**CIS 162 Project 4**

**A Credit Card Customer Database**

**Due Date**

* at the start of lab on Tuesday, July 7, 2015

**Before Starting the Project**

* Read chapter 7.4 – 7.5
* Read this entire project description before starting

**Learning Objectives**

After completing this project you should be able to:

* *use* ArrayLists to maintain and process collections of objects
* *use* for-each loops
* *use* while loops
* *read data* from external text files
* *use* static methods available in the Java library

**Step 1: Create a New BlueJ Project**

**Step 2: Download Files**

* Download the “CustomerNames.txt” file and save it in the folder that BlueJ created for this new project. There are 10,000 entries! You will not see it from within BlueJ but you can see it from within the Windows Explorer.
* Download the provided “GUI.java” and save it in the project folder. You should see this class within BlueJ. Do not make any changes to it until later.

**Step 3: Create a class called Customer (5 pts)**

Implement a class to maintain information about a customer including: gender (boolean), first name, last name, birth date (day, month, year), address, city, ST, zip code, credit card brand and current balance (double).

* Provide appropriate names and data types for each of the instance variables.
* public Customer (String info) - a constructor that initializes all of the instance variables to appropriate values. See information below about parsing Strings into smaller pieces.

female, Amy, Zu, 4/9/1960, 54 Elm St, Any City, NC, 27834, Visa, 15339.19

* public boolean isFemale() – return true if female and false if male.
* provide get methods for each of the instance variables (except gender).
* public String getFirst()
* public String getLast()
* public String getAddress()
* public String getCity()
* public String getState()
* public int getZip()
* public String getCreditCard()
* public int getDay()
* public int getMonth()
* public int getYear()
* public double getBalance()
* provide matching set methods for each instance variable. Although the set methods are not used for this project it is good practice to provide them for most classes.
* public String toString( ) - return a formatted mailing label (a single String with embedded newlines using "\n") . For example,

John Smith

123 Elm St.

Anytown, ST 12345

* public static void main(String args[]) – thoroughly test each of the methods in this class.

**Step 4: Create a class called CustomerDatabase (65 pts)**

Unless otherwise specified **use the elegant for-each loop** to search the ArrayList.

* Define an instance variable that holds an ArrayList of Customer. You should only need ONE instance variable.
* public CustomerDatabase ( ) - a constructor that instantiates an empty ArrayList in one line of code. No customers are inserted yet.
* public void readCustomerData(String filename) - open the provided filename and read all the data. There are 10,000 entries in “CustomerNames.txt”. Refer to the information about reading text files later in this document.
* public void addCustomer (Customer c) – add the customer to the ArrayList. This method contains one line of code.
* public int countCustomers ( ) - return the number of names in the database. This method only needs to contain one line of code!
* public int countCustomers (int zip) – this overloaded method has an identical name as the previous method but returns the number of customers with zip as a zipcode.
* public int countDebtFree ( ) - return the number of cardholders with no debt.
* public Customer getHighestDebt ( ) - return the person with the highest credit card balance.
* public Customer getYoungestCardholder ( ) - return the youngest person in the database. Give careful thought to this one! If you find more than one customer with the same age, return just ONE.
* public String getCardSummary(String card) - return a String that summarizes the total number of cards and average debt for the requested card type. Valid entries are: Discover, Visa and MasterCard. Return an appropriate error message if the card is not valid. If there are no customers with the specific card, return a message indicating that "Customers were not found". Use the NumberFormat class to help format the dollars and cents output. Sample result should look like the following.

Discover: 1981 cards with average balance of $3,735.20

* public ArrayList <Customer> getMailingList(double low, double high) - return an ArrayList of all cardholders who have a balance between low and high (inclusive).
* public ArrayList <Customer> getMailingList(String keyword) - return an ArrayList of all cardholders that have a city or state that contains the keyword (ignore case). For example, keyword of "IN" would return all cardholders in Indiana as well as any cities that contain "in". Convert strings to lower case before comparing in an effort to ignore case.

**Parsing Strings**

Customer data is contained in a long string with values separated by commas. You must write code that splits the long string into individual pieces and converts them to integers and doubles as necessary. Blank spaces at the start and end of String must be removed with the trim() method.

Limited sample of data

Amy, Zu, 4/9/1960, 15339.19

Here is sample code you will need to adapt. It is only processing six pieces of data.

public Customer(String info){

// Split the info string into multiple pieces separated by ‘,’ or ‘/’

String [] tokens = info.split(",|/");

first = tokens[0].trim();

last = tokens[1].trim();

month = Integer.parseInt(tokens[2].trim());

day = Integer.parseInt(tokens[3].trim());

year = Integer.parseInt(tokens[4].trim());

balance = Double.parseDouble(tokens[5].trim());

}**Customer Text File**

This file contains fictional customer information. A sample line of the data file is shown below. Note that items are separated by commas and the birthday is separated by slashes. The readCustomerData(String filename) method reads one line at a time and passes the text to the Customer class constructor.

female, Amy, Zu, 4/9/1960, 54 Elm St, AnyCity, NC, 27834, Visa, 15339.19

**Sample Method to Read a Text File**

The following method reads from a text data file one line at a time and prints to the screen. You will have to modify this code to create Customers and insert into the database.

**Sample Code**

public void readCustomerData(String filename){

// Attempt to read the complete set of data from file.

try{

File f = new File(filename);

Scanner sc = new Scanner(f);

String logline;

// read header information first

logline = sc.nextLine();

while(sc.hasNext()) {

logline = sc.nextLine();

// remove this print statement after method works

System.out.println(logline);

// add two lines of code to instantiate a new Customer

// and add to database

}

sc.close();

}

catch(IOException e) {

System.out.println("Failed to read the data file: " +

filename);

}

}

**Step 5: Software Testing (5 pts)**

Software developers must plan from the beginning that their solution is correct. BlueJ allows you to instantiate objects and invoke individual methods. You can carefully check each method and compare actual results with expected results. However, this gets tedious. Another approach is to write a main method that calls all the other methods.

Create 4-8 Customers and add them to the database. Your tests will be more thorough than the following incomplete example.

public static void main(String args[]){

String info = "male, Joe, Smith, 4/20/1963, addr, San Francisco, CA, 49401, Discover, 12345.67";

Customer cust1 = new Customer(info);

info = "female, Jo Anne, Henderson, 2/19/1972, addr2, Detroit, MI, 49401, Mastercard, 12000";

Customer cust2 = new Customer(info);

CustomerDatabase db = new CustomerDatabase();

db.addCustomer(cust1);

db.addCustomer(cust2);

System.out.println(db.getYoungestCardholder());

System.out.println(db.getHighestDebt());

}

**JUnit Testing**

As an alternative, JUnit is a Java library that helps to automate software testing. A JUnit test file DatabaseTest.java has been provided on BlackBoard. Refer to Project 2 instructions for additional information about JUnit Testing. Remember, your class names and method names must match exactly with these specifications. A compile error for DatabaseTest.java suggests a mismatch of some identifier.

**Sample Results**  
Results from the GUI with the text file properly read should yield the following results.

**Result from countCustomers**

Total Customers: 10000

Customers that are Debt Free: 2137

**Result from getHighestDebt**

Jessica Anderson

1957 Patterson Street

Houston, TX 77063

**Result of a search for “delf”**

0 customers found

**Result of a search for “allen”**

*First fourteen names omitted*

John Sadowski

2741 Dark Hollow Road

Allentown, NJ 8501

Jennie Lee

429 Jerome Avenue

Mcallen, TX 78501

16 names listed**Step 6: Create a GUI class (15 pts)**

Now that you have the basic application working within its own class it is time to create a more interesting graphical user interface for the user. See examples, posted on BlackBoard, of BorderLayout and BoxLayout.

* Modify the provided GUI class.
* Define instance variables for a CustomerDatabase object, and JButtons as needed to match the sample screenshot. The central JTextArea is provided for you.

**GUI Constructor**

* Instantiate a CustomerDatabase and invoke the readCustomerData() method.

db = new CustomerDatabase();

db.readCustomerData("CustomerNames.txt");

* Several menu items are created for you.
* The JTextArea is created for you and inserted in the CENTER region.
* The JPanel for the three TextFields with corresponding labels has been created for you and inserted in the SOUTH region.
* Create the following private method:

Private void setupButtons ()

* Create a JPanel for the four JButtons. This panel must use Box Layout to display buttons in a column.

JPanel eastPanel = new JPanel();

eastPanel.setLayout(new BoxLayout(eastPanel, BoxLayout.Y\_AXIS));

* Instantiate the buttons
* Add the buttons to the eastPanel
* Register the buttons with the ActionListener. An example:

youngestButton.addActionListener(this);

* Add the eastPanel to the EAST

theGUI.add(BorderLayout.EAST, eastPanel);

**ActionPerformed method**

The actionPerformed method has been created for you. You will add several if statements for each of the buttons and menu items.

if (e.getSource() == quitItem){

System.exit(1);

}

if(e.getSource() == youngest){

results.setText("");

c = db.getYoungestCardholder();

results.append(c.toString());

}

**Additional Methods**

**Add the following methods to the GUI class.**

* public void displayCustomers (ArrayList <Customer> list) – this method displays the mailing address of each customer in the list. Note that this method is invoked from a variety of places within the GUI class. The following example is incomplete but demonstrates the basic idea. This assumes the Customer toString() method is working properly.

results.setText("");

for (Customer c: list){

results.append(c + "\n");

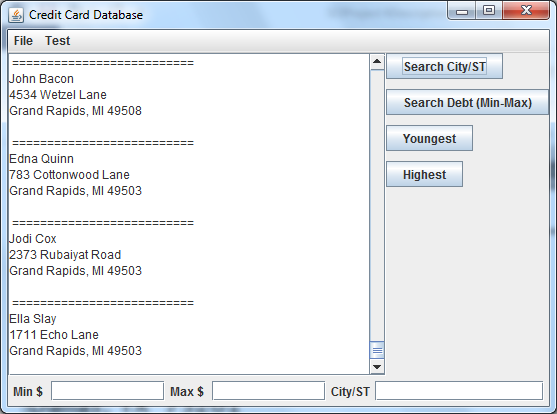
}

* public void showSummaries () – this method clears the display area and then shows the summaries for each of the three credit cards: Discover, MasterCard and Visa.

**Main method**

This method has been created for you.

**GUI Screenshot**

**Grading Criteria**

There is a 50% penalty on programming projects if your solution does not compile.

* Stapled cover page with your name and signed pledge. (-5 pts if missing)
* Project requirements as specified above. (90 pts)
* Elegant source code that follows the GVSU [Java Style Guide](http://www.cis.gvsu.edu/studentsupport/javaguide). (10 pts)

**Late Policy**

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

* The first 24 hours (-20 pts)
* Each subsequent weekday is an additional -10 pts
* Weekends are free days and the maximum late penalty is 50 pts.

**Turn In**

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

* Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
* Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/Academics/Honesty/).
* Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, “I spent 7 hours on this project from January 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document.”
* Source code - a printout of your elegant source code for the Customer, CustomerDatabase and GUI classes.
* Compress the BlueJ project folder and submit the compressed file using Blackboard.