# CIS 162 Project 3

# A COVID-19 Database

**Due Date**

Friday, December 11, 2020 before midnight.

Suggested phases:

* Phase 1 – one day
* Phase 2 – one week
* Phase 3 – one week

**Before Starting the Project**

* You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](https://www.cis.gvsu.edu/academic-honesty/).
* Read chapters 10 and 11 – GUI and ArrayLists
* Read chapter 14 – section 14.4
* 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 to process ArrayLists
* *read data* from external text files

**Project Summary**

Create a GUI application that allows someone to search a list of more than 10,000 Covid-19 data records. Each record contains six items: state, month, day, daily infections, daily deaths, and total infections and total deaths for that state up to that date. The database you create will allow health officials determine all sorts of statistics related to the Covid-19 pandemic in the U.S. The data is from [https://covidtracking.com](https://covidtracking.com/)

(March 1 – September 27, 2020)

# Phase 1 (10 pts)

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

**Step 2: Download Data File**

* Download the “covid\_data.csv” file and save it in the folder that BlueJ created for this new project. There are more than 10,000 entries! You will not see it from within BlueJ but you can see it from within Windows Explorer or Mac Finder.

**Step 3: Create a class called CovidEntry**

**Important Note:**

**We are providing two JUnit classes to test your project. Exact spelling is required in the CovidEntry class for the class name and all the method headers. Do not change the method headers in any way.**

#### Instance Variables:

* state (String)
* month and day (integer)
* daily deaths and daily infections (integer).
* total deaths and total infections (integer)

#### Constructor

* public CovidEntry(String st, int m, int d, int di, int dd, int ti, int td) - a constructor that initializes all the instance variables to appropriate values.

The input parameters are:

* st - state
* m - month
* d - day
* di - daily infections
* dd - daily deaths
* ti - total infections
* td - total deaths

#### Accessor Methods

* public int getMonth() – return the month
* public int getDay() – return the day
* public String getState() – return the state

NOTE: All the following methods return the required value for a particular state, month, and day

* public int getDailyInfections()- return the number of daily infections
* public int getDailyDeaths() - return the number of daily deaths
* public int getTotalInfections()- return the total number of infections
* public int getTotalDeaths()- return the total number of deaths
* public String toString()- return a String with the representation of a CovidEntry object. Use the DecimalFormat class to use commas for the thousands.

Example:

**NY 4/20 4,726 infections, 478 deaths**

**JUnit Testing**

Download CovidEntryJUnit.java to the folder of your project and run the test case – testing the constructor.

# Phase 2 (50 pts)

**Step 4: Create a class called CovidDatabase**

**Important Note:**

**We are providing a JUnit class to test your project. Exact spelling is required in the CovidDatabase class for the class name and all the method headers. Do not change the method headers in any way.**

#### Instance Variables:

* a reference to an ArrayList of CovidEntry objects

#### Constructor

* public CovidDatabase() - a constructor that instantiates an ArrayList of CovidEntry. This method will be one line of code.

#### Mutator Methods

* public void readCovidData(String filename) – reads the file and populates the ArrayList of CovidEntry objects.
  + open the provided filename
  + read the first record that contains the descriptions of the fields and do not store this information in any fields.
  + use a loop to repeatedly:
    - read data, one element at a time
    - instantiate a new CovidEntry object passing the data read as input parameters to the CovidEntry constructor
    - add the created object to the ArrayList.

**Background Info: Reading Text Files**

The first four lines of the data file are shown below. The first line provides descriptors for-each record. Read this line and do not assign it to any variable. Items are separated by commas. The database reads one item at a time. Note, the Scanner needs to be informed that commas, instead of blank spaces, will be used as a delimiter. A carriage return or new line character (aka the Enter key) is also used as a delimiter between each line.

**Sample record of the covid\_data.csv file: (first four records).**

state,month,day,dailyInfect,dailyDeaths,totalInfect,totalDeaths

WA,3,1,16,3,34,8

VA,3,1,0,0,0,0

RI,3,1,0,0,2,0

**Sample Method to Read a File**

The following method reads from a text data file one line at a time. The solution is a bit more complex than shown in Section 14.4 because the book is hiding some necessary details. The data file for this example includes a name, an age and a GPA on one line separated by commas. Your solution for reading the COVID data will be similar.

Johnny,18,1.9

Mary,21,3.9

Jasmine,20,2.8

public void sampleReadData(String filename){

Scanner inFS = null;

FileInputStream fileByteStream = null;

try{

// open the File and set delimiters

fileByteStream = new FileInputStream(filename);

inFS = new Scanner(fileByteStream);

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

// continue while there is more data to read

while(inFS.hasNext()) {

// read three data elements

String name = inFS.next();

int age = inFS.nextInt();

double gpa = inFS.nextDouble();

}

fileByteStream.close();

// error while reading the file

}catch(IOException error1) {

System.out.println("Oops! Error related to: " + filename);

}

}

**Accessor Methods**

* public int countRecords() – return the number of Covid entries. This method should be one line only.
* public int getTotalDeaths() – return the sum of all daily deaths. Use a for-each loop to process the ArrayList.
* public int getTotalInfections() – return the sum of all daily infections. Use a for-each loop to process the ArrayList.
* public int countTotalDeaths(int m, int d) – return the sum of all daily deaths from all states on the specified date. Use a for-each loop to process the ArrayList.
* public int countTotalInfections(int m, int d) – return the sum of all daily infections from all states on the specified date. Use a for-each loop to process the ArrayList.
* public CovidEntry peakDailyDeaths(String st) - return the CovidEntry object with the highest daily death for the requested state. If there are no entries for the state entered as input parameter return null. State abbreviations are stored in ALL CAPS, but you want to allow the user to type lower-case as well in the request. Therefore, use the String method equalsIgnoreCase.This allows someone to provide "fl" for "FL" and the search will still work for Florida. Use a for-each loop to process the ArrayList.
* public ArrayList <CovidEntry> getDailyDeaths(int m, int d) - return an Array list of all the records for a specific date. The ArrayList returned should have zero elements if no records were found for the date entered as parameter. Use a for-each loop to process the ArrayList.
* public CovidEntry peakDailyDeaths(int m, int d) - return the CovidEntry object with the highest daily death for the requested date. Use a for-each loop to find out the highest daily deaths for the month and day. If there are no entries for the month and day entered as input parameter return null.
* public CovidEntry mostTotalDeaths() - return the CovidEntry object with the highest total deaths. Use a for-each loop to process the ArrayList.
* public ArrayList <CovidEntry> listMinimumDailyInfections(int m, int d, int min) – return a new ArrayList containing all records (CovidEntry objects) that match the requested date AND have a minimum requested daily infection. For example, return all records from June 3rd with at least 1,000 daily infections. The ArrayList returned should have zero elements if no records were found for the input parameters.

**Safe to Open**

Public officials are encouraged by the U.S. Center for Disease Control to keep a state shut down until an appropriate downward trend in new daily infections is observed. The ideal goal is 14 days. Instead, we will use the lower threshold of only five consecutive days of decreasing daily infections. For program flexibility, declare a final instance variable that represents the required number of days. This can be changed easily by public officials.

private static final int SAFE = 5;

* public ArrayList <CovidEntry> safeToOpen(String st) – process the database from start to end looking for the first five consecutive days of decreasing daily infections for the requested state. There could be multiple safe stretches, but we want to return the earliest available open date. The records in the file are in increasing order by date. Return a new ArrayList with the five CovidEntry objects leading to the safe reopen. Return null if the state does not achieve this goal, or if the state entered as input parameter is not found in the ArrayList (invalid state). See sample results in Testing section below.

**Generate a Top Ten List**

This method requires sorting an ArrayList of CovidEntry objects in descending order by number of daily deaths.

This requires changes to the CovidEntry and CovidDatabase classes.

#### Changes to CovidEntry class

* Add two words to the end of the class header (shown below). This allows CovidEntry objects to be compared using compareTo(). The compareTo method will be used internally to do the sorting of the records.

public class CovidEntry **implements Comparable**{

Also, add the following method. This method allows two CovidEntry objects to be compared with respect to the number of daily deaths.

public int compareTo(Object other){

CovidEntry c = (CovidEntry) other;

return c.dailyDeaths - dailyDeaths;

}

#### Changes to CovidDatabase class

Add the following method to CovidDatabase.

* public ArrayList <CovidEntry> topTenDeaths(int m, int d) – return a new ArrayList of CovidEntry objects for the ten states with the highest daily deaths on the requested date. If there are not records for the requested date, the ArrayList returned will have zero elements. Results should be in sorted order from high to low. Here are a few hints.
* Create a new ArrayList containing all entries with the requested date. To avoid repeating code, invoke the getDailyDeaths method.
* Sort the new ArrayList with the help of the Collections Java class. The following example assumes your new temporary list is called “list”.

Collections.sort(list);

Once you have the list of all the daily deaths for the specific date sorted in descending order by the number of deaths, you can do any of these three options to figure out the top ten states with the highest number of deaths for that date.

* + you can remove all items from the temporary list except the first ten before returning the result. Use a for loop that goes backwards if you want to use this option. Remember the ArrayList shrinks when you delete objects.
  + you may create another temporary list and add to this new list only the first 10 records from the list created above
  + you may use the subList and removeAll methods of the ArrayList class. See Java API for documentation on how to use these two methods.

**Coding Style (10 pts)**

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/).

**Step 5: Software Testing (10 pts)**

The be able to test this project you need to know the results for a specific query. One strategy would be to create a test database with a few dozen records carefully crafted to test each method. For example, you create a specific record with the highest total deaths and confirm the method returns this appropriate record.

However, we will provide sample results for your comparison.

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.

**Create a new class called CovidDatabaseTest with a main method.**

Write a main method in a new class called CovidDatabaseTest that instantiates a CovidDatabase object and invokes each of the methods with a variety of parameters. 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 CovidDatabase.**

public class CovidDatabaseTest {

public static void main () {

System.out.println ("Testing starts");

CovidDatabase db = new CovidDatabase() ;

db.readCovidData("covid\_data.csv");

// check number of records, total infections, and total deaths

assert db.countRecords() == 10346 : "database should have 10,346";

assert db.getTotalDeaths() == 196696 : "Total deaths should be: 196,696";

assert db.getTotalInfections() == 7032090 : "infections should be: 7,032,090";

// check peak daily deaths for 5/5

CovidEntry mostDeaths = db.peakDailyDeaths(5, 5);

assert mostDeaths.getState().equals("PA") : "State with most deaths for 5/5 is PA";

assert mostDeaths.getDailyDeaths() == 554 : "Deaths for 5/5 is PA: 554";

// test other methods

System.out.println ("Testing ends");

}

}

**Sample Results – data from Mar 1 – Sep 27, 2020**

|  |  |
| --- | --- |
| Method | Results |
| countRecords () | 10,346 |
| getTotalDeaths () | 196,696 |
| getTotalInfections () | 7,032,090 |
| mostTotalDeaths() | NY with 25,456 deaths |
| peakDailyDeaths ("MI") | MI 4/16 922 infections, 169 deaths |
| peakDailyDeaths (5 , 5) | PA 5/5 865 infections, 554 deaths |
| topTenDeaths (5, 5) | Top Ten Daily Deaths for 5/5  PA 5/5 865 infections, 554 deaths  NJ 5/5 2,324 infections, 341 deaths  NY 5/5 2,239 infections, 230 deaths  IL 5/5 2,122 infections, 176 deaths  CT 5/5 1,334 infections, 138 deaths  MA 5/5 1,184 infections, 122 deaths  FL 5/5 542 infections, 113 deaths  OH 5/5 495 infections, 79 deaths  GA 5/5 343 infections, 66 deaths  CA 5/5 1,275 infections, 63 deaths |
| safeToOpen ("MI") | MI is safe to open  MI 5/26 443 infections, 25 deaths  MI 5/27 386 infections, 24 deaths  MI 5/28 336 infections, 24 deaths  MI 5/29 319 infections, 27 deaths  MI 5/30 205 infections, 24 deaths |
| listMinimumDailyInfections(6,12,1000) | All states with at least 1000 infections on 6/12  TX 6/12 2,097 infections, 19 deaths  NC 6/12 1,768 infections, 28 deaths  FL 6/12 1,902 infections, 29 deaths  CA 6/12 2,702 infections, 62 deaths  AZ 6/12 1,654 infections, 17 deaths |

**JUnit Testing**

Download CovidDatabaseJUnit.java to the folder of your project and run the test cases.

# Phase 3 (20 pts)

**Step 6: Complete a CovidDatabaseGUI class (15 pts)**

Now that you have the Database working in its own class it is time to create a more interesting graphical user interface for someone to use. Once again, we provide you a starting template. **Read the code for clues of how to position the other elements.** Read Zybooks - sections 10.1 – 10.3 and the additional link at the end of section 10.2 - How To Use GridBag Layout.

* Download the provided “CovidGUI.java” and save it in the project folder. You should see this class within BlueJ (although you may have to restart BlueJ).

**Additional instance variables**

* You will need to declare additional variables for buttons, menu items and text fields. Read the internal comments for clues. Your goal is to recreate Figure 1.
* Declare an instance field for a CovidDatabase object. See the FIX ME comment.

**Changes to main()**

* Change the JFrame title to display your name. See the FIX ME comment.

**Changes to the constructor**

* Create the CovidDatabase object using the new operator. See the FIX ME comment.
* Add statements to instantiate and display each of the GUI components. Buttons span two columns and are center justified.
* Add statements to register the additional buttons. See FIX ME comments for clues.

**Changes to setupMenus()**

* **This code is provided for you.**

**Changes to actionPerformed().** See the FIX ME comment.

* Add if statements for-each of the button clicks AND the two menu items. Invoke the appropriate private methods described below.

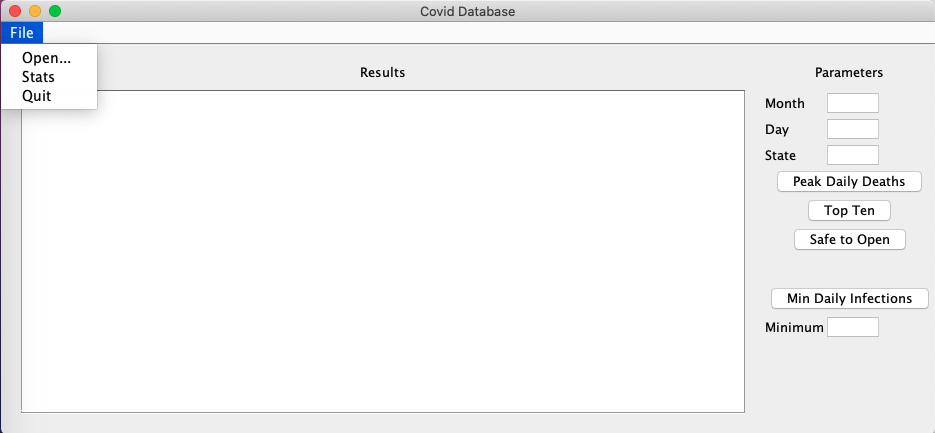
if (e.getSource() == peakBtn){

displayPeakDeaths();

}

**Add helper methods**

* Write the helper methods described on the next page.

****

**Figure 1. GUI layout**

**Helper Methods**

Helper methods keep the code easy to read by hiding details away in other areas of the program. These methods work with the Database to display results within the **results** text area using setText() and appendText(). For example:

int total = database.countTotalDeaths(5,23);

results.setText("Total U.S. Deaths\n");

results.append(total);

* private void displayPeakDeaths() - retrieve results from TWO methods. Use the state textfield to retrieve the peak date for the state. Use the month and day field. To retrieve the peak record for that date. See sample results for guidance. **This method is started for you.**
* private void displayStats() – display statistics. You will call four database methods. See sample results for guidance.
* private void displaySafeToOpen() – retrieve safe to open information from the database and display the results. See sample results for guidance.
* private void displayDailyInfections() – **This method is provided for you**
* private void displayTopTen() – retrieve top ten death information from the database and display the results. See sample results for guidance.

**Step 7: Enhance the GUI (5 pts)**

The following should only be attempted after all the other requirements have been completed.

**Check for Valid Input**

The basic GUI assumes that all values provided by the user are valid integers. If not, the program crashes. A better solution is to confirm values are valid integers BEFORE attempting to convert from a String to an integer and then passing to the database

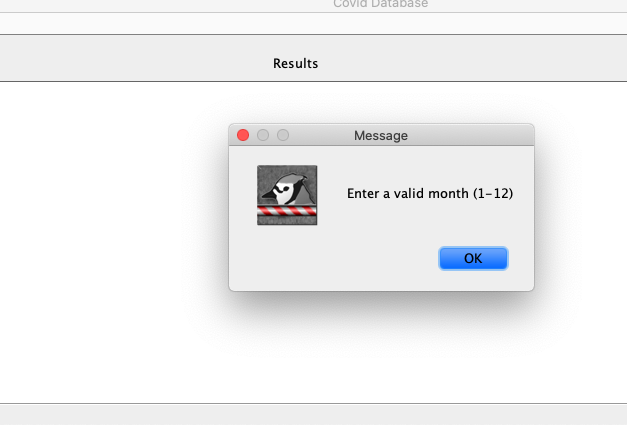
* private boolean isValidMonth() – retrieve text from month field and confirm it is an integer between 1 – 12. Return true if valid. Return false otherwise. **This method has been provided for you.** Review the code of this method for understanding.
* private boolean isValidDay() – retrieve text from day field and confirm it is an integer between 1 – 31. Return true if valid. Do not worry about the moths with less than 31 days. Return false otherwise.
* private boolean isValidState() – retrieve text from state field and confirm it contains a two-letter String. If you feel ambitious, then confirm it contains a valid two-letter state abbreviation, but this is not expected. Return true if valid. Return false otherwise.
* private boolean isValidMinInfections()- retrieve text from minimum field and confirm it is a valid integer. Return true if valid, return false otherwise.

Update each helper method that attempts to convert a text field to an integer (Figure 2). For example:

if (isValidMonth())

// valid integer entered in month text field

// OK to pass as parameter to database



**Figure 2. Sample error message**

**Grading Criteria**

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

**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
* No late projects accepted after the last Friday of classes, 12/11/2020 by midnight.

**Turn In**

1. A Word document or pdf file that includes:
2. Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
3. 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](http://www.cis.gvsu.edu/academic-honesty/).
4. Timecard – 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.”
5. Sample Output – a printout of the Terminal window after running the main method that shows a variety of the printed messages. You can copy and paste into the Word document that contains your cover page.
6. A screenshot after you run the JUnit test class JUnitCovidDatabase.java showing you passed all the test cases.
7. A screenshot of your GUI working
8. Source code – DO NOT PRINT - upload to Blackboard the source code of the following classes:

CovidEntry.java, CovidDatabase.java, CovidDatabaseGUI and CovidDatabaseTest.java