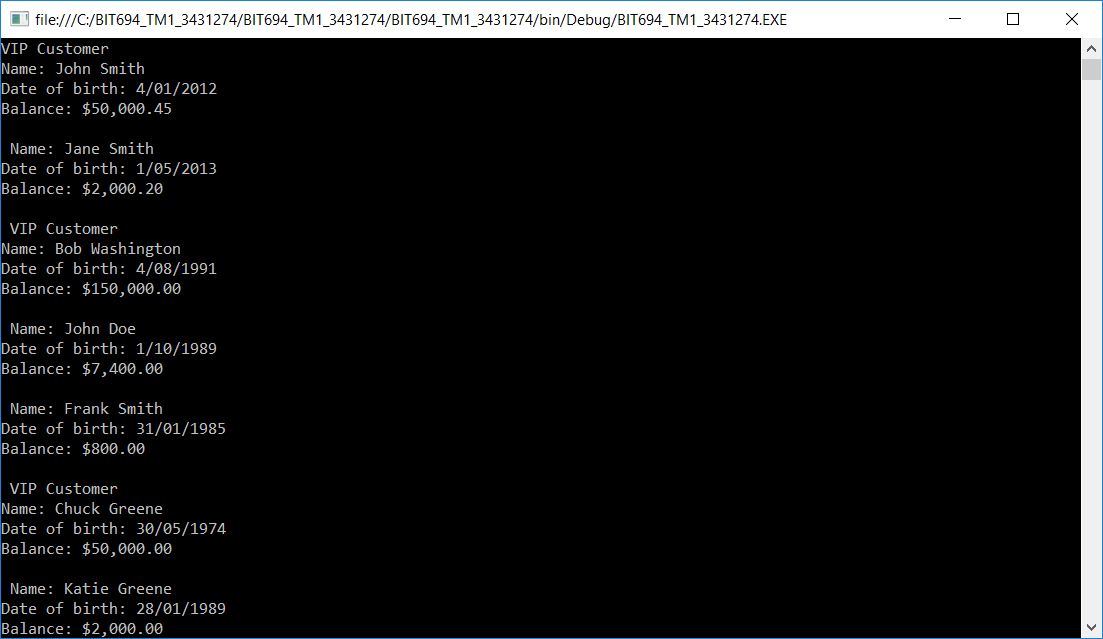
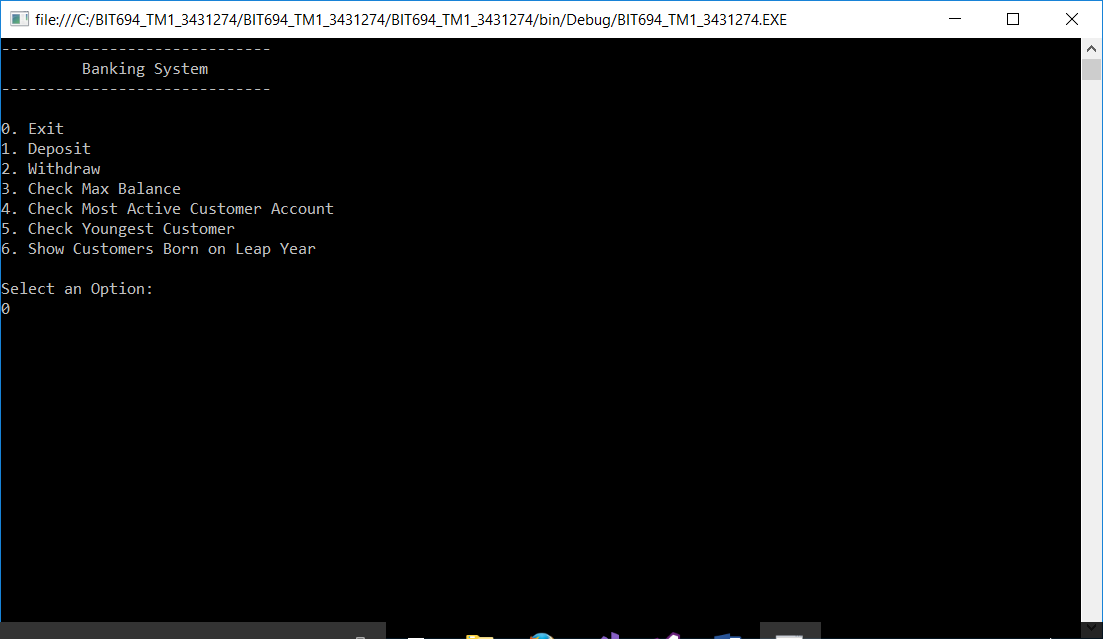
|  |
| --- |
| Assignment 1 (TMA1) |
| BIT694 Designing Applications with C#  Due Date: August 6th 2017 |
| Penelope Ann Williamson  3431274 |

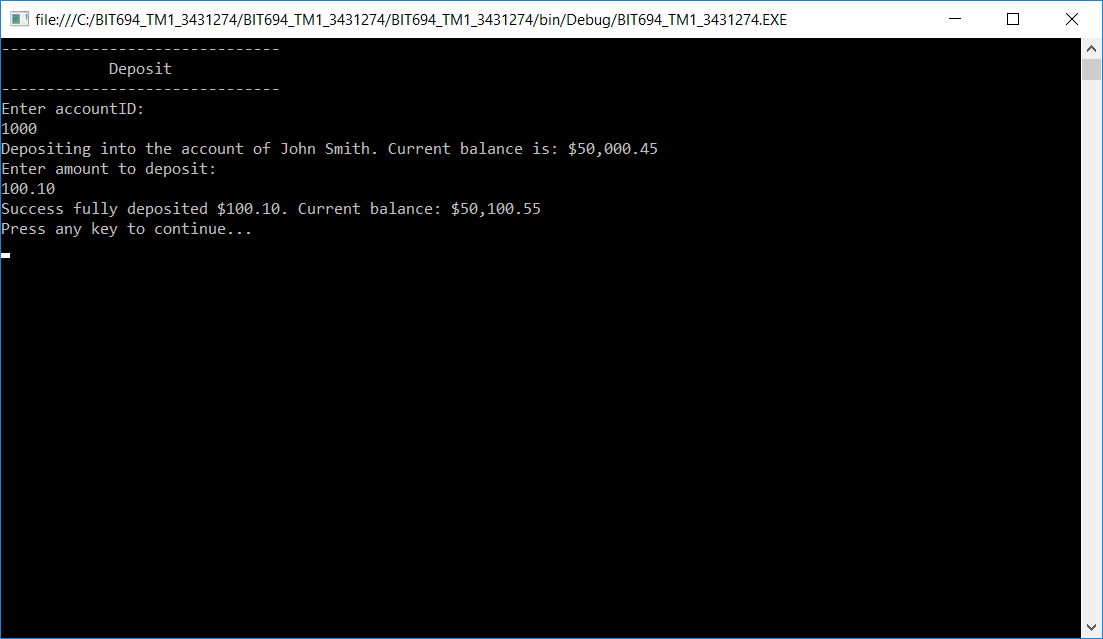
|  |
| --- |
|  |

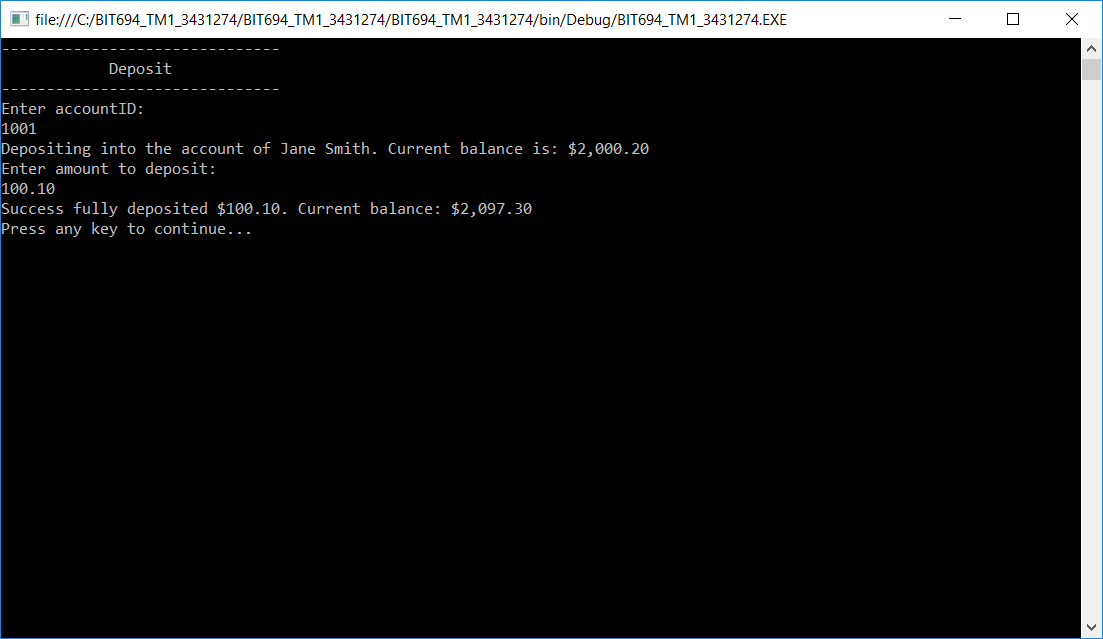
## **Part 1: Software Development.**

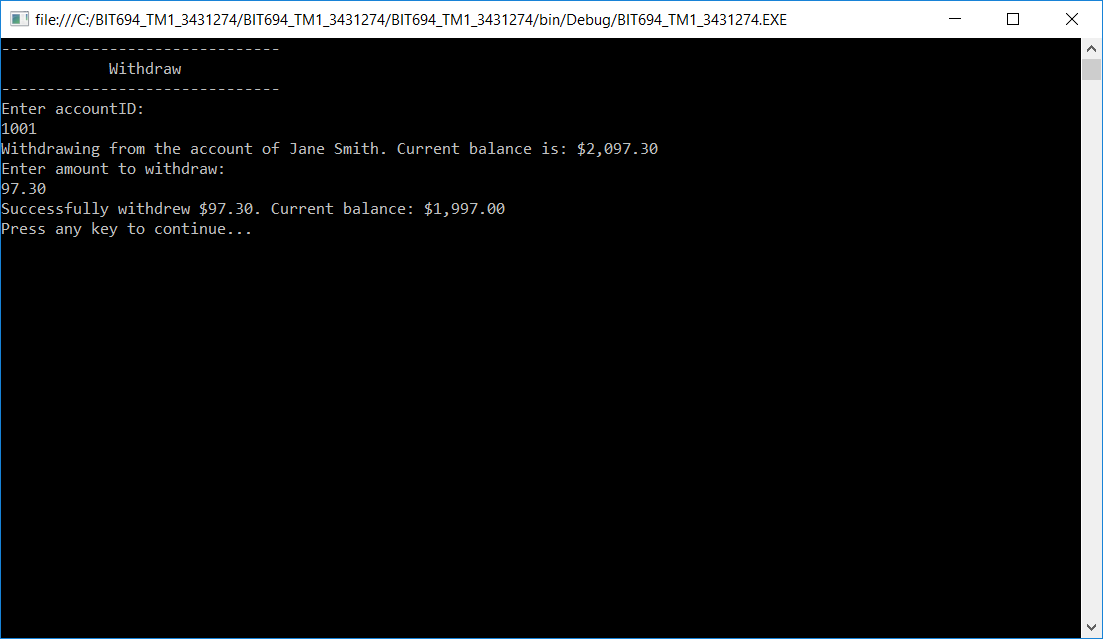
**a)**

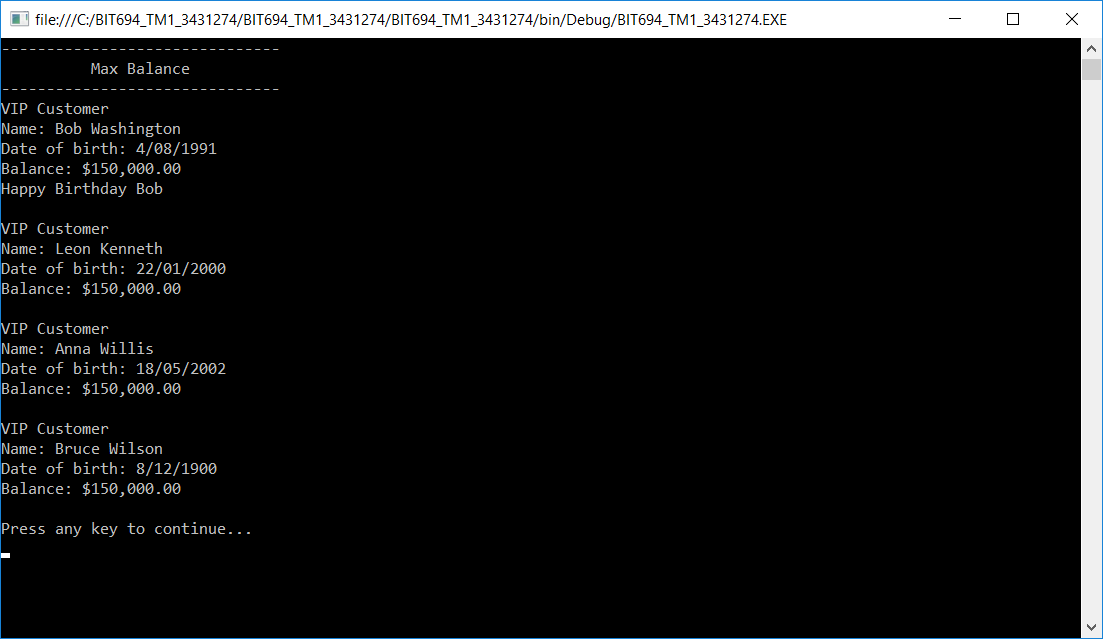
Figure 1. Screen shot of the data populating the Customer array, custData. Showing the basic information for each Customer and VIPCustomer object, as it is read into the array abject.

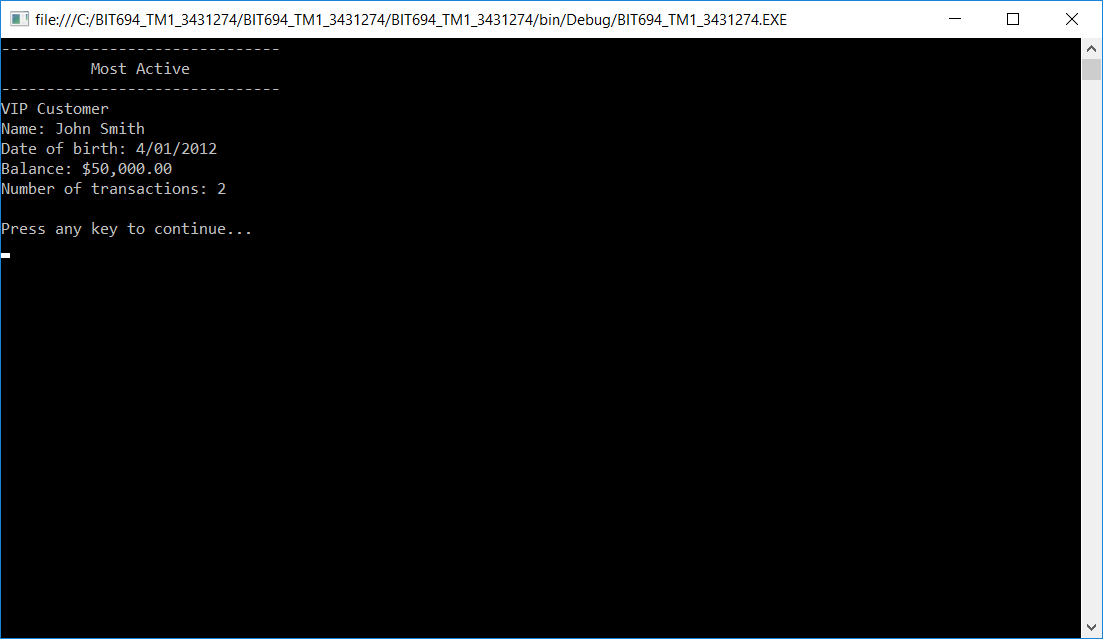
Figure 2. Screen Shot of the menu system

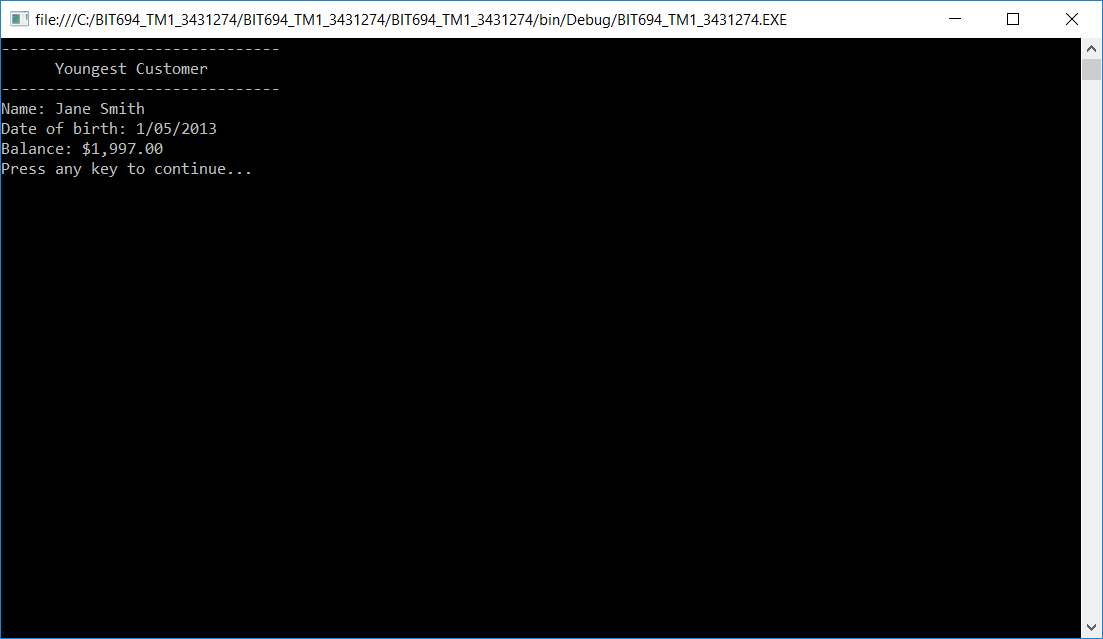
Figure 3. Screen shot of deposit method for VIP customer.

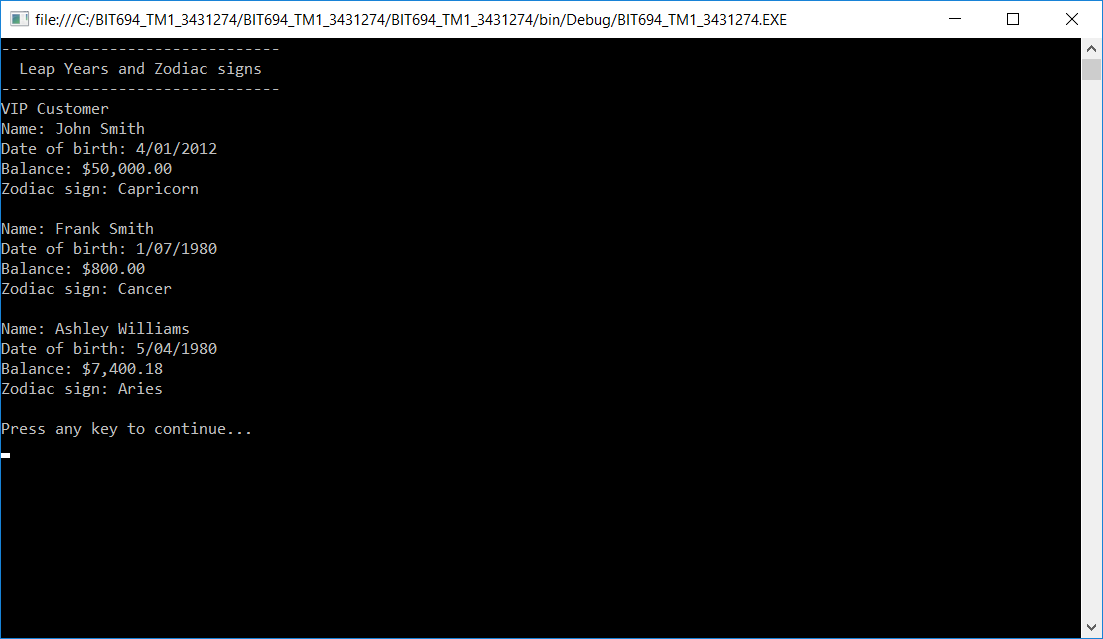
Figure 4. Screen shot of Deposit method for Customer.

Figure 5. Screen shot of withdraw method for Customer.

Figure 6. Screen shot of Maximum Balance

Figure 7. Screen shot of Most Active Customer

Figure 8. Screen shot of Youngest Customer.

Figure 9. Screen shot of customers born in a leap year.

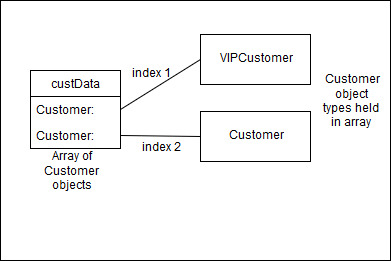
**b)**

The purpose of this program is to create a basic banking application that reads the data it uses from a text file into an object array. The object array contains a mix of two customer types, which allows the three polymorphed methods to be called, based on the customer type held at the appropriate Customer object array (named custData) index position. The array is passed into the methods, and classes so that it can be iterated over, read and updated.

The requirements specified 6 functions (deposit, withdraw, most active, maximum balance, youngest customer and customers born in leap year) with selection made in a menu system. This was achieved by writing a Menu class, with a switch statement menu method, that calls the appropriate Customer class function, and then redisplays the menu after the method has finished running. This continues until the user chooses to exit the program.

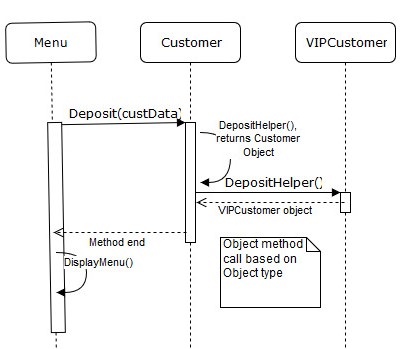
(word count 148)

Inheritance and polymorphism are demonstrated by using a Customer base class and a VIPCustomer subclass, that inherits most of its class variables and methods from the Customer Class. The array, custData, use the rule of inheritance that allows a subclass type to be used as a base class type, to populate a Customer object array with a mix of Customer objects and VIPCustomer objects.

 Figure 10. Object diagram of the first two Customer objects in custData array, showing two different customer object types.

There are three polymorphed, virtual methods in Customer class that are overridden in VIPClass, with the same name and method signatures. The method to be called is dependent on the type of customer that the array, custData, holds for that customer. As VIPCustomer class inherits the base methods, polymorphed methods can be called from the Customer class methods. The methods are:

* DisplayInfo(), which displays a customer’s basic information, including VIP Customer for VIPCustomer objects.
* WithdrawHelper() and DepositHelper(), where the VIPCustomer methods charge no fees and allow overdrawn accounts.

Figure 11. Sequence diagram showing deposit method being called by menu, then calling DepositHelper() method, by object type.

(Word count 152)

**Program Class code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Threading.Tasks;

namespace BIT694\_TM1\_3431274

{

class Program

{

/\*

\* Reads data lines from a text file, split at ',' into

\* an array line[]. Then sets them as

\* either a Customer or VIPCustomer object, based on

\* length of line array. Then adds them to

\* an array of Customer objects. Both types of objects can be held,

\* as VIPCustomer is a subclass of Customer.

\*/

static void ReadingInputFile(Customer[] custData)

{

try

{

TextReader tr = new StreamReader("C:/BIT694\_TM1\_3431274/input\_assignment\_1.txt");

//reads data from file in location in StreamReaders signature.

for (int i = 0; i < custData.Length; i++)//iterates custData length number of times.

{

String currentLine;//declares a String variable

String[] line;//declares a string array

int tranCount = 0;//declares and sets an int variable

currentLine = tr.ReadLine();//sets a String variable to a line of text from StreamReaders file.

line = currentLine.Split(',');//Splits a string, and adds each variable as a separate array item.

if(line.Length > 5)//fires if line array contains the sixth field, VIP.

{

line[5] = "VIP Customer";//changes VIP to VIP Customer

VIPCustomer t = new VIPCustomer(line[0], line[1], line[2], line[3], double.Parse(line[4]), line[5], tranCount);

//Sets a new VIPCustomer object to values from line array, and variable tranCount.

custData[i] = t;//Adds the VIPCustomer object to the custData array.

}

if(line.Length < 6)//fires if array does not contain the sixth field.

{

Customer s = new Customer(line[0], line[1], line[2], line[3], double.Parse(line[4]), tranCount);

//sets a new customer object to values from line array, and variable tranCount.

custData[i] = s;//Adds the Customer object to the custData array.

}

}

tr.Close();//closes the TextReader.

}

catch (FileNotFoundException exA)//StreamReader error

{

Console.WriteLine("" + exA);

}

catch(FormatException exB)//Parse error

{

Console.WriteLine(exB);

}

catch(ArgumentNullException exC)//ReadLine and StreamReader error

{

Console.WriteLine(exC);

}

catch(ArgumentOutOfRangeException exD)//ReadLine and StreamReader error

{

Console.WriteLine(exD);

}

catch(IOException exE)//ReadLine and StreamReader error

{

Console.WriteLine(exE);

}

}

static void Main(string[] args)

{

Customer[] custData = new Customer[20];//declares and initialises the array.

ReadingInputFile(custData);//calls the method to read the text file into the array

Menu menu = new Menu(custData);//creates a Menu object

menu.DisplayMenu(); //calls the Menu method, DisplayMenu().

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

}

}

**Menu Class Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BIT694\_TM1\_3431274

{

class Menu

{

Customer[] custData;//allows custData array to be used by methods in class.

//Constructor for Menu, takes custData as its signature, so it can be passed in menu calls.

public Menu(Customer[] custData)

{

this.custData = custData;//sets signatures custData to equal custData.

}

//Constructor with no argument to call Customer class methods.

Customer r = new Customer();

/\*

\* Menu method that displays a list of options,

\* then calls the option entered by user.

\* Uses switch statements in a while loop to achieve this.

\*/

public void DisplayMenu()

{//menu that appears on Console screen.

Console.WriteLine("------------------------------");

Console.WriteLine(" Banking System ");

Console.WriteLine("------------------------------");

Console.WriteLine(" ");

Console.WriteLine("0. Exit");

Console.WriteLine("1. Deposit");

Console.WriteLine("2. Withdraw");

Console.WriteLine("3. Check Max Balance");

Console.WriteLine("4. Check Most Active Customer Account");

Console.WriteLine("5. Check Youngest Customer");

Console.WriteLine("6. Show Customers Born on Leap Year");

Console.WriteLine(" ");

Console.WriteLine("Select an Option: ");

String option = "0";//declares and initialises option to 0.

option = Console.ReadLine();//sets option to String value entered by user.

while (option != "0")//fires so long as exit condition is not met.

{

switch(option)//used instead of if-else loops.

{

case "0": //exit method.

Console.WriteLine("Press any key to exit");

Console.ReadKey();

break;//exits the switch statement

case "1"://specifies the pattern to be matched.

r.Deposit(custData);//Calls the Customer method Deposit().

Console.Clear();//Clears the console screen of menu.

DisplayMenu();//displays the menu again, once Deposit method has finished.

break;

case "2":

r.Withdraw(custData);//Calls the Customer method Withdraw().

Console.Clear();

DisplayMenu();

break;

case "3":

r.HighestBal(custData);//Calls the Customer method HighestBal().

Console.Clear();

DisplayMenu();

break;

case "4":

r.MostActive(custData);//Calls the Customer method MostActive().

Console.Clear();

DisplayMenu();

break;

case "5":

r. YoungestCustomer(custData);//Calls the Customer method YoungestCustomer().

Console.Clear();

DisplayMenu();

break;

case "6":

r.LeapYear(custData);//Calls the Customer method LeapYear

Console.Clear();

DisplayMenu();

break;

default://fires if no case patterns are matched.

{

Console.WriteLine("Please enter a number between 0 and 6. Press any key to continue...");

Console.ReadKey();

Console.Clear();

DisplayMenu();//Calls itself, to recover from error.

break;

}

}

return;//ends while loop.

}

}

}

}

**Customer Class code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Threading.Tasks;

namespace BIT694\_TM1\_3431274

{

class Customer

{

//protected so they can be inherited by VIPCustomer Class

protected String firstName;//Declares variable to hold Customer and VIPCustomer firstName.

protected String lastName;//Declares variable to hold Customer and VIPCustomer LastName.

protected String dob;//Declares variable to hold Customer and VIPCustomer date of birth (dob).

protected String accNum;//Declares variable to hold Customer and VIPCustomer account number (accNum).

protected Double balance;//Declares variable to hold Customer and VIPCustomer balance.

protected int tranCount;//Declares variable to hold Customer and VIPCustomer transaction count (tranCount). Set to zero in Class Program method, readingInputFile().

private double fee = 3.00; //Declares and intailises variable for fee, so it only needs to be changes in one place.

//Private as it is only used in class.

Customer[] custData;//Allows Class methods to call and use array.

//Contructor for Customer that takes in firstName, lastName, dob, accNum, balance and tranCount in its signature.

public Customer(String firstName, String lastName, String dob, String accNum, double balance, int tranCount)

{

this.firstName = firstName;//sets firstName value in signature to firstName.

this.lastName = lastName;//sets lastName value in signature to LastName.

this.dob = dob;//sets dob value in signature to dob.

this.accNum = accNum;//sets accNum value in signature to accNum.

this.balance = balance;//sets balance value in signature to balance.

this.tranCount = tranCount;//sets tranCount value in signature to tranCount.

}

//A blank constructor, takes no values in, in its signature.

public Customer() { }

//Constructor that takes custData array in as its signature.

public Customer(Customer[] custData)

{

this.custData = custData;//sets custData value in signature to CustData.

}

//Allows firstName variable to be gotten and set.

public String AccessFirstName

{

set { this.firstName = value; }

get { return this.firstName; }

}

//Allows lastName variable to be gotten and set.

public String AccessLastName

{

set { this.lastName = value; }

get { return this.lastName; }

}

//Allows dob variable to be gotten and set.

public String AccessDob

{

set { this.dob = value; }

get { return this.dob; }

}

//Allows accNum variable to be gotten and set.

public String AccessAccNum

{

set { this.accNum = value; }

get { return this.accNum; }

}

//Allows balance variable to be gotten and set.

public double AccessBalance

{

set { this.balance = value; }

get { return this.balance; }

}

//Allows tranCount variable to be gotten and set.

public int AccessTranCount

{

set { this.tranCount = value; }

get { return this.tranCount; }

}

/\*A helper method that displays a customer’s basic information.

\* try catch is not being used for convert

\* as the customer constructor that calls it

\* requires dob to be in String format. This is caught

\* in the readingInputFile function in Main class.

\*/

protected virtual void DisplayInfo()//protected is the lowest access level that can be used for a virtual method.

{

DateTime aDob = Convert.ToDateTime(dob);

String bDob = aDob.ToShortDateString();

Console.WriteLine("Name: " + firstName + " " + lastName);

Console.WriteLine("Date of birth: " +bDob);

Console.WriteLine("Balance: " + balance.ToString("C"));

}

/\*

\* Is called from Menu Class, by menu selection number 6.

\* Public so it can be called from Menu Class.

\* Takes in Customer[] custData array in method signature, so the

\* array can be used in the method.

\* Displays a list of all customers born in a leap year,

\* their basic information, customer type and star sign.

\* Calls private method IsLeapYear() to determine whether or not

\* birth year is a leap year, and uses returned Boolean to trigger

\* the zodiac loop and displaying of basic information.

\*/

public void LeapYear(Customer[] custData)

{

Console.Clear();//Gives a console screen that only contains this methods output

//header for Console page.

Console.WriteLine("-------------------------------");

Console.WriteLine(" Leap Years and Zodiac signs ");

Console.WriteLine("-------------------------------");

try

{

for (int i = 0; i < custData.Length; i++)//iterates through custData array.

{

Customer customer = custData[i];//Declares and sets a customer object to the custData object. Preserves whether or not they are VIP customers

this.firstName = customer.firstName;//sets first name variable to custData's value.

this.lastName = customer.lastName;//sets last name variable to custData's value.

this.dob = customer.dob;//sets dob variable to custData's value.

this.accNum = customer.accNum;//sets accNum variable to custData's value.

this.balance = customer.balance;//sets balance variable to custData's value.

this.tranCount = customer.tranCount;//sets tranCount variable to custData's value.

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

//Declares and initialises a new Customer object that uses the same values as CustData[i]. This is used to call isLeapYear method.

if (s.IsLeapYear())//triggered by Boolean returned from IsLeapYear() method.

{

String[] year = dob.Split('-');//splits the String variable dob into its three date components, at the - character.

String zodiac = null;//Declares and initialises a String variable to hold a zodiac name. Set to null, so it holds no prior value.

int day = int.Parse(year[0]);//Declares and initialises an int variable by converting the day String into an int value.

int month = int.Parse(year[1]);//Declares and initialises an int variable by converting the month String into an int value.

//year is not needed for this method.

//if loops using day and month int variables to set the value of zodiac variable.

if ((month == 1) && (day < 20) || (month == 12) && (day > 21))//fires for dates December 22 to Jan 19

{

zodiac = "Capricorn";//sets zodiac variable to String value "Capricorn"

}

if ((month == 1) && (day > 19) || (month == 2) && (day < 19))//fires for dates Jan 20 to Feb 18

{

zodiac = "Aquarius";

}

if ((month == 2) && (day > 18) || (month == 3) && (day < 21))//fires for dates Feb 19 to Mar 20

{

zodiac = "Pisces";

}

if ((month == 3) && (day > 20) || (month == 4) && (day < 20))//fires for dates Mar 21 to April 19

{

zodiac = "Aries";

}

if ((month == 4) && (day > 19) || (month == 5) && (day < 21))//fires for dates April 20 to May 20

{

zodiac = "Taurus";

}

if ((month == 5) && (day > 20) || (month == 6) && (day < 21))//fires for dates May 21 to June 20

{

zodiac = "Gemini";

}

if ((month == 6) && (day > 20) || (month == 7) && (day < 23))//fires for dates June 21 to July 22

{

zodiac = "Cancer";

}

if ((month == 7) && (day > 22) || (month == 8) && (day < 23))//fires for dates July 23 to Aug 22

{

zodiac = "Leo";

}

if ((month == 8) && (day > 22) || (month == 9) && (day < 23))//fires for dates Aug 23 to Sep 22

{

zodiac = "Virgo";

}

if ((month == 9) && (day > 20) || (month == 10) && (day < 21))//fires for dates Sep 23 to Oct 22

{

zodiac = "Libra";

}

if ((month == 10) && (day > 22) || (month == 11) && (day < 22))//fires for dates Oct 23 to Nov 21

{

zodiac = "Scorpio";

}

if ((month == 11) && (day > 21) || (month == 12) && (day < 22))//fires for dates Nov 22 to Dec 21

{

zodiac = "Sagittarius";

}

custData[i].DisplayInfo();//uses the class specific DisplayInfo() method.

Console.WriteLine("Zodiac sign: " + zodiac);//uses the zodiac value set in the if loops.

Console.WriteLine(" ");//gives a blank line between customers on the console for readability.

}

}

}

catch(FormatException ex)//catches Parse errors.

{

Console.WriteLine(ex);

}

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

/\*

\* Helper method called by LeapYear() method.

\* Determines whether or not the customer object

\* that is used to call it, is born in a leap year.

\* Returns Boolean which triggers zodiac loop

\* in LeapYear method.

\* Only used within Customer class, hence private.

\*/

private Boolean IsLeapYear()

{

bool isLeap = false;

try

{

String[] year = dob.Split('-');

int dobYear = int.Parse(year[2]);//Declares and initilises an int value by converting the String year value into an int value.

if ((dobYear % 4 == 0) && (dobYear % 100 != 0))//formula for calculating leap year, divisible by 4, but not by 100.

{

isLeap = true;

}

else

{

isLeap = false;

}

}

catch(FormatException ex)//Parse error

{

Console.WriteLine(ex);

}

return isLeap;// returns true or false to trigger if loop.

}

/\*

\* Public method called from Menu Class, by

\* menu selection number 1, hence public access modifier.

\* Takes in Customer[] custData array in method signature

\* so array can be used within the method.

\* Calls DepositHelper() method in Customer class for Customer objects,

\* and in VIPCustomer class for VIPCustomer objects.

\* Iterates through array for account number, and calls that custData[i]

\* objects deposit helper method.

\* Then updates the balance value of the array with the new balance, and

\* updated transaction count (tranCount).

\*/

public void Deposit(Customer[] custData)

{

Console.Clear();//Gives a console screen that only contains output from this and depositHelper methods.

bool contains = false;//triggers an error message if account number not found.

//Console screen header.

Console.WriteLine("-------------------------------");

Console.WriteLine(" Deposit ");

Console.WriteLine("-------------------------------");

try

{

Console.WriteLine("Enter accountID: ");

String accountID = Console.ReadLine();//Declares a String that takes the value entered in response to the previous line.

if (accountID == null)//checks that a value has been entered.

{

Console.WriteLine("Please enter an account ID");

}

for (int i = 0; i < custData.Length; i++)//iterates through array looking for accNum matching accountID value, custData length number of times.

{

Customer customer = custData[i];//see comments in LeapYear method.

this.firstName = customer.firstName;

this.lastName = customer.lastName;

this.dob = customer.dob;

this.accNum = customer.accNum;

this.balance = customer.balance;

this.tranCount = customer.tranCount;

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

if (accountID == accNum)//fires when custData[i] accNum value = entered accountID value.

{

custData[i] = custData[i].DepositHelper();

contains = true;//sets Boolean for account number found to true.

}

}

}

catch(ArgumentOutOfRangeException exDep)//ReadLine error

{

Console.WriteLine(exDep);//This is not recovered from as it is a read error. Input errors are handled in method.

}

if (contains == false)//triggers error message if account number not found.

{

Console.WriteLine("Account ID invalid, press any key to try again");

Console.ReadKey();

Deposit(custData);//Recovers from error by recalling itself.

}

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

/\*

\* Helper method that handles class specific deposit

\* method. Charges a fee that is set in class variables,

\* for each deposit transaction.

\* Takes in and uses the values for the custData[i]

\* object that was used to call it.

\* Returns a Customer object to Customer Class's Deposit() method.

\* Protected as it is a virtual method, that is only used within class

\* and overridden in subclass.

\*/

protected virtual Customer DepositHelper()

{

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

//Customer object set to the calling custData[i] objects values.

try

{

Console.WriteLine("Depositing into the account of " + firstName + " " + lastName + ". Current balance is: " + balance.ToString("C"));

//displays calling Customer objects, first name, last name and current balance in $'s.

Console.WriteLine("Enter amount to deposit: ");

double amount = double.Parse(Console.ReadLine());

//Declares a double variable and initialises it by converting String amount entered in response to previous lines code into a double value.

if (amount < 0) // handles operator error of entering a negative number.

{

Console.WriteLine("Please enter a non negative amount. Press any key to continue");

Console.ReadKey();

s.DepositHelper();//recovers from error by calling itself

}

else //allows the program to continue if number is non-negative, and after amount has been corrected.

{

double aBalance = balance + amount - fee;//Declares and sets variable aBalance to amount entered by operator, plus fee set in class variables, plus the current balance.

Console.WriteLine("Success fully deposited " + amount.ToString("C") + ". Current balance: " + aBalance.ToString("C"));//displays amount entered, plus the new balance.

int aTranCount = tranCount + 1;//Declares an int variable to hold the increased by 1 value of tranCount.

Customer t = new Customer(this.firstName, this.lastName, this.dob, this.accNum, aBalance, aTranCount);

//a new Customer object with custData[i] objects name variables, dob, accNum and updated balance and tranCount variables.

s = t; //sets the values of the initial Customer object, s, to the values of the new Customer object, t.

//Allows the values for t to be called from outside the loop.

}

}

catch(ArgumentOutOfRangeException)//ReadLine error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.DepositHelper();//Operator error, allows recovery.

}

catch(FormatException)//Parse error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.DepositHelper();//Operator error, allows recovery.

}

return s; //returns the customer object with new balance and tranCount values.

}

/\*

\* Public method called from Menu Class, by

\* menu selection number 2.

\* Takes in Customer[] custData array in method signature

\* so array can be used within the method.

\* Calls WithdrawHelper() method in Customer class for Customer objects,

\* and in VIPCustomer class for VIPCustomer objects.

\* Iterates through array for account number, and calls that custData[i]

\* objects WithdrawHelper() method.

\* Then updates the balance value of the array with the new balance, and

\* updated transaction count (tranCount).

\* See Deposit() method for comments and logic.

\*/

public void Withdraw(Customer[] custData)

{

Console.Clear();//See deposit method for comments and logic.

bool contains = false;

Console.WriteLine("-------------------------------");

Console.WriteLine(" Withdraw ");

Console.WriteLine("-------------------------------");

try

{

Console.WriteLine("Enter accountID: ");

String accountID = Console.ReadLine();

if (accountID == null)

{

Console.WriteLine("Please enter an account ID");

}

for (int i = 0; i < custData.Length; i++)

{

Customer customer = custData[i];

this.firstName = customer.firstName;

this.lastName = customer.lastName;

this.dob = customer.dob;

this.accNum = customer.accNum;

this.balance = customer.balance;

this.tranCount = customer.tranCount;

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

if (accountID == accNum)

{

custData[i] = custData[i].WithdrawHelper();

contains = true;

}

}

}

catch(ArgumentOutOfRangeException exWith)//ReadLine error

{

Console.WriteLine(exWith);//This is not recovered from as it is a read error. Input errors are handled in method.

}

if (contains == false)

{

Console.WriteLine("Account ID invalid, press any key to try again");

Console.ReadKey();

Withdraw(custData);

}

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

/\*

\* Helper method that handles class specific withdrawal

\* method. Charges a fee that is set in class variables,

\* for each withdraw transaction. Does not allow a withdrawal if

\* amount (plus fee) to be withdrawn would take balance below $0.00.

\* Takes in and uses that values for the custData[i]

\* object that was used to call it.

\* Returns a Customer object to Customer Class's Withdraw() method.

\* Protected as it is a virtual method, that is only used within class

\* and overridden in subclass.

\* See method DepositHelper() for most comments and logic.

\*/

protected virtual Customer WithdrawHelper()

{

bool successfull = false;//Declares and initialises to false, a Boolean for whether or not the transaction was successful (amount to be withdrawn is less than or equal to balance).

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

try

{

Console.WriteLine("Withdrawing from the account of " + firstName + " " + lastName + ". Current balance is: " + balance.ToString("C"));

Console.WriteLine("Enter amount to withdraw: ");

double amount = double.Parse(Console.ReadLine());

if (amount < 0)//error handling for negative amounts.

{

Console.WriteLine("Please enter a non-negative amount. Press any key to continue");

Console.ReadKey();

s.WithdrawHelper();

}

else//allows the program to continue with positive amounts.

{

double totalAmount = amount + fee;//Declares a double variable and sets amount to be withdrawn to amount entered plus fee set in class variables.

if (totalAmount <= balance)//checks whether total amount to be withdrawn is less than or equal to current balance.

{

double aBalance = balance - totalAmount;//Declares and sets variable aBalance to current balance minus totalAmount to be withdrawn

int aTranCount = tranCount + 1;

Console.WriteLine("Successfully withdrew " + amount.ToString("C") + ". Current balance: " + aBalance.ToString("C"));

Customer t = new Customer(this.firstName, this.lastName, this.dob, this.accNum, aBalance, aTranCount);

successfull = true;

s = t;

return t;//returns customer object t, so that every code path returns a Customer object.

}

if (successfull == false)//fires if transaction was declined.

{

Console.WriteLine("Sorry, insufficient balance");

return s;//returns initial customer s, so that every code path returns a Customer object.

}

}

}

catch(ArgumentOutOfRangeException)//ReadLine error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.WithdrawHelper();//Operator error, allows program to recover.

}

catch(FormatException)//Parse error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.WithdrawHelper();//Operator error, allows recovery.

}

return s;//returns either updated customer s, or initial customer s, depending on value of successfull.

}

/\*

\* Public method called from Menu Class, by

\* menu selection number 5.

\* Takes in Customer[] custData array in method signature

\* so array can be used within the method.

\* Iterates through array for youngest customer, and calls that custData[i]

\* objects displayInfo method.

\* See Deposit() and LeapYear() methods for relevant comments and logic.

\*/

public void YoungestCustomer(Customer[] custData)

{

Console.Clear();

Console.WriteLine("-------------------------------");

Console.WriteLine(" Youngest Customer ");

Console.WriteLine("-------------------------------");

Customer youngest = new Customer(custData);//Declares and initialises a Customer object variable.

DateTime aAge = Convert.ToDateTime("1-1-1800");//Declares and initialises initial age value to a String value, converted to a DateTime object.

//Error not being handled as this a local variable initialisation.

for (int i = 0; i < custData.Length; i++)

{

Customer customer = custData[i];

this.firstName = customer.firstName;

this.lastName = customer.lastName;

this.dob = customer.dob;

this.accNum = customer.accNum;

this.balance = customer.balance;

this.tranCount = customer.tranCount;

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

DateTime bDay = Convert.ToDateTime(dob);//Declares a DateTime object variable and sets to custData[i] objects dob value converted to a DateTime object.

if (bDay > aAge) //fires if dob is more than value held in age (younger than).

{

youngest = custData[i];//sets the youngest Customer object variable to the custData[i] Customer object.

aAge = bDay;//sets the value of aAge variable to the current youngest objects date of birth.

}

}

youngest.DisplayInfo();//displays the class appropriate customer information.

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

/\*

\* Public method called from Menu Class, by

\* menu selection number 3.

\* Takes in Customer[] custData array in method signature

\* so array can be used within the method.

\* Iterates through array for looking for highest balance value, and sets that

\* to a variable. Then uses this variable to iterate through the array again,

\* so all customers with that balance can be dispayed, by calling

\* the class appropriate DisplayInfo() method.

\* See LeapYear() and Deposit() methods for relevant comments.

\*/

public void HighestBal(Customer[] custData)

{

Console.Clear();

Console.WriteLine("-------------------------------");

Console.WriteLine(" Max Balance ");

Console.WriteLine("-------------------------------");

double aBal = 0.00;//Declares and initialises the variable to hold the current highest balance.

for (int i = 0; i < custData.Length; i++)

{

Customer customer = custData[i];

this.firstName = customer.firstName;

this.lastName = customer.lastName;

this.dob = customer.dob;

this.accNum = customer.accNum;

this.balance = customer.balance;

this.tranCount = customer.tranCount;

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

if (balance > aBal)//Checks to see if the Customer objects balance is higher than the currently held aBal value.

{

aBal = balance;//sets the aBal value to the new highest balance value.

}

}

for (int i = 0; i < custData.Length; i++)

{

Customer customer = custData[i];

this.firstName = customer.firstName;

this.lastName = customer.lastName;

this.dob = customer.dob;

this.accNum = customer.accNum;

this.balance = customer.balance;

this.tranCount = customer.tranCount;

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

if (balance == aBal)//fires if a Customer objects balance equals the previously set highest balance.

{

DateTime today = DateTime.Now;//Declares and initialises a DateTime object to current system time

DateTime bDay = Convert.ToDateTime(s.dob);//Declares a DateTime object and initialises it by converting Customer objects dob to a DateTime object.

int day = today.Day;//Declares and initialises the int value for todays day (date)

int month = today.Month;//Declares and initialises the int value for current month

int dobDay = bDay.Day;//Declares and initialises the day value of Customer objects dob

int dobMonth = bDay.Month;//Declares and initialises the month value of Customer objects dob

//Year values are not needed in this method.

custData[i].DisplayInfo();//calls the appropriate DisplayInfo() method.

if((day == dobDay) && (month == dobMonth))//fires if Customer objects dob day and month values match todays day and month values.

{

Console.WriteLine("Happy Birthday " + custData[i].firstName);

}

Console.WriteLine(""); //Places a line between Customers information on Console screen for readability.

}

}

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

/\*

\* Public method called from Menu Class, by

\* menu selection number 4.

\* Takes in Customer[] custData array in method signature

\* so array can be used within the method.

\* Iterates through array for highest tranCount value, and calls that custData[i]

\* objects DisplayInfo() method.

\* For relevant comments see LeapYear() and Deposit() methods.

\*/

public void MostActive(Customer[] custData)

{

Console.Clear();

Console.WriteLine("-------------------------------");

Console.WriteLine(" Most Active ");

Console.WriteLine("-------------------------------");

int mostActive = 0;//Declares and initialises variable for highest tranCount to 0.

Customer mostActiveCust = new Customer();//Declares and initialises a Customer Object to hold current most active custData[i] object.

for (int i = 0; i < custData.Length; i++)

{

Customer customer = custData[i];

this.firstName = customer.firstName;

this.lastName = customer.lastName;

this.dob = customer.dob;

this.accNum = customer.accNum;

this.balance = customer.balance;

this.tranCount = customer.tranCount;

Customer s = new Customer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.tranCount);

if (tranCount > mostActive)//Fires when tranCount of a custData[i] object is higher than current value of mostActive.

{

mostActiveCust = custData[i];//sets mostActiveCust Object variable to value of custData[i]'s object.

mostActive = tranCount;//sets mostActive variable to tranCount of current custData[i] object.

}

}

mostActiveCust.DisplayInfo();//calls class appropriate DisplayInfo() method.

Console.WriteLine("Number of transactions: " + mostActiveCust.tranCount);//writes tranCount for the most active customer

Console.WriteLine("");

Console.WriteLine("Press any key to continue...");

Console.ReadKey();

}

}

}

**VIPCustomer Class code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BIT694\_TM1\_3431274

{

class VIPCustomer: Customer

{

//Most Class variables inherited from Customer.

private String custType;//Declares VIP customer variable. Private as it is only used in VIPCustomer.

Customer[] custData;//allows method to call and use array.

//Constructer that takes in firstName, lastName, dob, accNum, balance, custType and tranCount as method signature. Inherits from Customer constructor

public VIPCustomer(String firstName, String lastName, String dob, String accNum, double balance, String custType, int tranCount) : base(firstName, lastName, dob, accNum, balance, tranCount)

{

//other variables inherited from Customer via : Base call.

this.custType = custType;//sets custType to custType in signature

}

//A constructor that takes the custData array in its signature.

public VIPCustomer(Customer[] custData)

{

this.custData = custData;//sets array value in signature to array value.

}

//A constructor with no signature.

public VIPCustomer() { }

//Most get and set methods inherited from Customer class.

//Allows customer type to be gotten and set.

public String AccessCustType

{

set { this.custType = value; }

get { return this.custType; }

}

//A helper method that displays a VIP customers basic information.

//No error handling reasons, see virtual method in Customer.

protected override void DisplayInfo() //protected is the lowest access level that can be used for an override method.

{

DateTime aDob = Convert.ToDateTime(dob);

String bDob = aDob.ToShortDateString();

Console.WriteLine(custType);

Console.WriteLine("Name: " + firstName + " " + lastName);

Console.WriteLine("Date of birth: " + bDob);

Console.WriteLine("Balance: " + balance.ToString("C"));

}

/\*

\* Helper method that handles class specific deposit

\* method. Charges no fees.

\* Takes in and uses that values for the custData[i]

\* object that was used to call it.

\* Returns a Customer object to Customer Class's Deposit() method.

\* Protected as it is an overridden method, that is only used within class

\* and subclass.

\* See virtual method in Customer for comments and logic.

\*/

protected override Customer DepositHelper()

{

VIPCustomer s = new VIPCustomer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.custType, this.tranCount);

try

{

Console.WriteLine("Depositing into the account of " + firstName + " " + lastName + ". Current balance is: " + balance.ToString("C"));

Console.WriteLine("Enter amount to deposit: ");

double amount = double.Parse(Console.ReadLine());

if (amount < 0)//error handling, does not allow negative amounts to be deposited.

{

Console.WriteLine("Please enter a non negative amount. Press any key to continue");

Console.ReadKey();

s.DepositHelper();

}

else//allows the program to continue if number is non-negative, and after amount has been corrected.

{

double aBalance = balance + amount;

int aTranCount = tranCount + 1;

Console.WriteLine("Success fully deposited " + amount.ToString("C") + ". Current balance: " + aBalance.ToString("C"));

VIPCustomer t = new VIPCustomer(this.firstName, this.lastName, this.dob, this.accNum, aBalance, this.custType, aTranCount);

s = t;

}

}

catch(ArgumentOutOfRangeException)//ReadLine error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.DepositHelper();//Catching operator error, allows program recovery.

}

catch(FormatException)//Parse error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.DepositHelper();//Catching operator error, allows program recovery.

}

return s;

}

/\*

\* Helper method that handles class specific withdrawal

\* method. Charges no fee, and allows accounts to become

\* overdrawn.

\* Takes in and uses that values for the custData[i]

\* object that was used to call it.

\* Returns a Customer object to Customer Class's Withdraw() method.

\* Protected as it is an overridden method, that is only used within class

\* and subclass.

\* See virtual method in Customer for most comments and logic.

\*/

protected override Customer WithdrawHelper()

{

bool successfull = false;

VIPCustomer s = new VIPCustomer(this.firstName, this.lastName, this.dob, this.accNum, this.balance, this.custType, this.tranCount);

try

{

Console.WriteLine("Withdrawing from the account of " + firstName + " " + lastName + ". Current balance is: " + balance.ToString("C"));

Console.WriteLine("Enter amount to withdraw: ");

double amount = double.Parse(Console.ReadLine());

if (amount < 0)//error handling for negative amounts.

{

Console.WriteLine("Please enter a non-negative amount. Press any key to continue");

Console.ReadKey();

s.WithdrawHelper();

}

else//allows the program to continue with positive amounts.

{

double totalAmount = amount;

double aBalance = balance - amount;

int aTranCount = tranCount + 1;

Console.WriteLine("Successfully withdrew " + amount.ToString("C") + ". Current balance: " + aBalance.ToString("C"));

VIPCustomer t = new VIPCustomer(this.firstName, this.lastName, this.dob, this.accNum, aBalance, this.custType, aTranCount);

s = t;

successfull = true;

if (successfull == false)//handles unexpected processing errors.

{

Console.WriteLine("Sorry, due to a bank error, the transaction did not work");

return s;

}

}

}

catch(ArgumentOutOfRangeException)//ReadLine error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.WithdrawHelper();//Catching operator error, allows program recovery.

}

catch(FormatException)//Parse error

{

Console.WriteLine("Please enter a valid amount. Press any key to continue");

Console.ReadKey();

s.WithdrawHelper();//Catching operator error, allows program recovery.

}

return s;

}

}

}

## **Part 2: Self-assessment and review**

|  |  |  |
| --- | --- | --- |
| Criteria | Maximum  mark | Self-assessed  mark |
| Writing the class Customer with basic variables and methods. | 20% | 20% |
| Demonstration of inheritance and polymorphism. | 10% | 10% |
| Good use of programming techniques and modular programming   * The code in the Main function is kept to a minimum * Functions are short, not too long * Helper functions are defined when appropriate. | 30% | 28% |
| Error handling   * Input file does not exist * Lines have the wrong format * The program is unable to convert string to an integer * The program receives unexpected value from the user * Additional error handling where appropriate. | 10% | 8% |
| Criteria | Maximum mark | Self-assessed mark |
| Comments and clarity   * All variables declared as class instance fields need comment statement for their usage; all lines of code need to be indented properly according to their logical scope * Clear use of private, protected and public properties * Clear description of your program: purpose, inheritance and polymorphism. | 30% | 28% |
| Total | 100% | 96% |