Benjamin Kleynhans (M00858194)

Professor: Christine Forde

Project #1 Documentation Package

Customer Class

CSC 225-185 Advanced JAVA Programming

Fall 2017

Table of Contents

[Project Abstract / Overview 2](#_Toc494283469)

[Program Design 3](#_Toc494283470)

[UML Document 3](#_Toc494283471)

[Javadoc 4](#_Toc494283472)

[Pseudocode 11](#_Toc494283473)

[Constructor Definitions 12](#_Toc494283474)

[Method Definitions 12](#_Toc494283475)

[Source Code 16](#_Toc494283476)

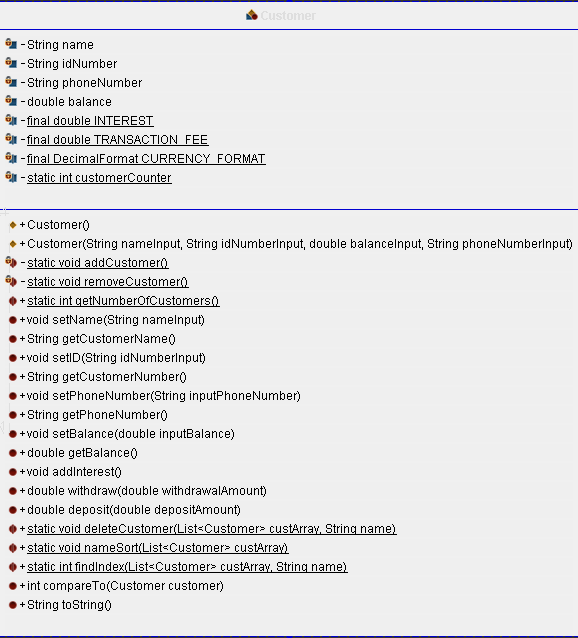
[Sample Input/Output 23](#_Toc494283477)

# Project Abstract / Overview

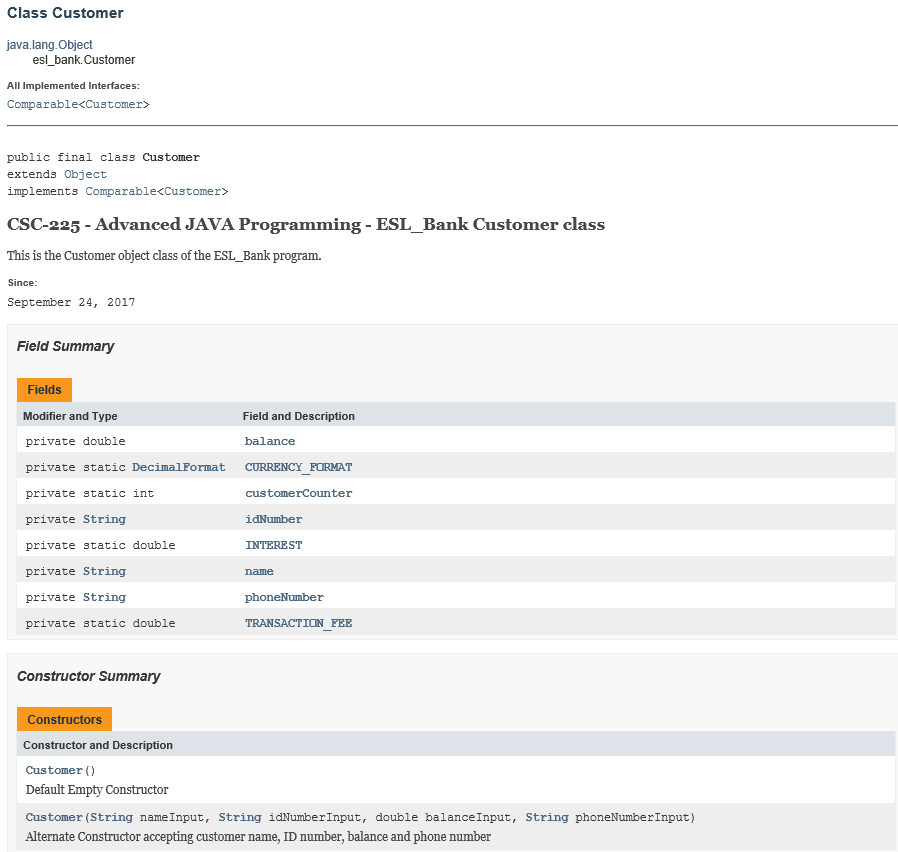
This program is designed to function as a customer account management system at a bank. It provides all the standard account management options of creating a customer, depositing and withdrawing funds from the customer account, deleting a customer and displaying a report of all customers. Whenever an amount is deposited or withdrawn, interest is added to the account and there is a transaction fee for each withdrawal.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 Customer class.

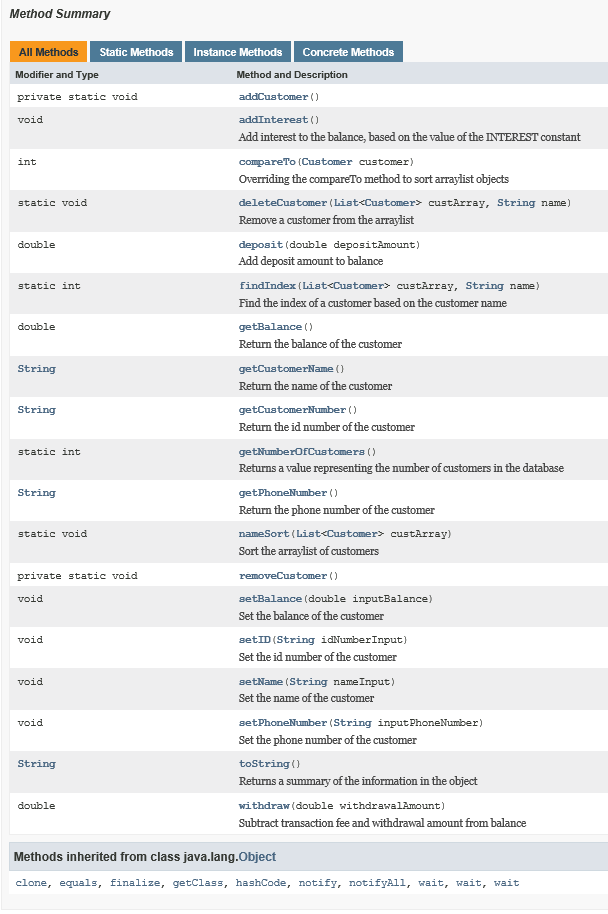
# Program Design

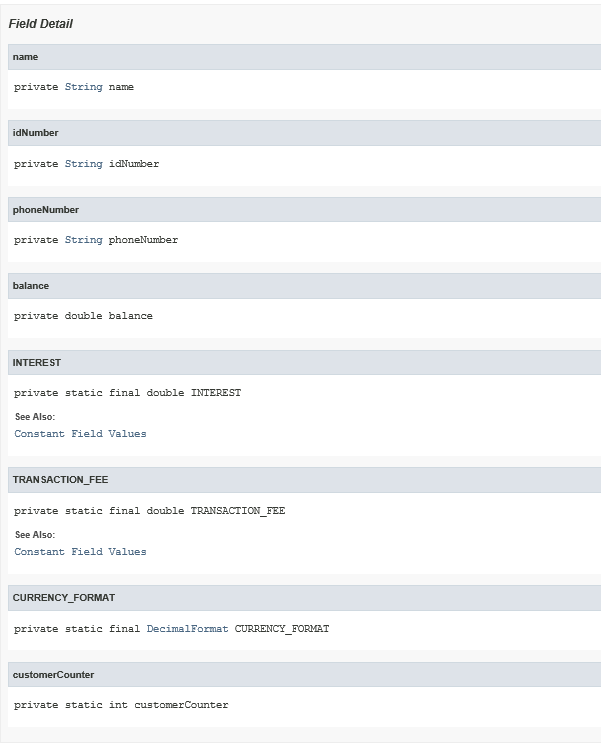
## UML Document

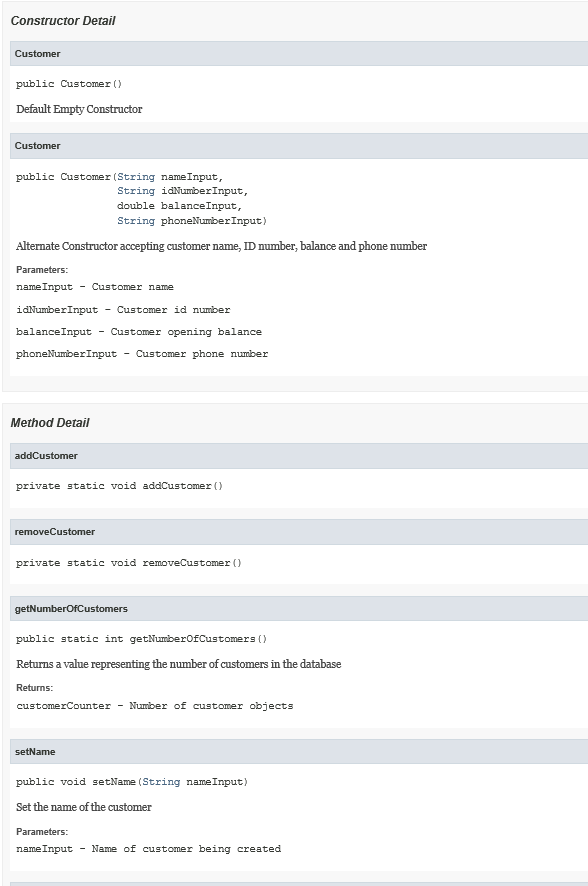


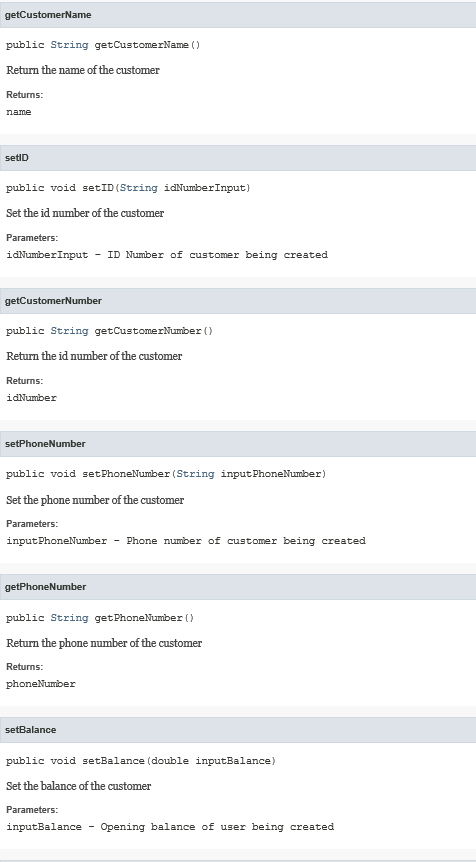
## Javadoc



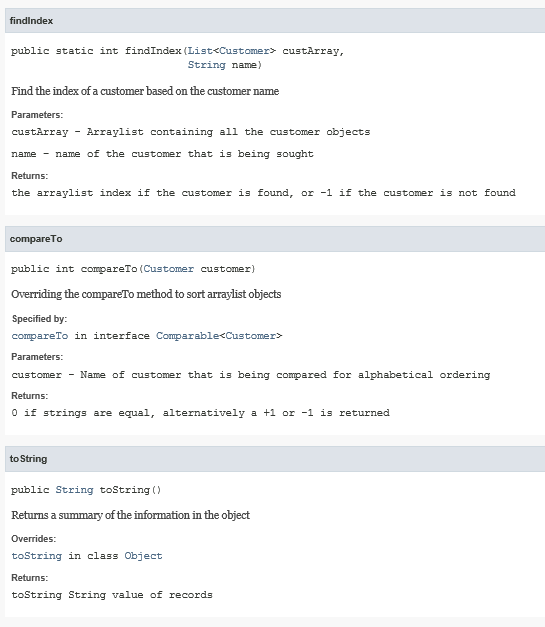












## Pseudocode

BEGIN Customer class

Import libraries

{All variables are declared as private}

Declare variables

object variables

static variables

static constants

Declare Constructors

Default constructor

Secondary constructor

Declare Methods

addCustomer

removeCustomer

getNumberOfCustomers

setName

getCustomerName

setID

getCustomerNumber

setPhoneNumber

getPhoneNumber

setBalance

getBalance

addInterest

withdraw

deposit

deleteCustomer

nameSort

findIndex

conpareTo

toString

END Customer class

### Constructor Definitions

#### Customer (default)

BEGIN Customer

Instantiate customer object using default values

Add one to customer counter

END Customer

#### Customer (secondary)

BEGIN Customer (w, x, y, z)

Instantiate customer object using variables w, x, y, z

Add one to customer counter

END Customer

### Method Definitions

#### addCustomer

BEGIN addCustomer

Add one to the customer counter

END addCustomer

#### removeCustomer

BEGIN removeCustomer

Substract one from the customer counter

END removeCustomer

#### getNumberOfCustomers

BEGIN getNumberOfCustomers

Return number of customer objects

END getNumberOfCustomers

#### setName

BEGIN setName (x)

Set contents of customer name variable to x

END setName

#### getCustomerName

BEGIN getCustomerName

Return contents of customer name variable

END getCustomerName

#### setID

BEGIN setID (x)

Set contents of customer id variable to x

END setID

#### getCustomerNumber

BEGIN getCustomerNumber

Return contents of customer id variable

END getCustomerNumber

#### setPhoneNumber

BEGIN setPhoneNumber (x)

Set contents of customer phone number variable to x

END setPhoneNumber

#### getPhoneNumber

BEGIN getPhoneNumber

Return contents of customer phone number variable

END getPhoneNumber

#### setBalance

BEGIN setBalance (x)

Set contents of customer balance variable to x

END setBalance

#### getBalance

BEGIN getBalance

Return contents of customer balance variable

END getBalance

#### addInterest

BEGIN addInterest

Add interest to customer balance

END addInterest

#### withdraw

BEGIN withdraw (x)

Subtract transaction fee from customer balance

Subtract x from customer balance

Return new customer balance

END withdraw

#### deposit

BEGIN deposit (x)

Add x to customer balance

END deposit

#### deleteCustomer

BEGIN deleteCustomer (x[], y)

Remove customer object y from database x[]

Subtract one from the customer counter

END deleteCustomer

#### nameSort

BEGIN nameSort (x[])

Sort customer database x[]

END nameSort

#### findIndex

BEGIN findIndex (x[], y)

If (x[]) is empty

Return -1

Else (x[]) is not empty

Loop through x[] and compare the iteration to y

If (x[]) equals y

Return the index of x

End-If

End-If

Return -1 if this point is reached

END findIndex

#### compareTo

BEGIN compareTo (x)

Compare x with this object

If x equals this object

return 0

Else

return 1 or -1

END compareTo

#### toString

BEGIN toString

Format a string containing all relevant information of this object

Return the formatted string

END toString

## Source Code

package esl\_bank;

/\*

\* CSC-225 - Advanced JAVA Programming

\* Project 1 - ES&L Bank program

\* Class Description : This is the Customer object class of the ESL\_Bank program.

\* Author : Benjamin Kleynhans

\* Date : September 24, 2017

\* Filename : BankDriver.java

\*/

import java.text.DecimalFormat;

import java.util.\*;

/\*\*

\* <h1>

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

\* </h1>

\* <p>

\* This is the Customer object class of the ESL\_Bank program.

\* </p>

\*

\* @version 1.0

\* @since September 24, 2017

\*/

/\*

\* Start of Customer class

\*/

public final class Customer implements Comparable<Customer> {

private String name = new String();

private String idNumber = new String();

private String phoneNumber = new String();

private double balance = 0;

private static final double INTEREST = 0.045; // interest in percentage divided by 100. ie: (4.5/100)

private static final double TRANSACTION\_FEE = 1.50; // fee paid per transaction in dollar amount

private static final DecimalFormat CURRENCY\_FORMAT = new DecimalFormat("#,##0.00");

private static int customerCounter;

/\*\*

\* Default Empty Constructor

\*/

public Customer() {

this.setName("DefaultUser");

this.setID("000000");

this.setPhoneNumber("000-000-0000");

this.setBalance(0.00);

addCustomer();

}

/\*\*

\* Alternate Constructor accepting customer name, ID number, balance and phone number

\*

\* @param nameInput Customer name

\* @param idNumberInput Customer id number

\* @param balanceInput Customer opening balance

\* @param phoneNumberInput Customer phone number

\*/

public Customer(String nameInput, String idNumberInput, double balanceInput, String phoneNumberInput) {

this.setName(nameInput);

this.setID(idNumberInput);

this.setPhoneNumber(phoneNumberInput);

this.setBalance(balanceInput);

addCustomer();

}

/\*

\* Add one to the customer counter

\*/

private static void addCustomer() {

customerCounter++;

}

/\*

\* Remove one from the customer counter

\*/

private static void removeCustomer() {

customerCounter--;

}

/\*\*

\* Returns a value representing the number of customers in the database

\*

\* @return customerCounter - Number of customer objects

\*/

public static int getNumberOfCustomers() {

return customerCounter;

}

/\*\*

\* Set the name of the customer

\*

\* @param nameInput Name of customer being created

\*/

public void setName(String nameInput) {

this.name = nameInput;

}

/\*\*

\* Return the name of the customer

\*

\* @return name

\*/

public String getCustomerName() {

return this.name;

}

/\*\*

\* Set the id number of the customer

\*

\* @param idNumberInput ID Number of customer being created

\*/

public void setID(String idNumberInput) {

this.idNumber = idNumberInput;

}

/\*\*

\* Return the id number of the customer

\*

\* @return idNumber

\*/

public String getCustomerNumber() {

return this.idNumber;

}

/\*\*

\* Set the phone number of the customer

\*

\* @param inputPhoneNumber Phone number of customer being created

\*/

public void setPhoneNumber(String inputPhoneNumber) {

this.phoneNumber = inputPhoneNumber;

}

/\*\*

\* Return the phone number of the customer

\*

\* @return phoneNumber

\*/

public String getPhoneNumber() {

return this.phoneNumber;

}

/\*\*

\* Set the balance of the customer

\*

\* @param inputBalance Opening balance of user being created

\*/

public void setBalance(double inputBalance) {

this.balance = inputBalance;

}

/\*\*

\* Return the balance of the customer

\*

\* @return balance

\*/

public double getBalance() {

return this.balance;

}

/\*\*

\* Add interest to the balance, based on the value of the INTEREST constant

\*/

public void addInterest() {

this.balance = (this.balance \* (1 + INTEREST)); // Add interest to balance

}

/\*\*

\* Subtract transaction fee and withdrawal amount from balance

\*

\* @param withdrawalAmount Amount of money to be withdrawn from account

\* @return balance Balance of account after money is withdrawn

\*/

public double withdraw(double withdrawalAmount) {

this.balance -= TRANSACTION\_FEE; // Subtract transaction fee

this.balance -= withdrawalAmount; // Subtract withdraw amount

return this.balance;

}

/\*\*

\* Add deposit amount to balance

\*

\* @param depositAmount Amount to be deposited into account

\* @return Balance of account after deposit

\*/

public double deposit(double depositAmount) {

this.balance += depositAmount; // Add deposit to balance

return this.balance;

}

/\*\*

\* Remove a customer from the arraylist

\*

\* @param custArray Arraylist containing all the customer objects

\* @param name name of the customer that needs to be deleted

\*/

public static void deleteCustomer(List<Customer> custArray, String name) {

custArray.remove(findIndex(custArray, name)); // Remove customer from arraylist

removeCustomer();

}

/\*\*

\* Sort the arraylist of customers

\*

\* @param custArray Arraylist containing all the customer objects

\*/

public static void nameSort(List<Customer> custArray) {

Collections.sort(custArray); // Sort arraylist

}

/\*\*

\* Find the index of a customer based on the customer name

\*

\* @param custArray Arraylist containing all the customer objects

\* @param name name of the customer that is being sought

\*

\* @return the arraylist index if the customer is found, or -1 if the customer is not found

\*/

public static int findIndex(List<Customer> custArray, String name) {

if (custArray.isEmpty()) {

return -1; // Return -1 if the arraylist is empty

} else {

for (int i = 0; i < custArray.size(); i++) {

if (custArray.get(i).getCustomerName().equals(name)) {

return i; // Return index of customer if found

}

}

}

return -1;

}

/\*\* Overriding the compareTo method to sort arraylist objects

\*

\* @param customer Name of customer that is being compared for alphabetical ordering

\* @return 0 if strings are equal, alternatively a +1 or -1 is returned

\*/

@Override

public int compareTo(Customer customer) {

return (this.getCustomerName()).compareTo(customer.getCustomerName()); // Return resutl of customer name comparison

}

/\*\*

\* Returns a summary of the information in the object

\*

\* @return toString String value of records

\*/

@Override

public String toString() {

return String.format( // Display contents of object using toString method

"<html><tr>"

+ "<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>"

+ "</tr><br>",

this.getCustomerName(),

this.getCustomerNumber(),

CURRENCY\_FORMAT.format(this.getBalance()),

this.getPhoneNumber(),

"</html>"

);

}

}

## Sample Input/Output

This class is an object class and does not have a front-end