Benjamin Kleynhans (M00858194)

Professor: Christine Forde

Project #1 Documentation Package

BankDriver Class

CSC 225-185 Advanced JAVA Programming

Fall 2017

Table of Contents

[Project Abstract / Overview 2](#_Toc494283145)

[Program Design 3](#_Toc494283146)

[UML Document 3](#_Toc494283147)

[JavaDoc 4](#_Toc494283148)

[Pseudocode 13](#_Toc494283149)

[Method Definitions 14](#_Toc494283150)

[Source Code 26](#_Toc494283151)

[Sample Input/Output 56](#_Toc494283152)

[Program Start and Main Menu 56](#_Toc494283153)

[Option 1: Deposit sum to account 57](#_Toc494283154)

[Options 1 and 2: Deposit sum to account and Withdraw sum from the account 58](#_Toc494283155)

[Option 2: Insufficient funds 60](#_Toc494283156)

[Option 3: Create and account 61](#_Toc494283157)

[Option 4: View all accounts 62](#_Toc494283158)

[Option 5: Delete an account 63](#_Toc494283159)

[End-of-File Processing (sort the array and print the Customers in the array) 64](#_Toc494283160)

# Project Abstract / Overview

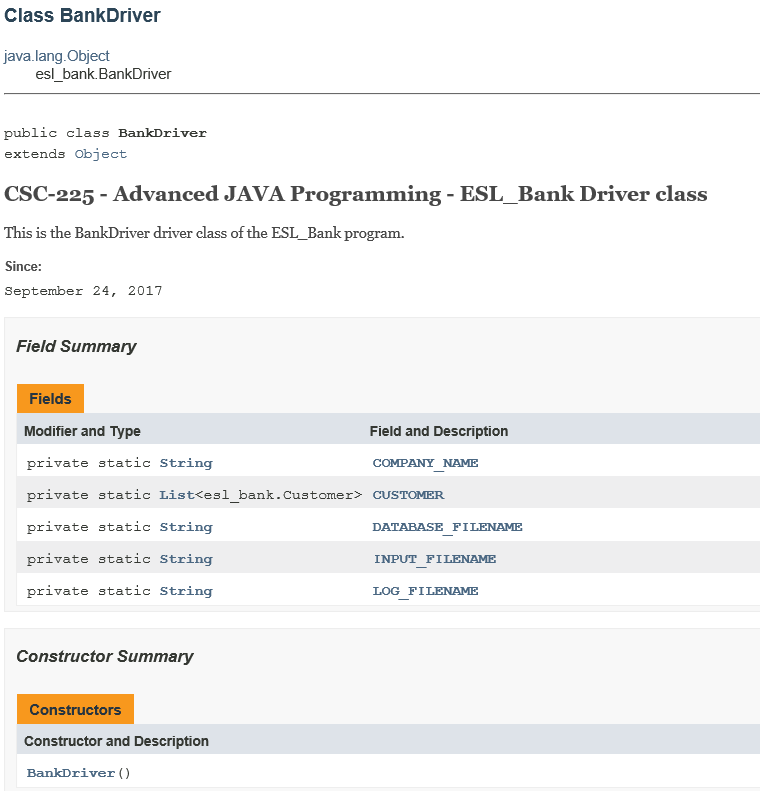
This program is designed to function as a customer account management system at a bank. It provides all the standard account management options of creating a customer, depositing and withdrawing funds from the customer account, deleting a customer and displaying a report of all customers. Whenever an amount is deposited or withdrawn, interest is added to the account and there is a transaction fee for each withdrawal. The program allows you to read customer data from a file and you are able to write the updated data back to a file as well. The system handles all user based illegal operations internally, but creates a log of any file-read or file-write errors for later resolution by a technical specialist. This document contains information pertaining to the BankDriver class.

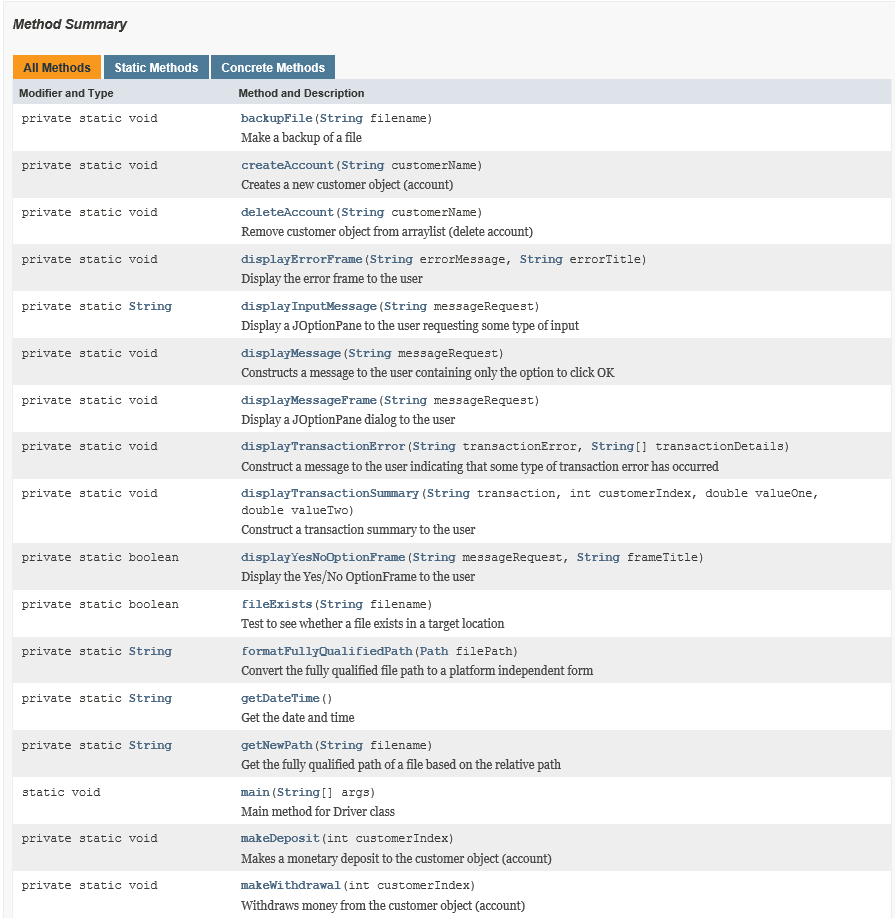
# Program Design

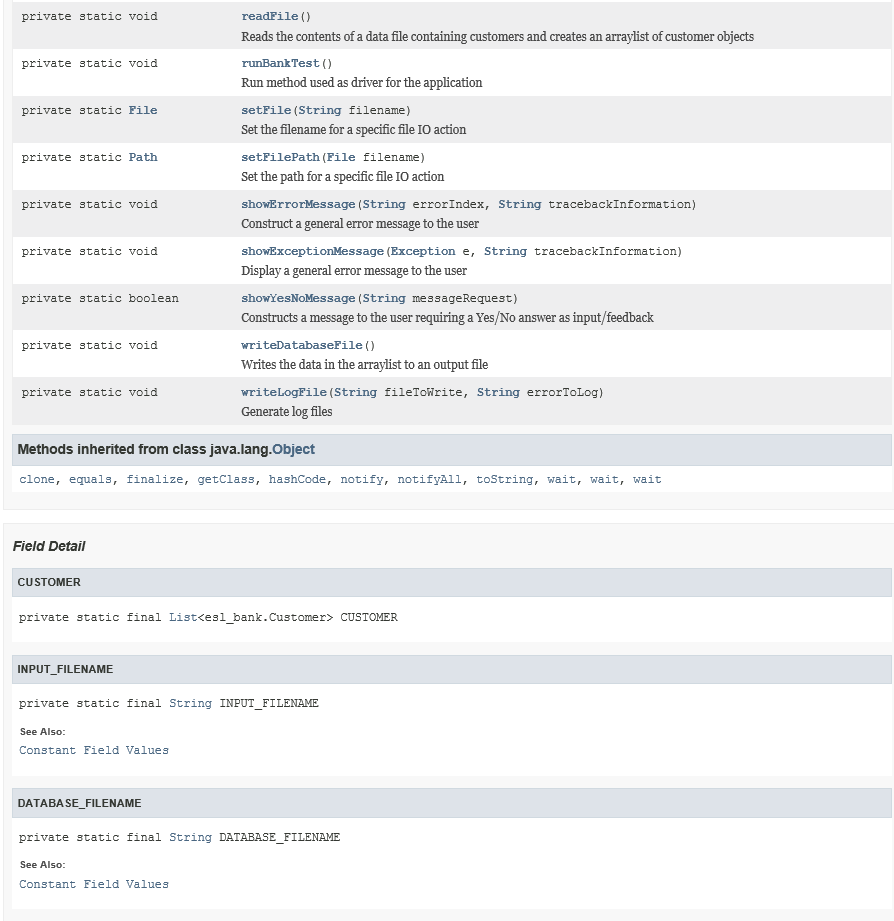
## UML Document

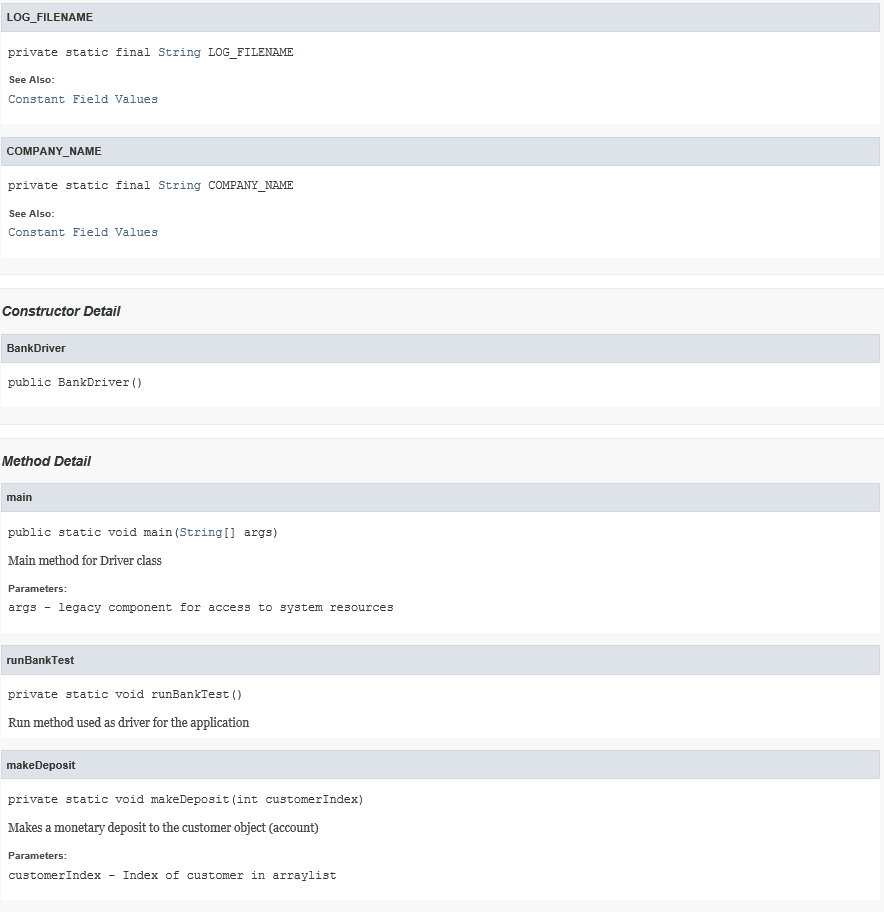


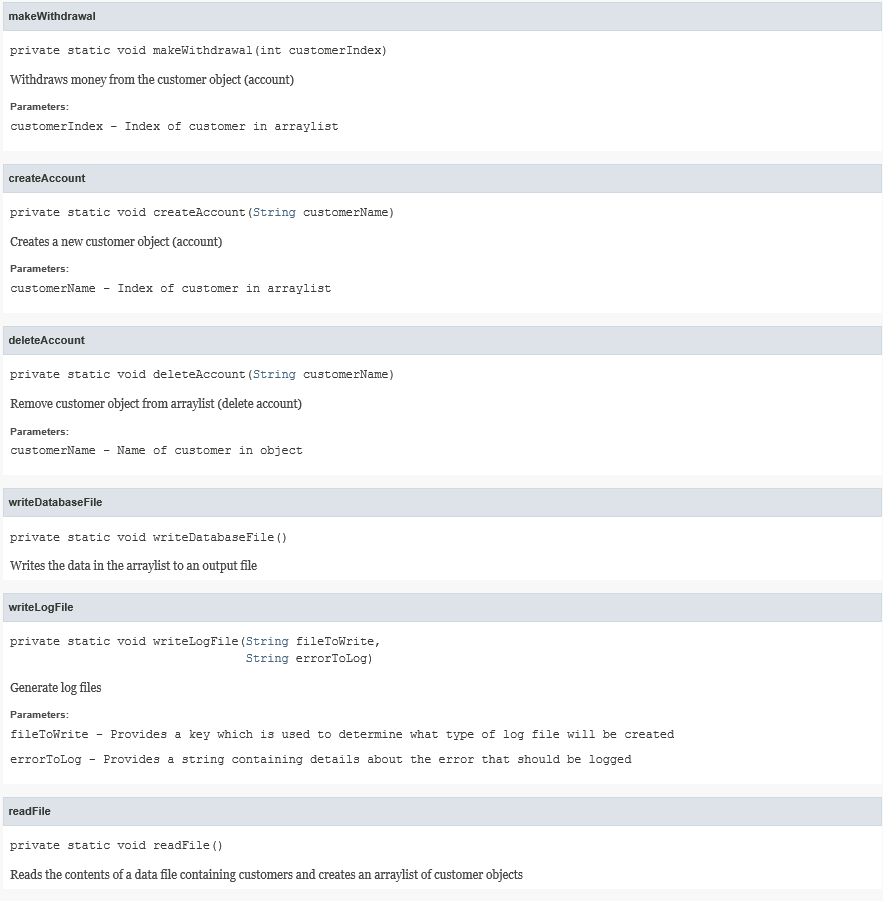
## JavaDoc

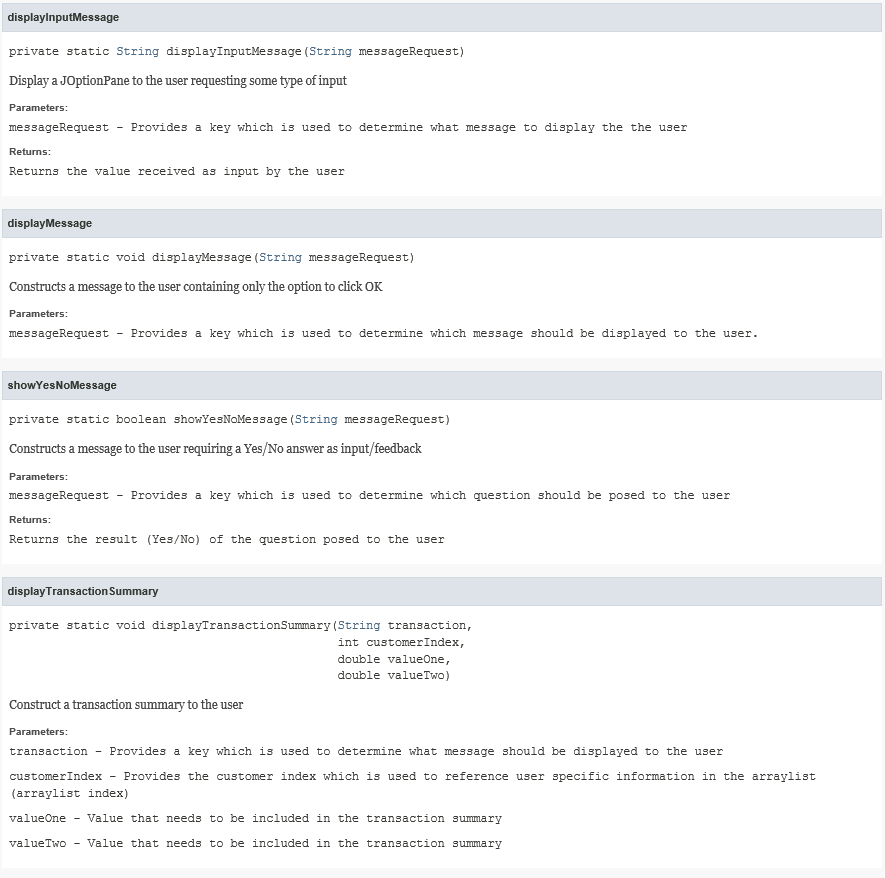


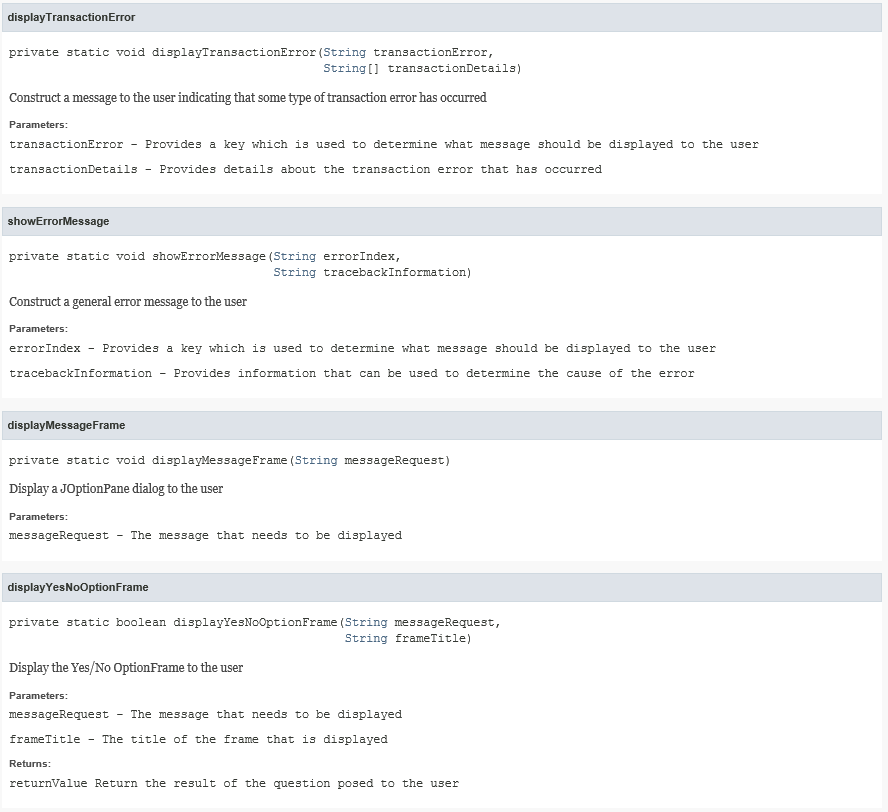


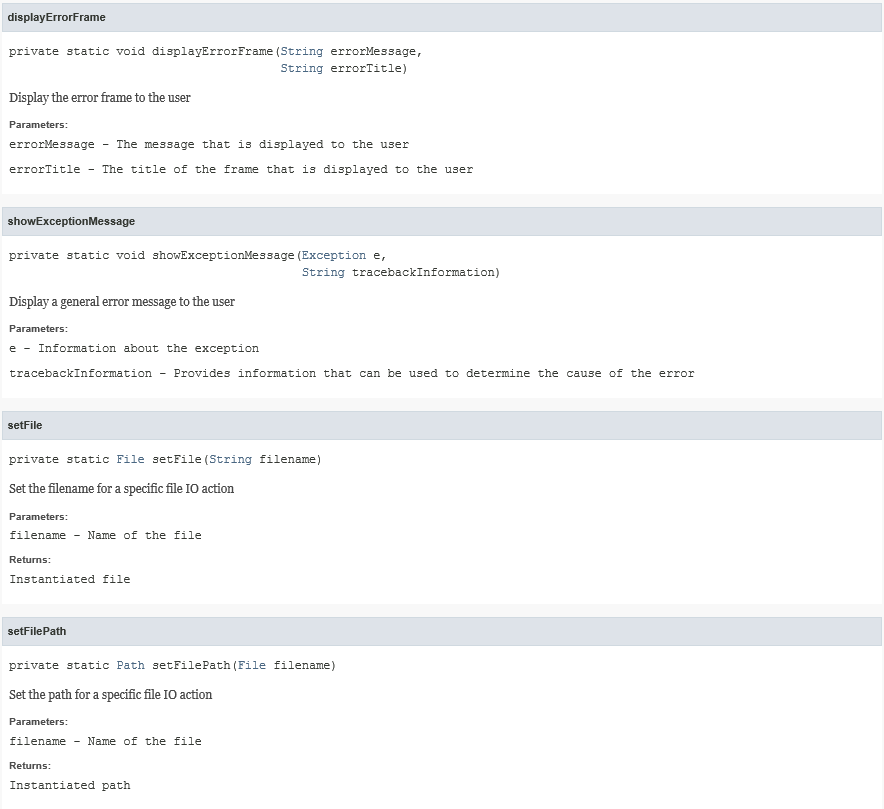


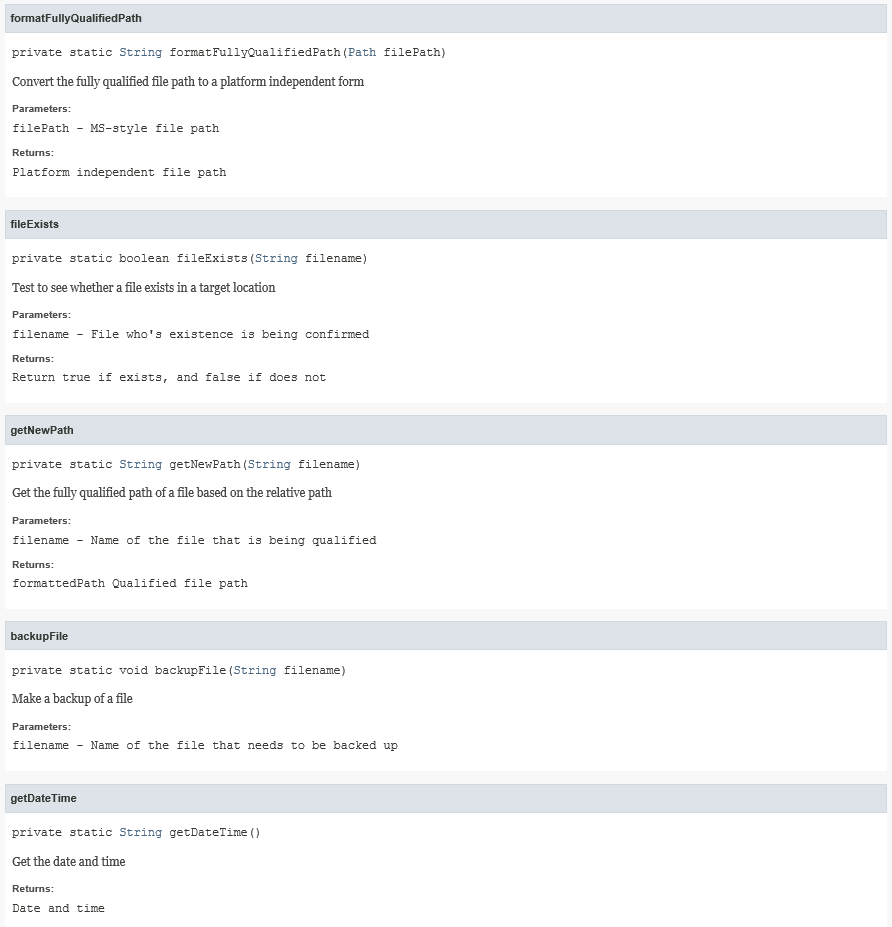












## Pseudocode

BEGIN BankDriver class

Import libraries

Declare Methods

main

runBankTest

makeDeposit

makeWithdrawal

createAccount

deleteAccount

writeDatabaseFile

writeLogFile

readFile

displayInputMessage

displayMessage

showYesNoMessage

displayTransactionSummary

displayTransactionError

showErrorMessage

displayMessageFrame

displayYesNoOption

displayErrorFrame

showExceptionMessage

setFile

setFilePath

formatFullyQualifiedPath

fileExists

getNewPath

backupFile

getDateTime

END BankDriver class

### Method Definitions

#### main

BEGIN main

Display welcome message

Call main method

Display farewell message

END main

#### runBankTest

BEGIN runBankTest

Define variable for menu option

LOOP until user selects the option to exit the program

Define variable for data validation

LOOP until the user makes a valid selection or asks to exit the program

DisplayMenu

IF no option selected

Continue to display menu

END-IF

IF the option selected is invalid

Continue to display menu

END-IF

IF the option selected is valid

THEN

Test if the arraylist is empty

IF the arraylist is empty

THEN

Ask the user if they want to add a customer

IF the user wants to add a customer

THEN

Go to add new user menu option

ELSE

Return the user to the main menu

ELSE-IF the arraylist is empty and the user asked for a report

THEN

Display a message stating there are no records

ELSE

Continue to next logic evaluation

END-IF

ELSE

Continue displaying the menu

END-IF

END-LOOP

IF the menu option selected is NOT one of the reporting options

THEN

LOOP until the user provides valid input or clicks cancel

Ask for the customer name

IF the customer DID NOT clicked the cancel key

THEN

IF the customer wants to add a new customer

THEN

IF the new customer exists in the database

THEN

Inform the user that the customer already exists and ask if they want to create another customer

IF the customer does not want to add another customer

THEN

Return to the main menu

END-IF

ELSE

Continue to the next logic evaluation

END-IF

ELSE-IF the customer DOES NOT exists in the database AND wants to make a deposit or a withdrawal

THEN

Display a message stating that the user does not exist

ELSE

Continue to next logic evaluation

END-IF

END-IF

ELSE-IF the customer didn't make any valid selection

Display the main menu

END-IF

END-LOOP

CASE (menuOption)

CASE 1

Launch make deposit method

CASE 2

Launch make withdrawal method

CASE 3

Launch create account method

CASE 4

Launch delete account method

END-CASE

ELSE-IF The customer wants to print a report

Sort the database and display the report

ELSE-IF The customer wanted to do system maintenance

LOOP until the user makes a valid selection or clicks cancel

Display the maintenance window

IF the user does not make a selection

THEN

Go back to main menu

END-IF

IF the user makes a valid selection

THEN

Go to next logic evaluation

END-IF

END-LOOP

CASE (menuOption)

CASE 1

Read the file into memory

CASE 2

Write memory to file

END-CASE

END-LOOP

END runBankTest

#### makeDeposit

BEGIN makeDeposit (x)

Ask the user for the deposit amount for customer x

IF the user does not click cancel

THEN

Read in the deposit amount

IF the deposit amount is negative

THEN

Display a message informing the user of the mistake

ELSE

Add the amount to the balance of customer x

Add interest to the balance of customer x

Display a summary with the balance before and after interest is added

END-IF

END-IF

END makeDeposit

#### makeWithdrawal

BEGIN makeWithdrawal (x)

Ask the user for the withdrawal amount for customer x

IF the user does not click cancel

THEN

Read the withdrawal amount

IF the withdrawal amount is negative

THEN

Display a message informing the user of the mistake

ELSE-IF the withdrawal amount is more than the balance of the customer

THEN

Display a message informing the user of the mistake

ELSE

Get the balance of the user before withdrawal

Add interest to the balance

Display a summary with the balance before and after interest is added

END-IF

END-IF

END makeWithdrawal

#### createAccount

BEGIN createAccount (x)

Get customer number from customer name x

IF the user does not click cancel

THEN

Ask the user for the opening balance

IF the user does not click cancel

THEN

Ask the user for the telephone number

IF the user does not click cancel

THEN

Create a new customer object with the supplied information

Display summary of new user that will be created

END-IF

END-IF

END-IF

END createAccount

#### deleteAccount

BEGIN deleteAccount (x)

Get customer number from customer name x

IF the user exists

THEN

Define a message to be displayed for confirmation

IF the user confirms the deletion

THEN

Delete the customer

Display a confirmation of deletion message

END-IF

ELSE

Display a message stating that the user does not exist

END-IF

END deleteAccount

#### writeDatabaseFile

BEGIN writeDatabaseFile

Define the File, stream and writer objects

IF the output file has not been defined

THEN

Inform the user that the output file has not been defined

ELSE

IF the output file exists

THEN

Ask the user if they want to backup the old file

IF the user confirms

THEN

create a backup of the file

END-IF

END-IF

Write the records from the arraylist to the file one at a time

END-IF

IF the writer is still opening

THEN

Close the writer

END-IF

END writeDatabaseFile

#### writeLogFile

BEGIN writeLogFile (x, y)

Define the File, stream and writer objects

CASE (x)

Define log file to write based on x

END-CASE

IF the output file has not been defined

THEN

Inform the user that the log file has not been defined

ELSE

IF the log file exists

THEN

Append to the existing file

ELSE

Create a new file

END-IF

END-IF

IF the writer is still opening

THEN

Close the writer

END-IF

END writeLogFile

#### readFile

BEGIN readFile

Define input file, file reader and buffered reader

IF the input file has not been defined

THEN

Inform the user that the input file has not been defined

ELSE

IF the input file exists

THEN

LOOP

Read lines one at a time

IF there is an error

THEN

Write the error to the log file

END-IF

Write the record to the file

END-LOOP

END-IF

END-IF

IF the writer is still opening

THEN

Close the writer

END-IF

END readFile

#### displayInputMessage

BEGIN displayInputMessage (x)

CASE (x)

Define output message based on case request x

END-CASE

LOOP

Request input value from user

IF the value is valid

THEN

IF the requested value is of type double

THEN

Test if the value is a double

IF the value is not of type double

THEN

Display notification of incorrect data provided

END-IF

END-IF

ELSE-IF the value is the cancel button

Cancel and go back to main menu

END-IF

END-LOOP when valid value is entered

Return input value back to calling method

END displayInputMessage

#### displayMessage

BEGIN displayMessage (x)

CASE (x)

Deifne the output message based on x

END-CASE

END displayMessage

#### showYesNoMessage

BEGIN showYesNoMessage (x)

CASE (x)

Define the output message based on x

END-CASE

Display the output message

Return the result of the message to calling method

END showYesNoMessage

#### displayTransactionSummary

BEGIN displayTransactionSummary (x)

CASE (x)

Define the transaction summary based on x

END-CASE

END displayTransactionSummary

#### displayTransactionError

BEGIN displayTransactionError (x)

CASE (x)

Define the transaction error based on x

END-CASE

END displayTransactionError

#### showErrorMessage

BEGIN showErrorMessage

CASE (x)

Define the error message based on x

END-CASE

END showErrorMessage

#### displayMessageFrame

BEGIN displayMessageFrame (x)

Display the message defined in x from calling methods

END displayMessageFrame

#### displayYesNoOption

BEGIN displayYesNoOption (x)

Display the option defined in x from calling methods

Return result

END displayYesNoOption

#### displayErrorFrame

BEGIN displayErrorFrame (x, y)

Display the error frame with message x and title y

END displayErrorFrame

#### showExceptionMessage

BEGIN showExceptionMessage (x, y)

Isolate the exception by dissasembling the exeception passed (x)

CASE (isolated message)

Display an exception message based on information extraced from x and y for traceback information

END-CASE

END showExceptionMessage

#### setFile

BEGIN setFile (x)

Set the file object with x

Return File object

END setFile

#### setFilePath (x)

BEGIN setFilePath

Set the file path object with x

Return Path object

END setFilePath

#### formatFullyQualifiedPath

BEGIN formatFullyQualifiedPath (x)

Format fully qualified path x for use by JAVA

Return formatted path

END formatFullyQualifiedPath

#### fileExists

BEGIN fileExists

Test if file exists

Return true if file exists or false if file does not exist

END fileExists

#### getNewPath

BEGIN getNewPath (x)

Get relative path of file x

Interpret fully qualified path

Format the fully qualified path string

Return formatted path string

END getNewPath

#### backupFile

BEGIN backupFile (x)

Set the source path for file x

Set the destination path for file x

Create the new file name for x with date and time of backup

Create backup of old file

END backupFile

#### getDateTime

BEGIN getDateTime ()

Get and format the current date and time

Return current date and time to calling method

END getDateTime

## Source Code

package esl\_bank;

/\*

\* CSC-225 - Advanced JAVA Programming

\* Project 1 - ES&L Bank program

\* Class Description : This is the BankDriver driver class of the ESL\_Bank program.

\* Author : Benjamin Kleynhans

\* Date : September 24, 2017

\* Filename : BankDriver.java

\*/

import java.awt.Dimension; // Used for JOptionPane formatting

import java.awt.Font;

import java.io.\*; // Used for file management

import java.io.File;

import java.io.IOException;

import java.nio.file.Files; // Used for file management

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.StandardCopyOption;

import java.time.LocalDateTime; // Used for date and time formatting

import java.time.format.DateTimeFormatter;

import java.util.\*; // Used for arraylist and general functions

import java.util.ArrayList;

import javax.swing.\*; // Used for JOptionPane/swing formatting

/\*\*

\* <h1>

\* CSC-225 - Advanced JAVA Programming - ESL\_Bank Driver class

\* </h1>

\* <p>

\* This is the BankDriver driver class of the ESL\_Bank program.

\* </p>

\*

\* @version 1.0

\* @since September 24, 2017

\*/

public class BankDriver {

private static final List<Customer> CUSTOMER = new ArrayList<Customer>(); // Create an arraylist of customer objects

private static final String INPUT\_FILENAME = "customer.txt"; // Define input and output files

private static final String DATABASE\_FILENAME = "customer.txt";

private static final String LOG\_FILENAME = "logfile.txt";

private static final String COMPANY\_NAME = "ES&L Bank";

/\*\*

\* Main method for Driver class

\*

\* @param args legacy component for access to system resources

\*/

public static void main(String[] args) {

displayMessage("WelcomeMessage"); // Display welcome message to user

runBankTest(); // Launch menu for continued execution

displayMessage("FarewellMessage"); // Display farewell message to user

}

/\*\*

\* Run method used as driver for the application

\*/

private static void runBankTest() {

String menuOption = new String();

do {

boolean validInput = false;

do {

try {

menuOption = JOptionPane.showInputDialog( // Display main menu

null,

"Please select an option\n"

+ "\n"

+ "1. Deposit sum to account\n"

+ "2. Withdraw sum from account\n"

+ "3. Create account\n"

+ "4. View all accounts\n"

+ "5. Delete an account\n"

+ "\n"

+ "7. System maintenance\n"

+ "\n"

+ "9. Quit\n"

+ "\n",

"ES&L Bank",

JOptionPane.QUESTION\_MESSAGE);

Integer.parseInt(menuOption);

if (menuOption == null) { // If no option chosen, do nothing

menuOption = "99";

}

if (CUSTOMER.isEmpty() &&

!menuOption.equals("9") &&

!menuOption.equals("99") &&

!menuOption.equals("7")) {

boolean loadDefaultFile = showYesNoMessage("LoadDefaultDataFile");

if (loadDefaultFile) {

readFile();

displayMessage("DatabaseLoaded");

}

}

//Integer.parseInt(menuOption);

if (((Integer.parseInt(menuOption) >= 1) && (Integer.parseInt(menuOption) <= 5))

|| (Integer.parseInt(menuOption) == 7)

|| (Integer.parseInt(menuOption) == 9)) {

if (CUSTOMER.isEmpty() // Test if arraylist is empty

&& ((Integer.parseInt(menuOption) == 1)

|| (Integer.parseInt(menuOption) == 2))) {

boolean addUser = showYesNoMessage("DatasetEmpty"); // If empty, ask if user wants to add customers, returns true if yes

if (addUser) {

menuOption = "3"; // Invoke create account menu option

validInput = true;

} else {

menuOption = "99"; // Don't close window until valid input received

}

} else if (CUSTOMER.isEmpty() && (Integer.parseInt(menuOption) == 4)) {

displayMessage("NoDataToDisplay"); // If arraylist empty, inform the user

menuOption = "99";

} else {

validInput = true;

}

}

} catch (Exception e) {

showExceptionMessage(e, ""); //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

} while (!validInput); // Continue displaying menu until valid input is received

validInput = false;

if ((Integer.parseInt(menuOption) != 4)

&& (Integer.parseInt(menuOption) != 7)

&& (Integer.parseInt(menuOption) != 9)) {

String customerName;

int customerIndex;

do {

customerName = displayInputMessage("RequestName"); // Ask the user to enter a customer name

customerIndex = Customer.findIndex(CUSTOMER, customerName); // Find the index of the customer

if (!customerName.equals("CancelPressed")) { // Return to menu if cancel key pressed

if (Integer.parseInt(menuOption) == 3) { // If the customer requested to create a user

if (customerIndex != -1) { // see if the customer exists

boolean addUser = showYesNoMessage("CustomerExists"); // Inform the user if the customer exists

// and return to main menu

if (!addUser) {

menuOption = "99"; // If the customer does not exist, continue

validInput = true; // to add a new user

}

} else {

validInput = true;

}

} else if ((customerIndex == -1) // If the customer is not found

&& ((Integer.parseInt(menuOption) == 1)

|| (Integer.parseInt(menuOption) == 2))) {

boolean addUser = showYesNoMessage("CustomerNotFound"); // Inform the user and ask

// if they want to create

if (addUser) { // a new user

menuOption = "3"; // If yes, continue to user creation

validInput = true;

} else {

menuOption = "99"; // If no, return to main menu

validInput = true;

}

} else {

validInput = true;

}

} else {

menuOption = "99";

validInput = true;

}

} while (!validInput);

switch (Integer.parseInt(menuOption)) {

case 1:

makeDeposit(customerIndex); // call make deposit method

break;

case 2:

makeWithdrawal(customerIndex); // call make withdrawal method

break;

case 3:

createAccount(customerName); // call create customer method

break;

case 5:

deleteAccount(customerName); // call delete customer method

break;

default:

break;

}

} else if (Integer.parseInt(menuOption) == 4) {

Customer.nameSort(CUSTOMER); // sort arraylist

displayMessage("ShowAllAccounts"); // display all customers

} else if (Integer.parseInt(menuOption) == 7) {

String maintenanceMenuOption = new String();

do {

maintenanceMenuOption = JOptionPane.showInputDialog(

null,

"Please select an option\n"

+ "\n"

+ "1. Load customer data from file\n" // load records from file

+ "2. Write customer data to file\n" // write records to file

+ "\n",

COMPANY\_NAME + " System",

JOptionPane.QUESTION\_MESSAGE);

if (maintenanceMenuOption == null) {

maintenanceMenuOption = "9";

}

try {

Integer.parseInt(maintenanceMenuOption);

if (maintenanceMenuOption.equals("1") ||

maintenanceMenuOption.equals("2") ||

maintenanceMenuOption.equals("9")) {

validInput = true;

}

} catch (Exception e) {

showExceptionMessage(e, "maintenanceMenuOption");

}

} while (!validInput);

switch (Integer.parseInt(maintenanceMenuOption)) {

case 1:

readFile();

displayMessage("DatabaseLoaded");

break;

case 2:

writeDatabaseFile();

displayMessage("WrittenToFile");

break;

default:

break;

}

}

} while (Integer.parseInt(menuOption) != 9);

}

/\*\*

\* Makes a monetary deposit to the customer object (account)

\*

\* @param customerIndex Index of customer in arraylist

\*/

private static void makeDeposit(int customerIndex) {

String tempValue = displayInputMessage("RequestDepositAmount"); // get deposit amount from user

if (!tempValue.equals("CancelPressed")) { // continue if Cancel button not pressed

double depositAmount = Double.parseDouble(tempValue);

if (depositAmount < 0) { // test for negative amount

String[] errorMessage = new String[]{ // define negative amount deposit message

CUSTOMER.get(customerIndex).getCustomerName(),

Double.toString(depositAmount)

};

displayTransactionError("NegativeAmountDeposit", errorMessage); // display negative amount deposit message

} else {

CUSTOMER.get(customerIndex).deposit(depositAmount); // make a deposit

double oldBalance = CUSTOMER.get(customerIndex).getBalance();

CUSTOMER.get(customerIndex).addInterest();

double newBalance = CUSTOMER.get(customerIndex).getBalance();

// display deposit confirmation message

displayTransactionSummary("makeDeposit", customerIndex, oldBalance, newBalance);

}

}

}

/\*\*

\* Withdraws money from the customer object (account)

\*

\* @param customerIndex Index of customer in arraylist

\*/

private static void makeWithdrawal(int customerIndex) {

String tempValue = displayInputMessage("RequestWithdrawalAmount"); // request amount for withdrawal

if (!tempValue.equals("CancelPressed")) { // continue if Cancel button not pressed

double withdrawalAmount = Double.parseDouble(tempValue);

if (withdrawalAmount < 0) { // test for negative amount

String[] errorMessage = new String[]{ // define negative amount deposit message

CUSTOMER.get(customerIndex).getCustomerName(),

Double.toString(withdrawalAmount)

};

displayTransactionError("NegativeAmountWithdrawal", errorMessage); // display negative amount withdrawal message

} else if (withdrawalAmount > CUSTOMER.get(customerIndex).getBalance()) { // test for sufficient funds

String[] errorMessage = new String[]{ // define insufficient funds message

CUSTOMER.get(customerIndex).getCustomerName(),

Double.toString(withdrawalAmount + 1.50),

Double.toString(CUSTOMER.get(customerIndex).getBalance())

};

displayTransactionError("InsufficientFunds", errorMessage); // display insufficient funds message

} else {

CUSTOMER.get(customerIndex).withdraw(withdrawalAmount); // withdraw amount from customer

// double oldBalance = CUSTOMER.get(customerIndex).getBalance(); // define confirmation message

// CUSTOMER.get(customerIndex).addInterest();

// double newBalance = CUSTOMER.get(customerIndex).getBalance();

// // display amount withdrawal message

// displayTransactionSummary("makeWithdrawal", customerIndex, oldBalance, newBalance);

}

double oldBalance = CUSTOMER.get(customerIndex).getBalance(); // define confirmation message

CUSTOMER.get(customerIndex).addInterest();

double newBalance = CUSTOMER.get(customerIndex).getBalance();

// display amount withdrawal message

displayTransactionSummary("makeWithdrawal", customerIndex, oldBalance, newBalance);

}

}

/\*\*

\* Creates a new customer object (account)

\*

\* @param customerName Index of customer in arraylist

\*/

private static void createAccount(String customerName) {

String customerNumber;

double balanceAmount;

String phoneNumber;

customerNumber = displayInputMessage("RequestCustomerNumber"); // get customer number input

if (!customerNumber.equals("CancelPressed")) { // test if cancel key is pressed

String tempValue = displayInputMessage("RequestBalanceAmount"); // get opening balance amount

if (!tempValue.equals("CancelPressed")) { // test if cancel key is pressed

balanceAmount = Double.parseDouble(tempValue);

phoneNumber = displayInputMessage("RequestPhoneNumber"); // get telephone number

if (!phoneNumber.equals("CancelPressed")) { // test if cancel key is pressed

Customer newCustomer = new Customer( // define new user object

customerName,

customerNumber,

balanceAmount,

phoneNumber

);

CUSTOMER.add(newCustomer); // add new user object

JLabel newUserLabel = new JLabel( // configure confirmation message

String.format("<html>"

+ "<h2 align='center'>The following customer has been added</h2><br>"

+ "<table>"

+ "<tr>"

+ "<td align='left' width='150'>%s</td>"

+ "<td align='right' width='200'>%s</td>"

+ "</tr>"

+ "<tr>"

+ "<td align='left' width='150'>%s</td>"

+ "<td align='right' width='200'>%s</td>"

+ "</tr>"

+ "<tr>"

+ "<td align='left' width='150'>%s</td>"

+ "<td align='right' width='200'>%.2f</td>"

+ "</tr>"

+ "<tr>"

+ "<td align='left' width='150'>%s</td>"

+ "<td align='right' width='200'>%s</td>"

+ "</tr><br><br>",

"Customer Name",

customerName,

"Customer Number",

customerNumber,

"Account Balance",

balanceAmount,

"Phone Number",

phoneNumber

+ "</table>"

+ "</html>"

)

);

JOptionPane.showMessageDialog(

null,

newUserLabel,

COMPANY\_NAME + " System",

JOptionPane.INFORMATION\_MESSAGE

); // display user creation confirmation message

}

}

}

}

/\*\*

\* Remove customer object from arraylist (delete account)

\*

\* @param customerName Name of customer in object

\*/

private static void deleteAccount(String customerName) {

int customerIndex = Customer.findIndex(CUSTOMER, customerName); // get customer name for deletion

if (customerIndex != -1) { // test if the customer exists

String frameTitle = "User Deletion"; // define user deletion confirmation message

String requestMessage = "Are you certain you want to delete the user\n\n"

+ customerName + "\n\n"

+ "The user will be removed from the system and if the\n"

+ "output file is written all his/her records will be\n"

+ "permanently deleted from the database.\n\n";

boolean deleteUser = displayYesNoOptionFrame(requestMessage, frameTitle);// get uer deletion confirmation

if (deleteUser) { // if deletion confirmed, proceed with deletion

String tempValue = CUSTOMER.get(customerIndex).getCustomerName();

Customer.deleteCustomer(CUSTOMER, customerName); // delete customer

displayMessageFrame( // define customer deletion confirmation message

String.format(

"The user %s has been deleted\n",

tempValue

)

);

}

} else {

displayMessage("CustomerDoesNotExist"); // display user deletion confirmation message

}

}

/\*\*

\* Writes the data in the arraylist to an output file

\*/

private static void writeDatabaseFile() {

File outputFile; // define file object

FileOutputStream fileOutputStream; // define fileoutputstream object

PrintWriter printWriter = null; // define printwriter object

if (DATABASE\_FILENAME.isEmpty()) { // test if the outputfilename variable has a value

showErrorMessage("NoOutputFileSpecified", "writeFile Method"); // if empty, inform user there is nothing to write to file

} else {

if (fileExists(DATABASE\_FILENAME)) { // if not empty, test if file exists

boolean backupOldFile = showYesNoMessage("OutputFileExists"); // if file exists, ask user if a backup is required

if (backupOldFile) {

backupFile(DATABASE\_FILENAME); // if yes, backup file

}

}

try {

outputFile = new File(getNewPath(DATABASE\_FILENAME)); // redefine file object with new file details

fileOutputStream = new FileOutputStream(outputFile); // redefine fileoutputstrem with new outputfile details

printWriter = new PrintWriter(fileOutputStream); // redefine printwriter with new fileoutputstream details

for (int i = 0; i < CUSTOMER.size(); i++) { // for each record in arraylist, write record to file

printWriter.print(((Customer) CUSTOMER.get(i)).getCustomerName() + " ");

printWriter.print(((Customer) CUSTOMER.get(i)).getCustomerNumber() + " ");

printWriter.print(((Customer) CUSTOMER.get(i)).getBalance() + " ");

printWriter.print(((Customer) CUSTOMER.get(i)).getPhoneNumber() + " ");

printWriter.println();

}

} catch (Exception e) { // catch any errors

} finally {

if (printWriter != null) {

printWriter.close(); // close the printwriter

}

}

}

}

/\*\*

\* Generate log files

\*

\* @param fileToWrite Provides a key which is used to determine what type of log file

\* will be created

\* @param errorToLog Provides a string containing details about the error that should

\* be logged

\*/

private static void writeLogFile(String fileToWrite, String errorToLog) {

File outputFile; // define new file object

FileOutputStream fileOutputStream; // define new fileoutputstream object

PrintWriter printWriter = null; // define new printwriter object

String outputFileName = "";

switch (fileToWrite) {

case "ErrorLogFile":

outputFileName = LOG\_FILENAME; // define filename based on logfile requirements

break;

default:

break;

}

if (outputFileName.isEmpty()) {

showErrorMessage("NoOutputFileSpecified", "writeFile Method"); // test if the outputfilename variable has a value

} else {

try {

outputFile = new File(getNewPath(outputFileName)); // redefine outputfile object

if (!fileExists(outputFileName)) {

fileOutputStream = new FileOutputStream(outputFile); // redefine fileoutputstream object for new file creation

printWriter = new PrintWriter(fileOutputStream); // redefine printwriter object for new file creation

printWriter.println(getDateTime() + " " + errorToLog); // write log to file

} else {

fileOutputStream = new FileOutputStream(outputFile, true); // redefine fileoutputstream object to append to existing file

printWriter = new PrintWriter(fileOutputStream); // redefine printwriter object to append to existing file

printWriter.append(getDateTime() + " " + errorToLog); // write to log file

printWriter.println();

}

} catch (Exception e) { // catch any errors

} finally {

if (printWriter != null) {

printWriter.close(); // close the printwriter

}

}

}

}

/\*\*

\* Reads the contents of a data file containing customers and creates

\* an arraylist of customer objects

\*/

private static void readFile() {

File inputFile; // define inputfile object

FileReader fileReader; // define filereader object

BufferedReader bufferedReader = null; // define bufferedreader object

if (INPUT\_FILENAME.isEmpty()) { // test if the inputfilename variable has a value

showErrorMessage("NoInputFileSpecified", "readFile Method");

} else {

if (fileExists(INPUT\_FILENAME)) { // test i the file exists

try {

inputFile = new File(getNewPath(INPUT\_FILENAME)); // redefine inputfile object with input filename

fileReader = new FileReader(inputFile); // redefine filereader object with new inputfile

bufferedReader = new BufferedReader(fileReader); // redefine bufferedreader object with new filereader

String inputString;

Customer readCustomer;

int rowCounter = 0;

while (true) { // while there are entries int he input file, read the values

rowCounter++;

inputString = bufferedReader.readLine();

if (inputString == null) {

break;

}

inputString = inputString.replaceAll("\\s+", " "); // remove any additional spaces from the input file data

String[] tempArray = inputString.split(" "); // split the input string into an array for parsing

try { // configure the input data array

readCustomer = new Customer(

tempArray[0],

tempArray[1],

Double.parseDouble(tempArray[2]),

tempArray[3]

);

if ((Customer.findIndex(CUSTOMER, tempArray[0])) == -1) {

CUSTOMER.add(readCustomer); // if the customer does not exist, read into arraylist

}

} catch (Exception e) {

writeLogFile("ErrorLogFile" // if there is an error, write the error to the log file

, "Read File error: "

+ INPUT\_FILENAME

+ " line : "

+ rowCounter

+ " cause : "

+ e.toString()

);

}

}

bufferedReader.close(); // close the bufferedreader

Customer.nameSort(CUSTOMER); // sort the arraylist

} catch (IOException e) { // catch any errors

System.out.println();

} catch (Exception e) {

System.out.println();

} finally { // if there was an unrecoverable error, close the reader

if (bufferedReader != null) {

try {

bufferedReader.close();

} catch (IOException e) {

}

}

}

} else {

showErrorMessage("FileNotFound", "readFile Method"); // if the file doesn't exist in the specified location

// inform user

}

}

}

/\*\*

\* Display a JOptionPane to the user requesting some type of input

\*

\* @param messageRequest Provides a key which is used to determine what message

\* to display the the user

\*

\* @return Returns the value received as input by the user

\*/

private static String displayInputMessage(String messageRequest) {

String requestText = "";

String inputValue = "";

boolean validInput = false;

switch (messageRequest) {

case "RequestName":

requestText = "Enter Customer's name:\n"; // configure new username message

break;

case "RequestCustomerNumber":

requestText = "Enter Customer's Number, e.g., 11111:\n"; // configure new customer number message

break;

case "RequestPhoneNumber":

requestText = "Enter Customer's phone number:\n"; // configure new customer phone number message

break;

case "RequestDepositAmount":

requestText = "Enter the deposit, e.g., 10000.00:\n"; // configure customer deposit message

break;

case "RequestBalanceAmount":

requestText = "Enter Customer's Balance, e.g., 1000.00:\n"; // configure customer balance message

break;

case "RequestWithdrawalAmount":

requestText = "Enter the withdrawal, e.g., 10.00:\n"; // configure customer withdrawal message

break;

default:

}

do {

try {

inputValue = JOptionPane.showInputDialog( // display the message until valid input is received

null,

requestText,

COMPANY\_NAME + " System",

JOptionPane.QUESTION\_MESSAGE);

} catch (Exception e) {

showExceptionMessage(e, messageRequest);

}

if ((inputValue != null) && (inputValue.length() > 0)) { // if the value has a money amount

if ((messageRequest.equals("RequestDepositAmount"))

|| (messageRequest.equals("RequestBalanceAmount"))

|| (messageRequest.equals("RequestWithdrawalAmount"))) {

try {

Double.parseDouble(inputValue); // test if the input is valid

validInput = true;

} catch (Exception e) {

showExceptionMessage(e, messageRequest);

}

} else {

validInput = true;

}

} else if (inputValue == null) { // if no input received, canel operation

inputValue = "CancelPressed";

validInput = true;

}

} while (!validInput);

return inputValue;

}

/\*\*

\* Constructs a message to the user containing only the option to click OK

\*

\* @param messageRequest Provides a key which is used to determine which message

\* should be displayed to the user.

\*/

private static void displayMessage(String messageRequest) {

switch (messageRequest) {

case "WelcomeMessage": // configure welcome message

displayMessageFrame("Welcome to the ES&L Bank account management program\n"

+ "\n"

+ "This program is used to create accounts, manage balances\n"

+ "and delete accounts if it is required\n");

break;

case "FarewellMessage": // configure farewell message

displayMessageFrame("Thank you for using the ES&L Bank account management program\n");

break;

case "NotANumberMessage": // configure incorrect datatype message

displayMessageFrame("The value you entered is not a number in the specified range, please try again\n");

break;

case "NoDataToDisplay": // configure empty dataset message

displayMessageFrame("There is nothing to display, the database is empty\n");

break;

case "CustomerDoesNotExist": // configure customer not found message

displayMessageFrame("The specified user does not exist\n");

break;

case "DatabaseLoaded":

displayMessageFrame("The database has been loaded"); // configure database load message

break;

case "WrittenToFile":

displayMessageFrame("The database has been written to file"); // configure database write message

break;

case "ShowAllAccounts": // configure show accounts message

JLabel accountListLabel = new JLabel(String.format("<html><table>"

+ "<td align='left' width='300'>%s</td>"

+ "<td align='right' width='200'>%s</td>"

+ "<td align='right' width='200'>%s</td>"

+ "<td align='right' width='200'>%s</td>"

+ "<br><br>",

"Customer Name",

"Customer Number",

"Account Balance",

"Phone Number"

+ CUSTOMER.toString()

.replace(",", "") // remove commas from output

.replace("[", "") // remove left bracket from output

.replace("]", "") // remove right bracket from output

+ "</table></html>"

)

);

accountListLabel.setFont(new Font(Font.MONOSPACED, Font.PLAIN, 16));// set new font and font size for report

JScrollPane scrollPane = new JScrollPane(accountListLabel, // create scrollpane object with scrollbars

JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED,

JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED

);

scrollPane.setSize(new Dimension(200, 200)); // define specified size for window

scrollPane.setPreferredSize(new Dimension(950, 400));

JOptionPane.showMessageDialog( // display records

null,

scrollPane,

COMPANY\_NAME + " System",

JOptionPane.INFORMATION\_MESSAGE

);

break;

default:

}

}

/\*\*

\* Constructs a message to the user requiring a Yes/No answer as input/feedback

\*

\* @param messageRequest Provides a key which is used to determine which

\* question should be posed to the user

\*

\* @return Returns the result (Yes/No) of the question posed to the user

\*/

private static boolean showYesNoMessage(String messageRequest) {

String frameTitle = "";

String requestMessage = "";

switch (messageRequest) {

case "DatasetEmpty": // configure empty dataset message

//frameTitle = "No Records On File";

requestMessage = "You are trying to edit records, but no records exist in the database\n"

+ "would you like to create a new account?\n\n";

break;

case "OutputFileExists": // configure duplicate filename message

//frameTitle = "Output File Already Exists";

requestMessage = "The output file specified already exists, do you want to make a backup of\n"

+ "this file and create a new output file?\n\n";

break;

case "CustomerExists": // configure duplicate customer message

//frameTitle = "Duplicate Customer Name Found";

requestMessage = "The customer name you specified already exists in the database\n"

+ "If you wish to create another customer click yes, alternatively click no to be\n"

+ "returned to the main menu\n\n";

break;

case "CustomerNotFound": // configure customer not found message

//frameTitle = "Customer Not Found";

requestMessage = "The customer name you specifed does not exist\n"

+ "if you wish to create a customer click yes, alternatively click no to be\n"

+ "returned to the main menu\n\n";

break;

case "LoadDefaultDataFile":

//frameTitle = "Load Default Data File"; // configure default data file load message

requestMessage = "You have not loaded any data files, would you like to load the default data file?";

break;

default:

break;

}

return displayYesNoOptionFrame(requestMessage, COMPANY\_NAME + " System"/\*frameTitle\*/);

}

/\*\*

\* Construct a transaction summary to the user

\*

\* @param transaction Provides a key which is used to determine

\* what message should be displayed to the user

\* @param customerIndex Provides the customer index which is used to

\* reference user specific information in the

\* arraylist (arraylist index)

\* @param valueOne Value that needs to be included in the transaction

\* summary

\* @param valueTwo Value that needs to be included in the transaction

\* summary

\*/

private static void displayTransactionSummary(String transaction, int customerIndex, double valueOne, double valueTwo) {

switch (transaction) {

case "makeDeposit": // display deposit summary message

displayMessageFrame(

String.format(

"%s balance after deposit: %50.2f\n" // define output format

+ "%s balance after interest is added: %35.2f\n",

CUSTOMER.get(customerIndex).getCustomerName(),

valueOne,

CUSTOMER.get(customerIndex).getCustomerName(),

valueTwo

)

);

break;

case "makeWithdrawal": // display withdrawal summary message

displayMessageFrame(

String.format(

"%s balance after withdrawal: %50.2f\n" // define output format

+ "%s balance after interest is added: %35.2f\n",

CUSTOMER.get(customerIndex).getCustomerName(),

valueOne,

CUSTOMER.get(customerIndex).getCustomerName(),

valueTwo

)

);

break;

default:

break;

}

}

/\*\*

\* Construct a message to the user indicating that some type of transaction error

\* has occurred

\*

\* @param transactionError Provides a key which is used to determine

\* what message should be displayed to the user

\* @param transactionDetails Provides details about the transaction error

\* that has occurred

\*/

private static void displayTransactionError(String transactionError, String[] transactionDetails) {

switch (transactionError) {

case "NegativeAmountWithdrawal": // display negative withdrawal message

displayMessageFrame(String.format(

"Error: Withdraw amount is invalid.\n\n"

+ "Customer: %50s \n"

+ "Requested: %52.2f \n",

transactionDetails[0],

Double.parseDouble(transactionDetails[1])

));

break;

case "NegativeAmountDeposit": // display negative deposit message

displayMessageFrame(String.format(

"Error: Withdraw amount is invalid.\n\n"

+ "Customer: %50s \n"

+ "Requested: %52.2f \n",

transactionDetails[0],

Double.parseDouble(transactionDetails[1])

));

break;

case "InsufficientFunds": // display insufficient funds message

displayMessageFrame(String.format(

"Error: Insufficient funds.\n\n"

+ "Customer: %50s \n"

+ "Requested: %52.2f \n"

+ "Available: %54.2f \n",

transactionDetails[0],

Double.parseDouble(transactionDetails[1]),

Double.parseDouble(transactionDetails[2])

));

break;

case "NotANumberMessage": // display not a number message

displayMessageFrame("The value you entered is not a number in the specified range, please try again\n");

break;

default:

}

}

/\*\*

\* Construct a general error message to the user

\*

\* @param errorIndex Provides a key which is used to determine

\* what message should be displayed to the user

\* @param tracebackInformation Provides information that can be used to

\* determine the cause of the error

\*/

private static void showErrorMessage(String errorIndex, String tracebackInformation) {

String errorTitle = new String();

String errorMessage = new String();

switch (errorIndex) {

case "NoInputFileSpecified": // define no input file specified message

//errorTitle = "No Input File Specifed";

errorMessage = "Please provide a name for the customer input file";

break;

case "NoOutputFileSpecified": // define no output file specified message

//errorTitle = "No Output File Specified";

errorMessage = "Please provide a name for the customer output file";

break;

case "FileNotFound": // define file not found message

//errorTitle = "Input File Not Found";

errorMessage = "The input file specified in your configuration does not exist\n"

+ "please correct the path to the input file and try again";

break;

default:

}

displayErrorFrame(errorMessage, COMPANY\_NAME + " System" /\*errorTitle\*/); // display error message

}

/\*\*

\* Display a JOptionPane dialog to the user

\*

\* @param messageRequest The message that needs to be displayed

\*/

private static void displayMessageFrame(String messageRequest) {

JOptionPane.showMessageDialog( // display messages

null,

messageRequest,

COMPANY\_NAME + " System",

JOptionPane.INFORMATION\_MESSAGE);

}

/\*\*

\* Display the Yes/No OptionFrame to the user

\*

\* @param messageRequest The message that needs to be displayed

\* @param frameTitle The title of the frame that is displayed

\*

\* @return returnValue Return the result of the question posed

\* to the user

\*/

private static boolean displayYesNoOptionFrame(String messageRequest, String frameTitle) {

boolean returnValue = false;

int yesNo = JOptionPane.showConfirmDialog( // display Jes/No option frame

null,

messageRequest,

frameTitle,

JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE);

if (yesNo == JOptionPane.YES\_OPTION) {

returnValue = true;

}

return returnValue; // return Yes/No optionframe result

}

/\*\*

\* Display the error frame to the user

\*

\* @param errorMessage The message that is displayed to the user

\* @param errorTitle The title of the frame that is displayed to

\* the user

\*/

private static void displayErrorFrame(String errorMessage, String errorTitle) {

JOptionPane jErrorPane = new JOptionPane(errorMessage, JOptionPane.ERROR\_MESSAGE);

JDialog jDialog = jErrorPane.createDialog(errorTitle); // define error message pane

jDialog.setAlwaysOnTop(true); // display error message pane

jDialog.setVisible(true);

}

/\*\*

\* Display a general error message to the user

\*

\* @param e Information about the exception

\* @param tracebackInformation Provides information that can be used to

\* determine the cause of the error

\*/

private static void showExceptionMessage(Exception e, String tracebackInformation) {

String errorMessage = new String();

String errorTitle = new String();

String isolatedMessage = new String( // extract exception message

e.toString().substring(

e.toString().lastIndexOf(

".",

e.toString().indexOf("Exception")) + 1,

e.toString().indexOf(

":",

e.toString().indexOf("Exception")

)

)

);

switch (isolatedMessage) {

case "NumberFormatException": // define numberformatexception message

errorTitle = "Invalid data type";

errorMessage = "You did not enter a valid numeric value, please try again";

break;

default:

}

displayErrorFrame(errorMessage, errorTitle); // display error message

}

/\*\*

\* Set the filename for a specific file IO action

\*

\* @param filename Name of the file

\*

\* @return Instantiated file

\*/

private static File setFile(String filename) {

File file = new File(filename); // create actual file object

return file;

}

/\*\*

\* Set the path for a specific file IO action

\*

\* @param filename Name of the file

\*

\* @return Instantiated path

\*/

private static Path setFilePath(File filename) {

try {

Path filePath = Paths.get(filename.getCanonicalPath()); // create actual path object

return filePath;

} catch (IOException e) {

}

return null;

}

/\*\*

\* Convert the fully qualified file path to a platform independent form

\*

\* @param filePath MS-style file path

\*

\* @return Platform independent file path

\*/

private static String formatFullyQualifiedPath(Path filePath) {

String returnValue = filePath.toString().replace("\\", "/"); // redefine file path to fully quallified path

return returnValue;

}

/\*\*

\* Test to see whether a file exists in a target location

\*

\* @param filename File who's existence is being confirmed

\*

\* @return Return true if exists, and false if does not

\*/

private static boolean fileExists(String filename) {

String newPath = getNewPath(filename);

File testFile = setFile(newPath); // Create a new File object with fully qualified path

return testFile.exists();

}

/\*\*

\* Get the fully qualified path of a file based on the relative path

\*

\* @param filename Name of the file that is being qualified

\*

\* @return formattedPath Qualified file path

\*/

private static String getNewPath(String filename) {

File relativePathFile = setFile(filename); // Create a file object

Path fullyQualifiedPath = setFilePath(relativePathFile); // Get fully qualified path for file

String formattedPath = formatFullyQualifiedPath(fullyQualifiedPath); // Format the fully qualified path for multi platform use

return formattedPath;

}

/\*\*

\* Make a backup of a file

\*

\* @param filename Name of the file that needs to be backed up

\*

\*/

private static void backupFile(String filename) {

Path copySourceFile = setFilePath(setFile(filename)); // define path for source file

Path copyDestinationFile = setFilePath( // define path for destination file

setFile(((setFilePath(

setFile(filename)

)).toString()).substring(

0, (setFilePath(

setFile(filename)

)).toString().lastIndexOf(".")

) + "\_" + getDateTime() + ".txt"

)

);

try {

Files.copy(copySourceFile, copyDestinationFile, StandardCopyOption.REPLACE\_EXISTING);// create a backup of existing file

} catch (IOException e) {

}

}

/\*\*

\* Get the date and time

\*

\* @return Date and time

\*/

private static String getDateTime() {

DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyyMMdd\_HHmmss"); // get the date/time in specific format

LocalDateTime now = LocalDateTime.now();

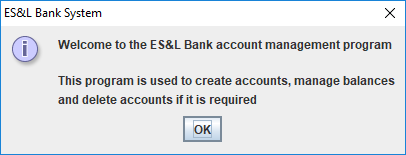
return dateTimeFormatter.format(now); //20161116\_120843

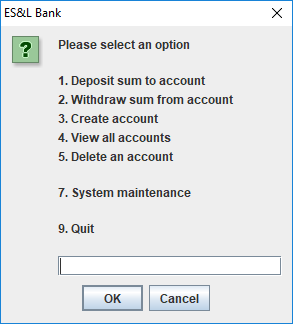
}

}

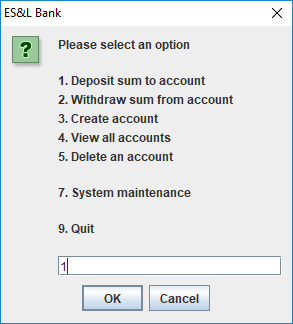
## Sample Input/Output

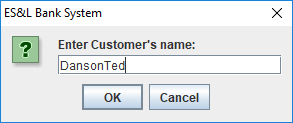
### Program Start and Main Menu

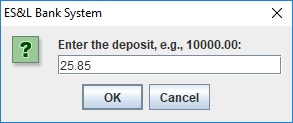


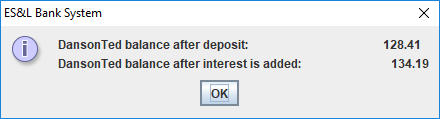


### Option 1: Deposit sum to account



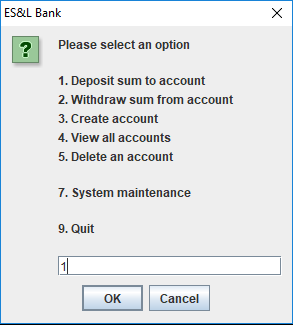


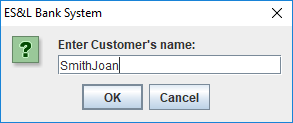


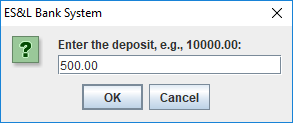


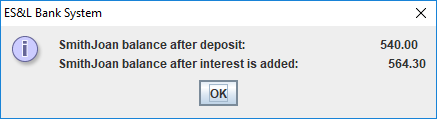
### Options 1 and 2: Deposit sum to account and Withdraw sum from the account

#### Option 1

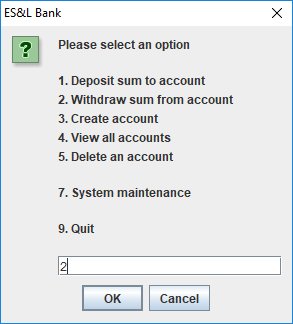


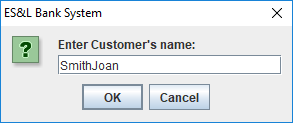


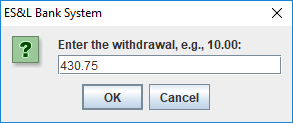


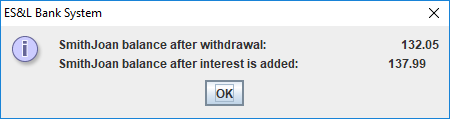


#### Option 2: Withdraw sum from account

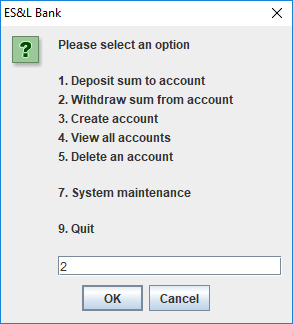


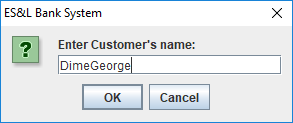


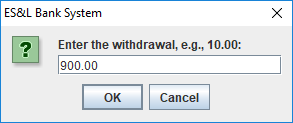


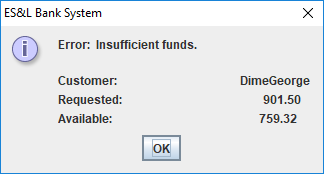


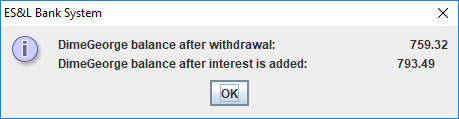
### Option 2: Insufficient funds



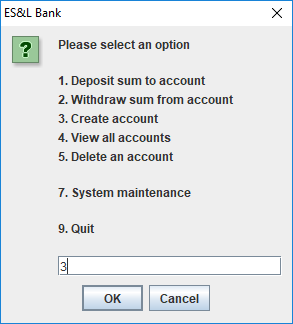


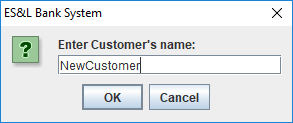


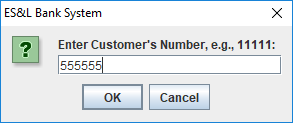


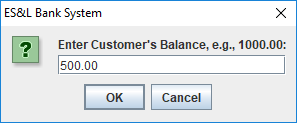


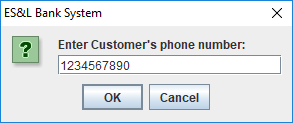
### Option 3: Create and account



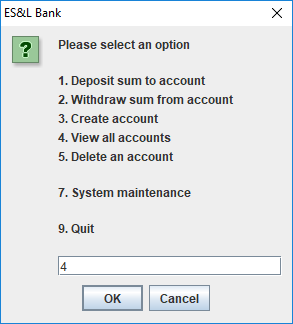


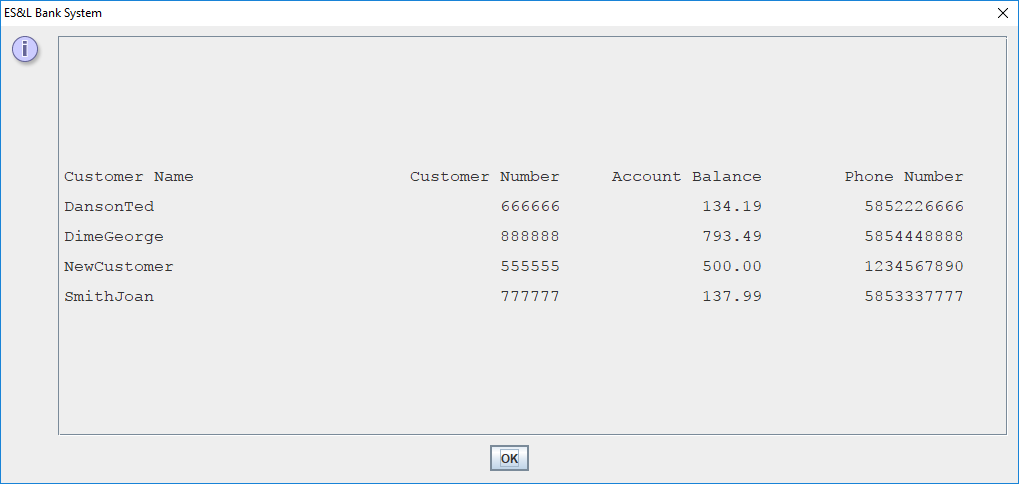




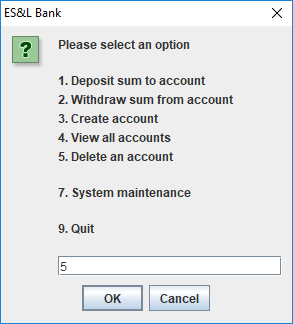


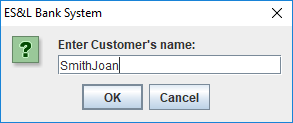
### Option 4: View all accounts

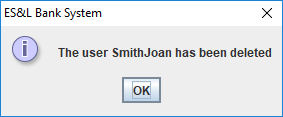




### Option 5: Delete an account



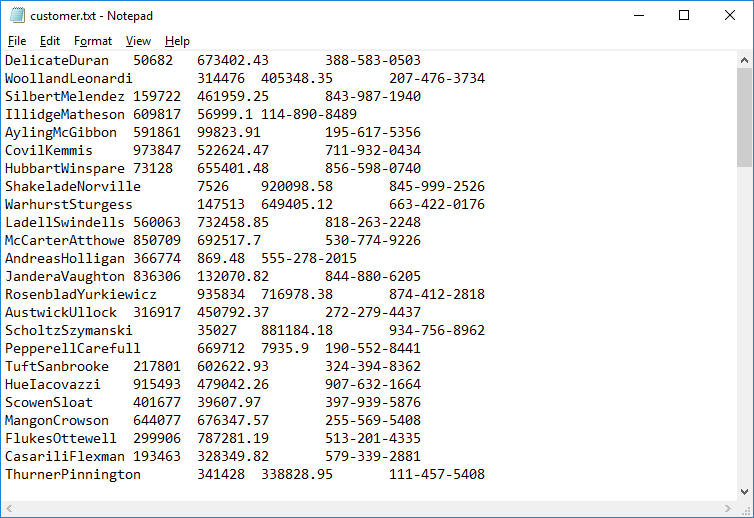




### End-of-File Processing (sort the array and print the Customers in the array)

This action is performed automatically, here is an example

1. Input file



1. The file when viewed using option 4 on the main menu

