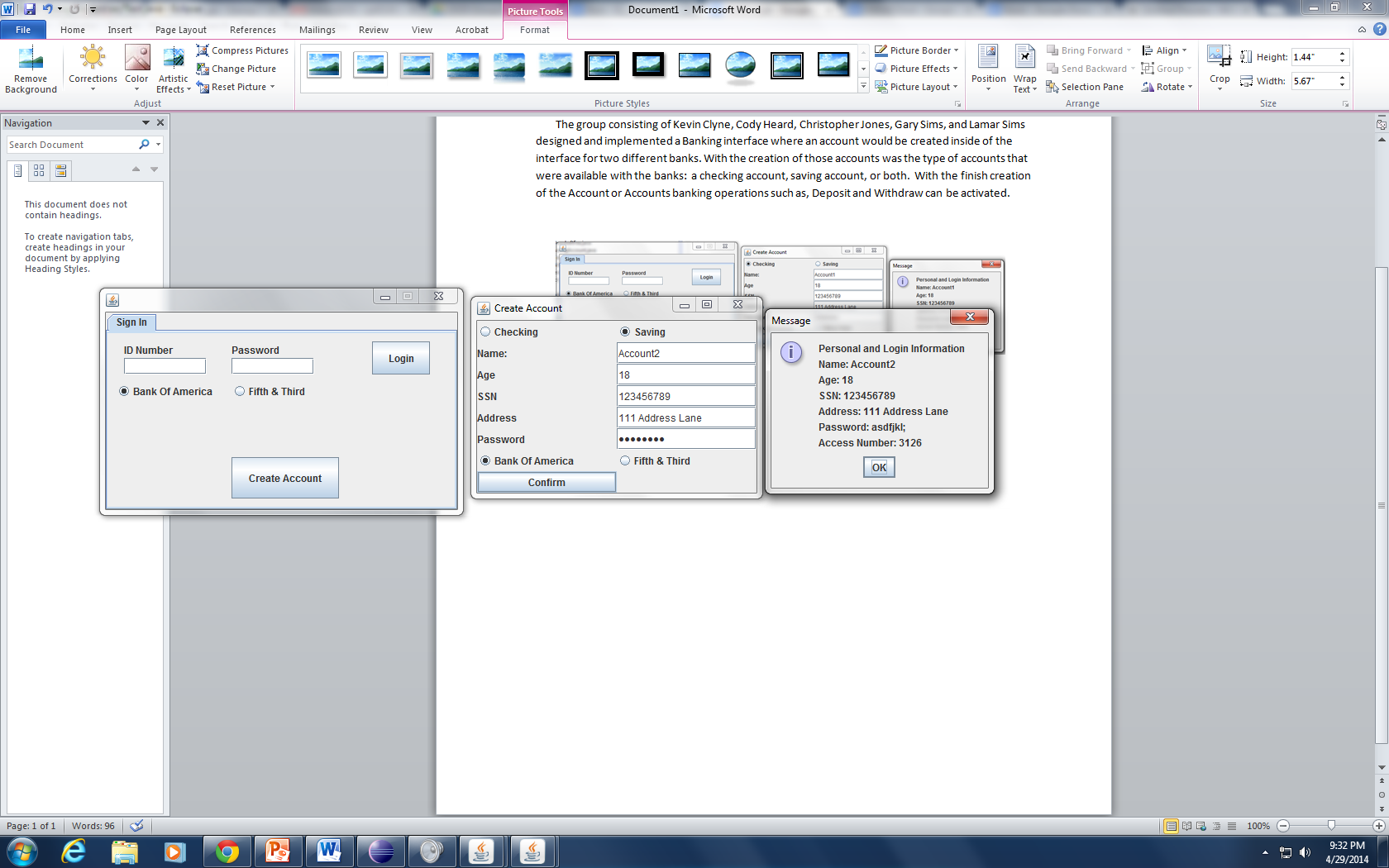
}

**White Box/ Black Box Testing with Use case Examples**

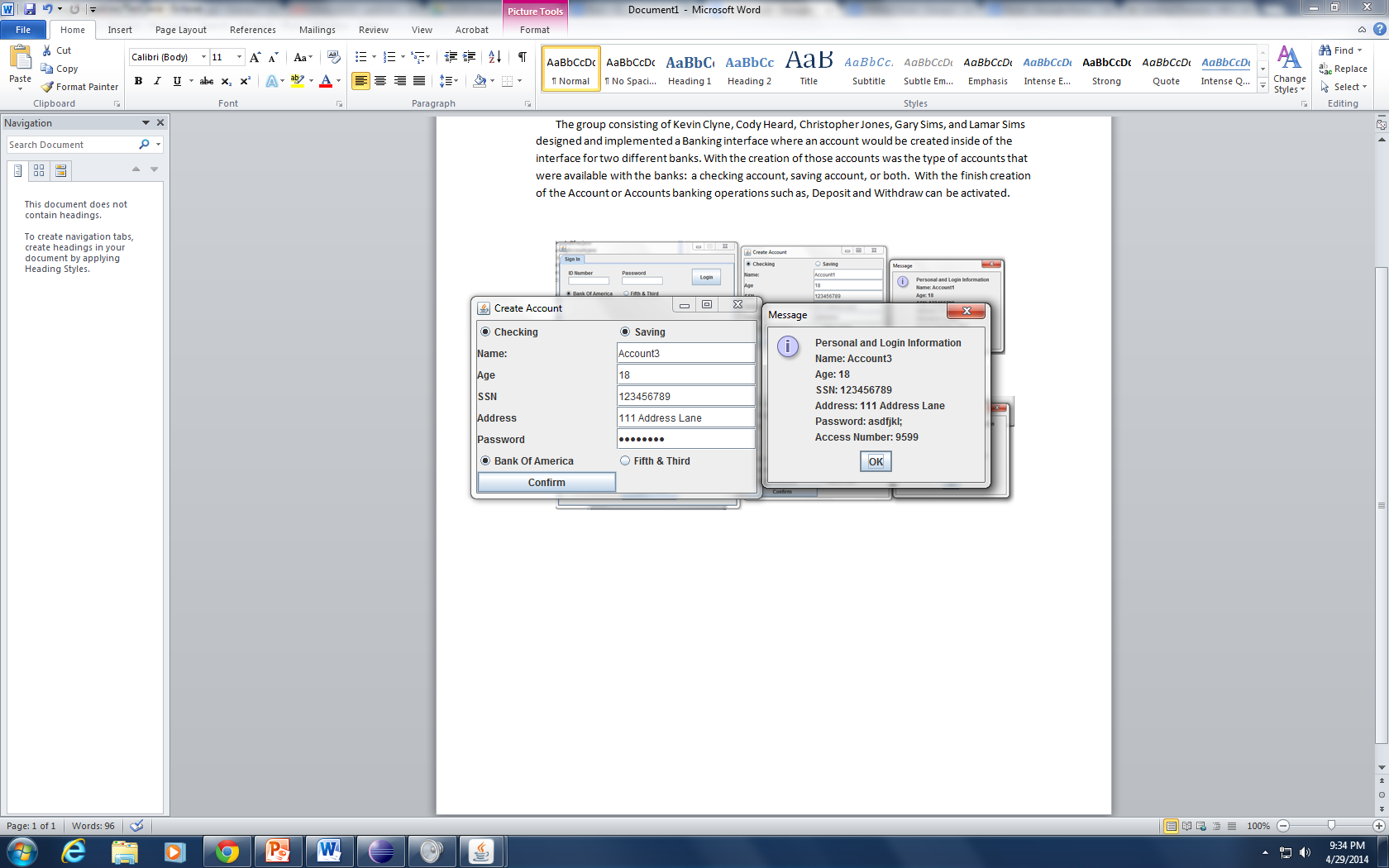
Account1 was Created with the randomly generated number of access number of4834 Checking Account an initial balance of $100. This is A Bank Of America. The Deposit and Withdraw are used and 100 is withdraw and then re-deposited.

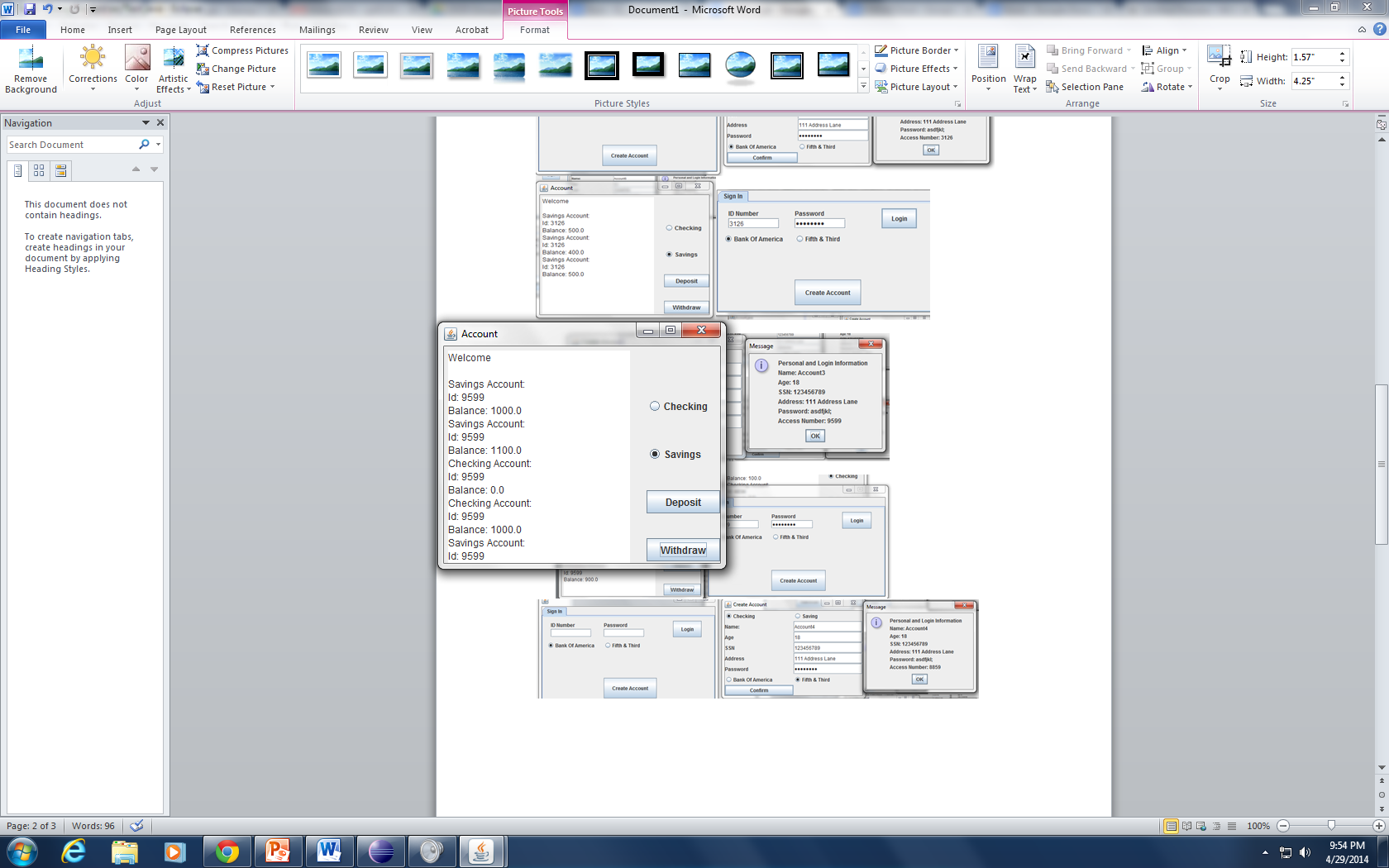
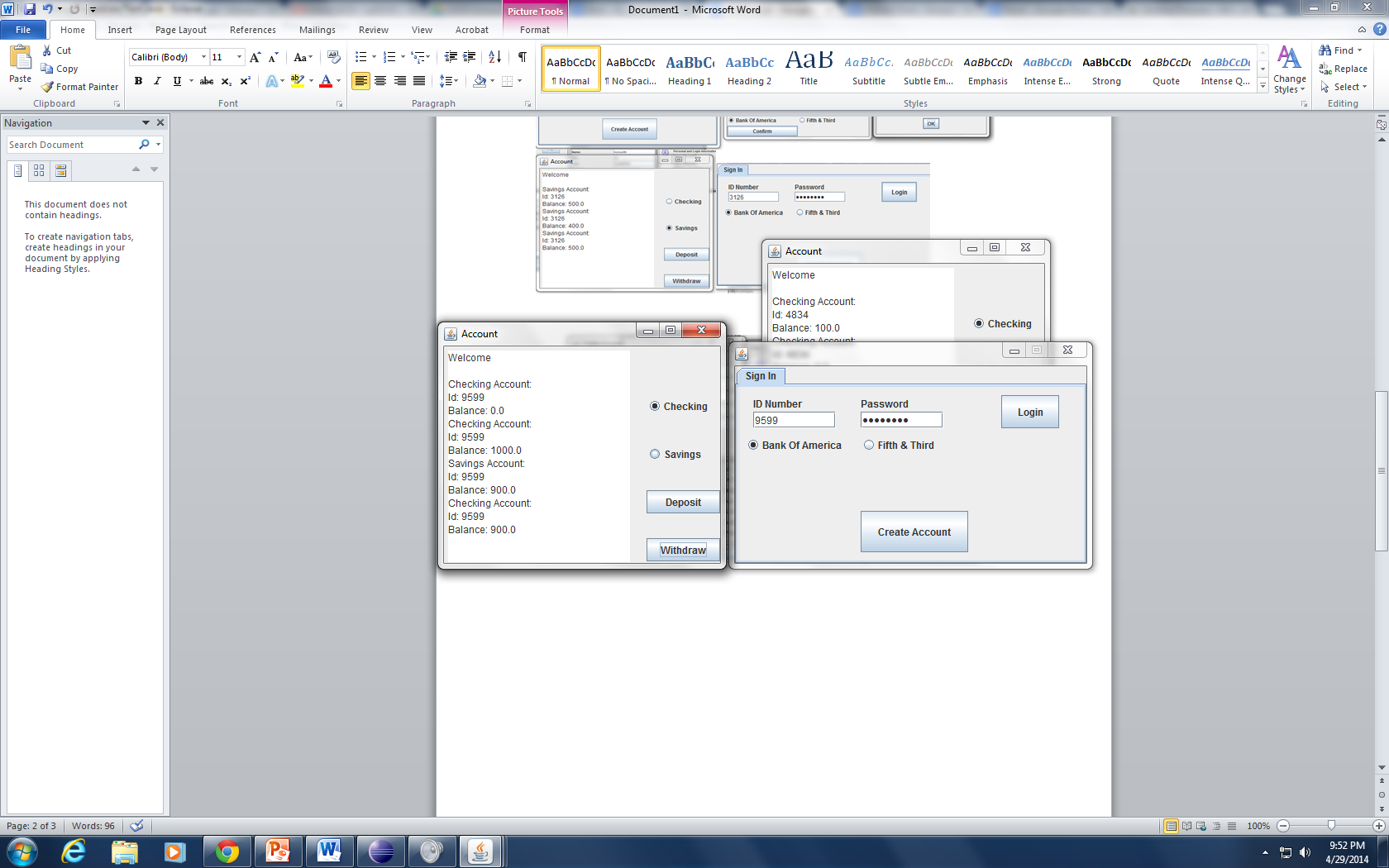


Account2 was Created with the randomly generated number of access number of3126 Saving Account an initial balance of $500.This is a Bank of America. The Deposit and Withdraw are used and 100 is withdraw and then re-deposited.

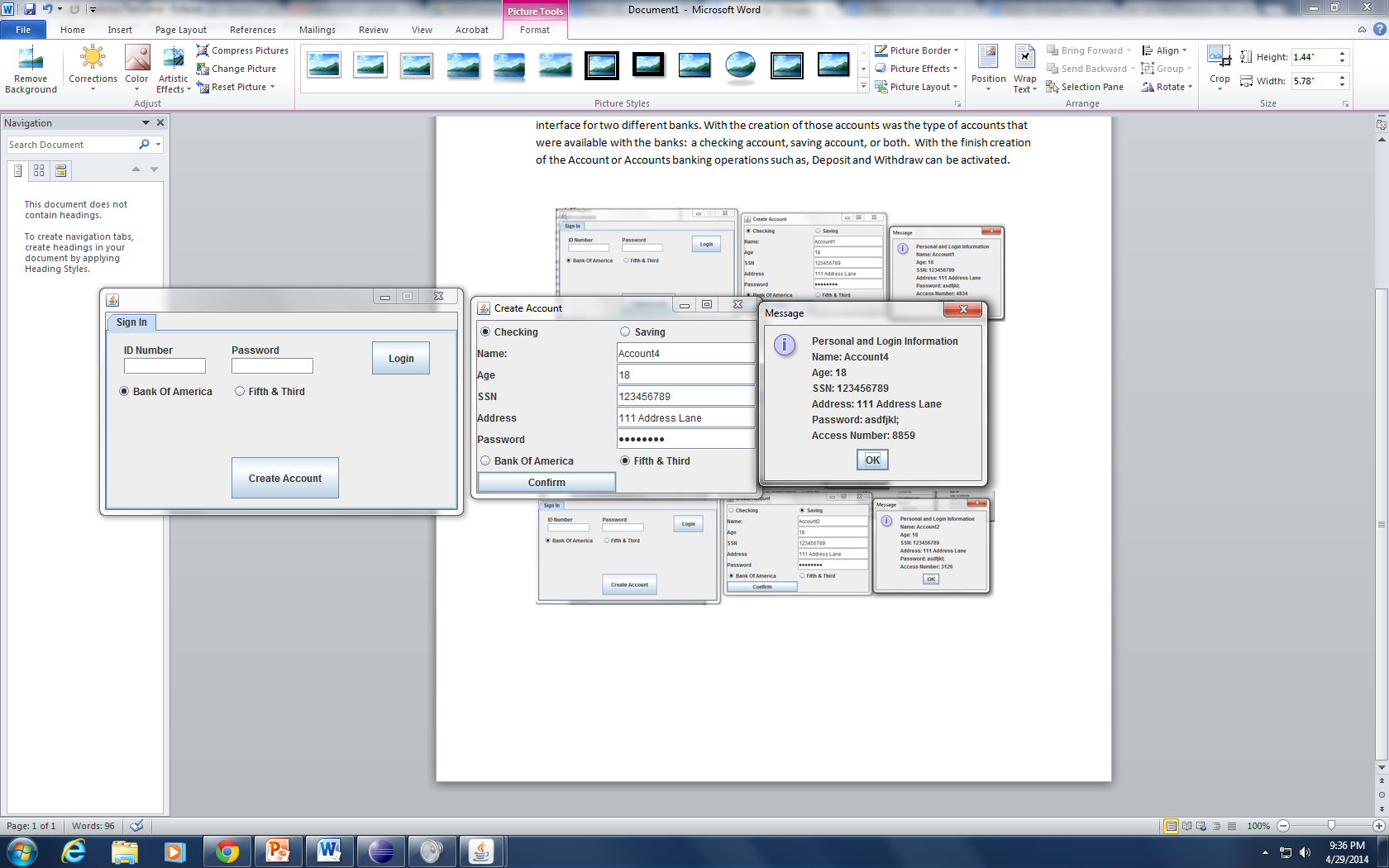


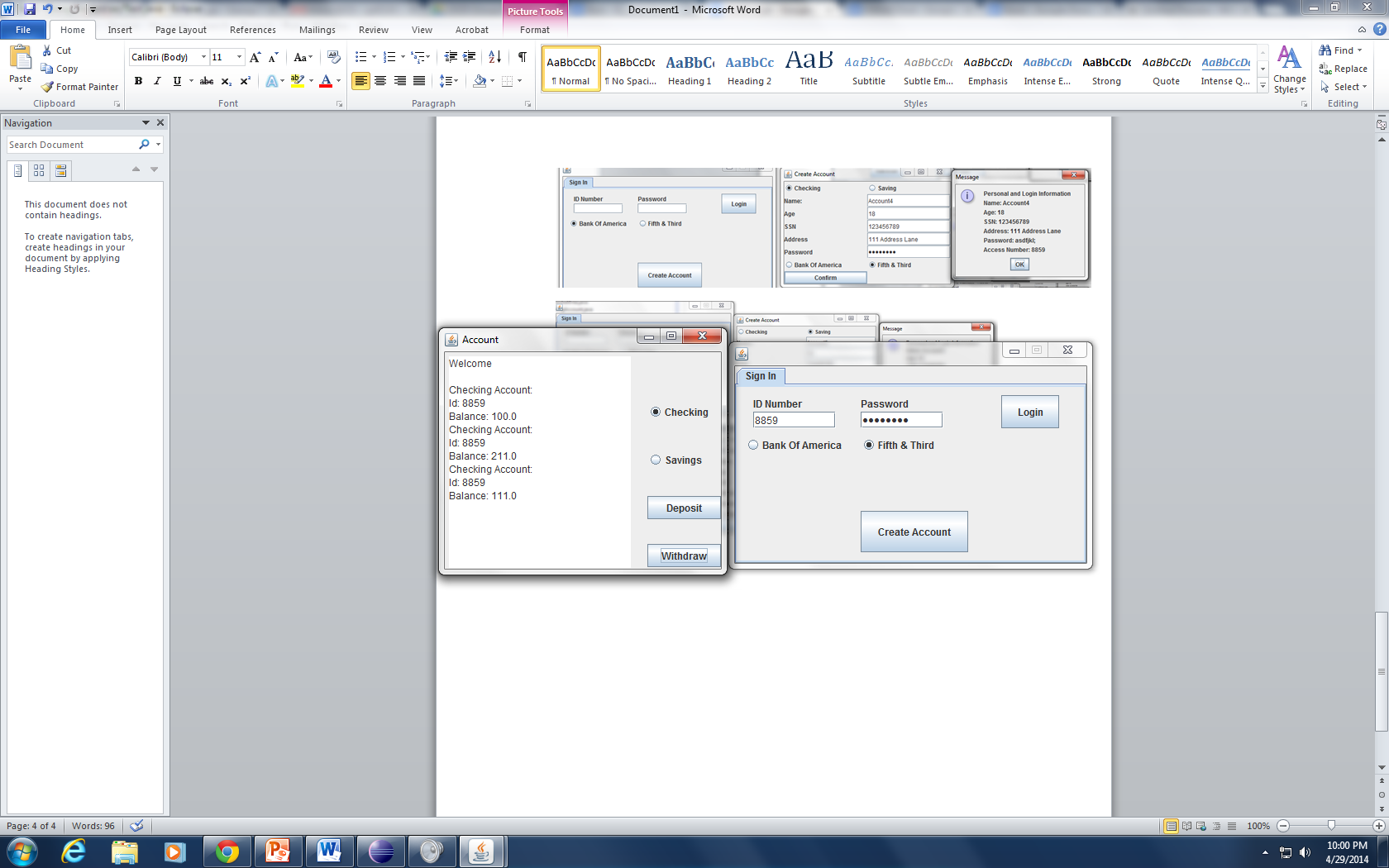
Account3 was Created with the randomly generated number of access number of 9599 Checking Account and Savings Account with an initial balance of $900.This is a Bank of America Account. $100 was then deposited and $1000 was withdrawn.



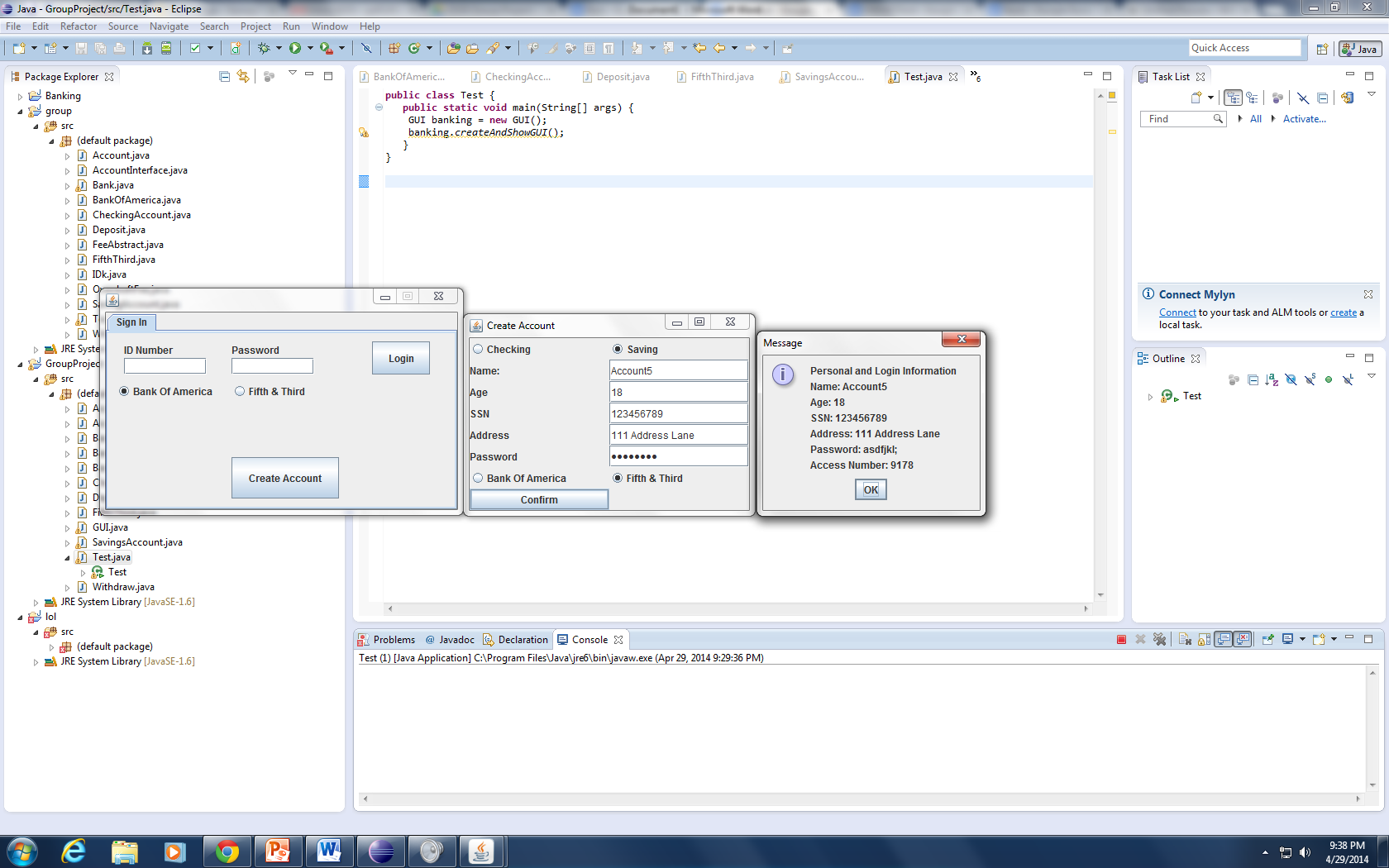


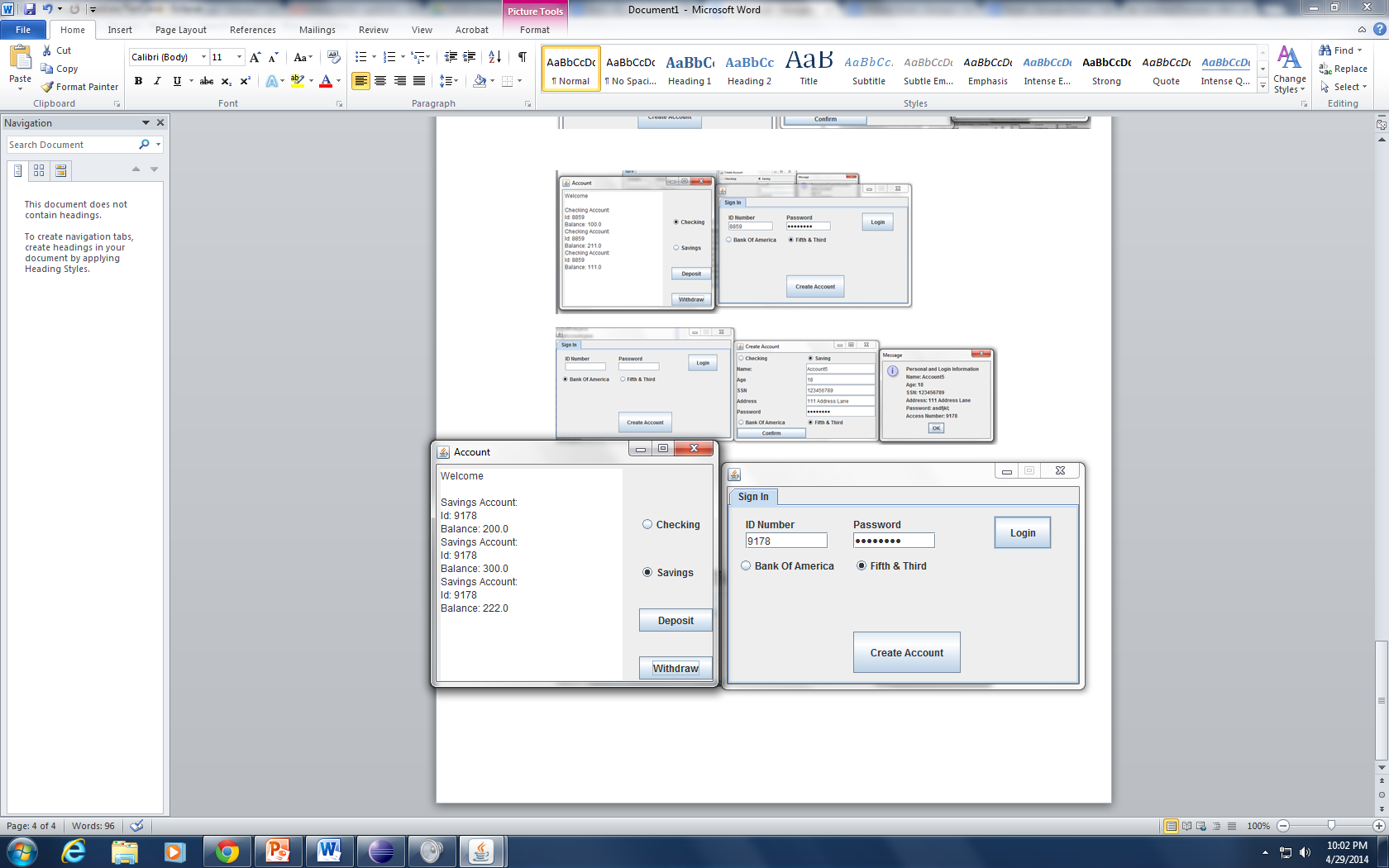
Account4 was Created with the randomly generated number of access number of8859 Checking Account an initial balance of $110. This Account is a Fifth & Third Account. $100 was deposited and $111 was withdrawn





Account5 was Created with the randomly generated number of access number of 9178.This is a Fifth Third Account. Savings Account an initial balance of $222. $78 was deposited and $100 was Withdrawn





Account6 was Created with the randomly generated number of access number of 6436 Checking Account and Savings with an initial balance of $333.This is a Fifth & Third Account. $78 was deposited in the Checking Account and $300 was withdrawn. $78 was deposited and $400 was withdrawn from the Saving Account.

