**CIS 162 Project 4**

**Baby Names Database**

**Due Date**

* at the start of class on November 14, 2017

**Before Starting the Project**

* Read sections 8.1 - 8.5 about ArrayLists
* 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
* *read data* from external text files

**U.S. Baby Names**

The U.S. Social Security Administration has tracked all births since 1880 and provides a database with over one million records. Each record contains: a name, gender, number of births with that name and a year. We simplified and cleaned up the data. We also removed all records with fewer than 100 births in a year but there are still over 200,000 records.

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

**Step 2: Download Data File**

* Download the “BabyNames.csv” file and save it in the folder that BlueJ created for this new project. There are over 200,000 records! You will not see the file within BlueJ but you can see it from within the Windows Explorer.

**Step 3: Create a class called BabyName (5 %)**

Hundreds of babies are born each year in the U.S. with the same name. For example, there were 346 girls named Amelia born in 1889. This simple class stores birth information about a single name for a specific year: name (String), gender (boolean), number of new born babies given that name and birth year (integers).

* Provide appropriate names and data types for each of the four instance variables.
* public BabyName (String n, boolean g, int count, int yr) - a constructor that initializes all of the instance variables to appropriate values.
* public boolean isFemale() – returns true if a girl. Otherwise, return false.
* provide get methods for the name, count and year.
* provide matching set methods for each instance variable. Although set methods are not used for this project it is good practice to provide them for most classes.
* public String toString() - return a formatted String. For example,

46,473 girls named Jessica in 1990

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

**Step 4: Create a class called BabyNamesDatabase (45 %)**

Create a class that maintains an unlimited number of baby names. **Use the elegant for-each loop** to search the ArrayList.

**Class Fields**

* Define an instance variable that holds an ArrayList of BabyName. Information is read from a text file.

**Constructor**

A *constructor* is a special method with the same name as the class and generally initializes the fields to appropriate starting values. Refer to section 3.7.

* public BabyNameDatabase () - instantiate an empty ArrayList for the names. No items are inserted yet. This method will be one line of code.

**Mutator Methods**

Mutator methods perform tasks that may modify class fields. Methods that set a field with the parameter value often begin with the prefix ‘set’. Refer to section 3.6.

* public void readBabyNameData(String filename) - open the provided filename and read all the data. There are thousands of records in “BabyNames.csv”. Starter code is provided later in this document.

**Accessor Methods**

An *accessor* method does not modify class fields. The names for these methods, which simply return the current value of a field, often begin with the prefix ‘get’. Refer to section 3.6.

* public int countAllNames () - return the number of items in the ArrayList using one line of code.
* public int countAllGirls() – Use a for-each loop to search the full list and count the total number of girls born since 1880. This number will be much higher than just the number of girl names since each name represents hundreds, if not thousands, of newborn babies.
* public int countAllBoys() – Use a for-each loop to search the full list and count the total number of boys since 1880. This number will be much higher than just the number of boy names since each name represents hundreds, if not thousands, of newborn babies.
* public BabyName mostPopularGirl(int year) – search the full list and return the most popular girl name for a specific year.
* public BabyName mostPopularBoy(int year) – search the full list and return the most popular boy name for a specific year.
* public ArrayList <BabyName> searchForName(String name) – search the full list and return a new list of baby names that match the requested name (in the parameter). Spelling should match exactly but it is not case sensitive. For example, ‘Angie’, ‘angie’ and ‘ANGIE’ all match. Hint, check out the String method called equalsIgnoreCase(). If no matches are found, the returned list will exist but have zero elements.
* public ArrayList <BabyName> searchForYear(int year) – search the full list and return a new list of baby names that match the requested year (in the parameter). If no matches are found, the returned list will exist but have zero elements.

**Step 5: Generate a Top Ten List (10 %)**

To generate a top ten list for a particular year your solution must be able to sort names in order by number of births. This requires small changes to BabyName and a new method in BabyNameDatabase.

**Changes to BabyName**

* Add two words to the end of the class header (shown below). You will need to import a new java package for this to compile. This allows BabyName objects to be compared using compareTo() and therefore sorted.

public class BabyName implements Comparable{

* Also, add the following method. You can try to copy and paste the code but if BlueJ gives unusual compile errors then simply retype it. This method allows two BabyName objects to be compared with respect to the number of births.

public int compareTo(Object other){

BabyName b = (BabyName) other;

return (b.count – count);

}

**Changes to BabyNameDatabase**

* public ArrayList <BabyName> topTenNames(int year) – search the full list and return a new list of the ten most popular baby names in a given year (in the parameter). The returned list will exist but have zero elements if no matches are found. Otherwise, it will have ten names. Avoid duplicating code by calling ssearchByYear() to first select all names in the year.

Sorting an ArrayList, called tempList, can be performed with one line of code (i.e. Collections.sort(tempList);)Remove all items from the temporary list except the first ten before returning the result. This may take some thought!

* Now that you can sort search results, sort the temporary lists in searchForYear() and searchForName() before returning. You only need to add one line of code to each method.

**Coding Style (10 %)**

Good programming practice includes writing elegant source code for the human reader. Follow the GVSU [Java Style Guide](http://www.cis.gvsu.edu/java-coding-style-guide/).

**Sample Results**  
The GUI should yield the following results.

**Result from selecting menu item Count**

Total Counts

Total Girls: 153,467,149

Total Boys: 161,943,150

Total Names: 227,881

**Last few lines for a search for “janet”**

\*

\*

\*

108 girls named Janet in 1900

108 girls named Janet in 1901

103 girls named Janet in 1897

102 girls named Janet in 1896

Total: 120

All years with janet

**Result of a search for most popular names of 1952**

Most Popular Names in 1952

87,061 boys named James in 1952

67,082 girls named Linda in 1952

**Results for top ten list in 1923**

Top Ten Baby Names in 1923

71,634 girls named Mary in 1923

57,469 boys named John in 1923

56,118 boys named Robert in 1923

52,137 boys named William in 1923

50,467 boys named James in 1923

39,042 girls named Dorothy in 1923

31,493 girls named Helen in 1923

28,960 boys named Charles in 1923

27,027 boys named George in 1923

26,133 girls named Margaret in 1923

Total: 10

**Step 6: Software Testing (10 %)**

Software developers must plan from the start 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.

**a) Create a short sample text file**

You do not know the correct answers for the provided data. Instead, you should create your own small text file with 10-15 baby names. Use NotePad, TextPad or similar basic text editor and save the file as plain text (.txt). Give careful thought to the names you create to test all of your database methods. Your file should have more names than the following.

Mary,F,7065,1880

**b) Create a new class called BabyTest with a main method.**

Testing a class can be done by creating a special program, sometimes known as a *testbench*, whose job is to thoroughly test the class. The process of creating and running a program that tests a specific class (or "unit"), is known as ***unit testing.*** (see section 4.12)

For this project, write a main method in a new class called BabyTest that instantiates a baby name database and invokes each of the methods with a variety of parameters. Provide multiple *if* statements to test each method along with error messages as needed. It takes careful consideration to anticipate and test every possibility. This is an incomplete example. **Your solution should be longer to test all methods in BabyNameDatabase.**

public static void main(String args[]){

BabyNameDatabase db = new BabyNameDatabase();

// read small data file created just for testing

db.readBabyNameData("MySampleData.txt");

// check number of records

if(db.countAllNames() != 6){

System.out.println("Error: Number of names should be 6");

}

// check most popular boy

BabyName popular = db.mostPopularBoy(1999);

String name = popular.getName();

if(name.equals("Scott") == false){

System.out.println("Error: Popular boy in 1999 should be Scott");

}

// check number of records for one year

ArrayList<BabyName> tempList = db.searchForYear(1999);

if(tempList.size() != 4){

System.out.println("Error: Should be 4 records in 1999");

}

System.out.println("Scanning complete.");

}

**Step 7: Adapt the provided BabyNameGUI class (15 %)**

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. Refer to zyBook Ch 6.1 – 6.3.

* Download the provided “BabyNameGUI.java” and save it in the project folder.
* The central JTextArea, called resultsArea, is provided for you and spans ten rows and one column to make room for the buttons and text fields.
* Several menu items are created for you.

**Instance Fields**

* Define instance variables for a BabyNameDatabase object, JTextFields and JButtons as needed to match the sample screenshot (Figure 1).

**In GUI Constructor**

* Instantiate the BabyNameDatabase object.
* The Results and Searches labels are provided for you.
* Define additional local JLabels for “Year” and “Name”. Labels can be local variables since they are not accessed from anywhere else in the class.
* Instantiate and display the remaining labels, text fields and buttons to match Figure 1.
* Look for FIX ME comments for clues of what to add.

****

**Figure 1. GUI Screenshot**

**In setupMenus ( )**

* Look for FIX ME comments to add the action listeners for the menu items

**In openFile ( )**

* Look for FIX ME comments

**In actionPerformed ( )**

* Add if statements for each of the buttons and menu items. Look for FIX ME comments for clues of what to add. Here are two samples to get you started.

if (buttonPressed == quitItem){

System.exit(1);

}else if (buttonPressed == topTen){

displayTopTen();

**Helper Methods (private)**

This helper methods are all called from actionPerformed() in an effort to keep it short and easy to read.

* private void displayNames (ArrayList <BabyName> list) – this method displays each record in the provided list. **Provide a final line of output stating how many items are listed.** The method is called by other helper methods. The following example is incomplete but demonstrates the basic idea.

resultsArea.setText("");

for(BabyName b : list){

resultsArea.append("\n" + b.toString());

}

* private void displayMostPopular () – display the most popular boy and girl names for the requested year. Read the requested year from the text field and call the appropriate database methods.
* private void displayByYear () – display all names for the requested year. Read the requested year from the text field and call the appropriate database method.
* private void displayTopTen () – display top names for the requested year. Read the requested year from the text field and call the appropriate database method.
* private void displayByName () – display all records for the requested name. Read the requested name from the text field and call the appropriate database method.
* private void displayCounts () – display the total number of records, total number of girls and total number of boys. Call the appropriate database methods.

**Step 8: Preventing User Errors (5 %)**

Unless you make special preparation, your code will crash if the user does not provide the requested information in each of the text fields depending on the action. Use a popup JOptionPane.showMessageDialog to display error messages: 1) if the database has not been loaded, 2) if the name field is empty when attempting a search or 3) if the year field is empty when displaying other results.

**Missing Fields**

You can check the contents of a text field and display an error message if it is empty.

Add code to the helper methods to test for:

* Most Popular if no year is provided
* Top Ten if no year is provided
* Search for Year if no year is provided
* Search for Name if no name is provided

Here is a sample to get you started:

if(yearField.getText().length() == 0){

JOptionPane.showMessageDialog(this, "Provide a Year");

}else{

// display most popular for the year

}

**Missing Database**

Within actionPerformed() check if the database has zero elements. If so, remind the user to open the file. The only actions allowed without a loaded database are “quit” and “open”.

**Background Information: Reading From a Text File**

The data file contains baby name records from the U.S. Social Security Administration since 1880. A sample line is shown below. The readBabyNameData() method reads four items and passes them to the BabyName class constructor.

Mary,F,7065,1880

The following method reads from a text file one line at a time. The solution is a bit more complex than an example in section 12.5.1 but similar to Figure 11.3.3 in section 11.3. Look for FIX ME comments for suggestions of improvements.

**Sample Code**

public void readBabyNameData(String filename){

// Read the full set of data from a text file

try{

// open the text file and use a Scanner to read the text

FileInputStream fileByteStream = new FileInputStream(filename);

Scanner scnr = new Scanner(fileByteStream);

scnr.useDelimiter("[,\r\n]+");

// keep reading as long as there is more data

while(scnr.hasNext()) {

// FIX ME: read the name, gender, count and year

String name = scnr.next();

String gender = scnr.next();

int count;

int year;

// FIX ME: remove this print statement after method works

System.out.println(name);

// FIX ME: assign true/false to boolean isFemale based on

// the gender String

boolean isFemale;

// FIX ME: instantiate a new Baby Name and add to ArrayList

BabyName entry = new BabyName(name, isFemale, count, year);

}

fileByteStream.close();

}

catch(IOException e) {

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

}

}

**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 % if missing)

**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 %)
* Each subsequent weekday is an additional -10 %
* Weekends are free days and the maximum late penalty is 50 %.

**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, an appropriate picture or clip art for the project, time card and signed pledge (-5 pts if missing)
* 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). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the School of CIS Guidelines for Academic 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:

BabyName.java

BabyNameDatabase.java

BabyNameGUI.java

* Demo – be prepared to demo your project on a lab computer or your laptop.