CIS 217 Final Project
A COVID-19 Database

Due Date
Sunday, April 25th, 2021.
Suggested phases:
- Phase 1 – 1 week
- Phase 2 – 2 weeks
- Phase 3 – 1 week

Before Starting the Project
- Read chapter 5 in our Book: JDBC Connectivity
- Review our 1st Project to correctly set up our Junit tests
- Review our notes on Object Orientation and ArrayLists
- Review videos on BB regarding sample DB projects
- 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
- *create, read, and write data* from and to a Database

Project Summary
Create an 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
(March 1 – September 27, 2020)

Phase 1 (20 pts)

Step 1: Create a New IntelliJ Project

Step 2: Download Data File
- Download the "covid_data.csv" file and save it in the folder that IntelliJ created for this new project. There are more than 10,000 entries! You can see its contents from within Windows Explorer or Mac Finder.
- You might want to create a modified file with a few entries to make it easier to test

Step 3: Create a class called CovidEntry

Important Note:

We are providing 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 projec. Follow the same steps we used in our first project to set up IntelliJ. Run the tests and make sure you get all green test results.

Phase 2 (40 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.

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

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

- Make the CovidEntry implement the Comparable class. Implementing Comparable 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 no 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 .

Step 5: Software Testing (5 pts)

To 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.

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

Phase 3 (30 pts)

Step 1: Create a SQLite Database and Insert Data from CSV File into Database

Before we transition our current solution using the csv file to one that reads data from a database, we need to create our database. Our database will have a similar structure to the csv file that we will import. You will need the following structure for your table, ENTRY.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
state	varchar	NO		NULL	
month	int	NO		NULL	
daily_infections	int	NO		NULL	
daily_deaths	int	NO		NULL	
total_infections	int	NO		NULL	
total_deaths	int	NO		NULL	

Our database will have a single table named ENTRY and you will query the database one time to get all data that table is hosting. Please follow the tutorials I have created in Blackboard in order for you to connect your database.

Step 2: Create a helper method to transfer the data from the csv file to the database

Use the following link to get an idea on how to insert data into the database from a csv file.

https://www.codejava.net/coding/java-code-example-to-insert-data-from-csv-to-database

- `private void transferCovidData(String filename)` – reads the file and inserts data into database
    - The code is this method will be very similar to the contents of the link. The only difference will be your database settings and the data that you are inserting into the database

Step 3: Modify readCovidData() Method to read data from the database

- The only method that we will modify is readCovidData. Instead of reading from a file, it will have all the necessary steps to read the data from the database
- `public void readCovidData()` – reads the database and populates the ArrayList of CovidEntry objects.
  - Connect to the database
  - Create prepared statement to query database
  - 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.

Grading Criteria

There is a 50% penalty on programming projects if your solution does not compile. Here are the points for each phase

Phase 1 (20 pts)

Phase 2 (40 pts)
Phase 2 Main Method Testing (5 pts)

Phase 3 (30 pts)
Final Summary PDF Document to be included in Phase 3(5 pts)

Due Dates

- The project has three distinct phases. I have provided some suggested deadlines for each phase. Please let me know if you get stuck at any point of the phases. All phases must be turned in by December 20th.
- Please turn in individual phases in the appropriate Blackboard Assignment.

What Turn In at Each Phase

Phase 1

1. Source Code
   `CovidEntry.java`

2. Screenshot passing all the test from the Junit Test

Phase 2

1. Source Code
   `CovidDatabase.java` and `CovidDatabaseTest.java`

2. Screenshot passing all the test from the Junit Test
3. Screenshot of your terminal after running the main method

Phase 3

1. A Word document or pdf file that includes:
   1) Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
   2) 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.
   3) Timecard – The cover page must also include a brief statement of how much time you spent on the project. For example:
   - I spent 3 hours on this project from April 2-7 reading the book.
   - I spent 4 designing a solution, writing code, and fixing errors
   4) 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.

4. Source code – DO NOT PRINT - upload to Blackboard the source code of the following classes and database:

   `CovidEntry.java` and `CovidDatabase.java` and `CovidDatabaseTest.java` and `Covid.db`