University of New York Tirana

Assignment 1

Statstical Processing: Covid Cases

Professor:
Elton Ballhysa
Advanced Java
Department of Computer Science

Group:
Regi Nishani,
Daniele Llazo

Deadline: 23 January 2022

Contents

1 Introduction

1.1 Target Application

We are asked for this assignment to do some statistical processing on Covid dataset, which is open source. Data must be read through an appropriate data structure, and processed through command line. Despite the fact that this CSV contains 151K lines of data and more than 60 columns, we will only focus on 16 of them. The columns we will focus are:

1. "iso_code (COD)"
2. "continent (CNT)"
3. "location (LOC)"
4. "date (DT)"
5. "total_cases (TC)"
6. "new_cases (NC)"
7. "new_cases_smoothed (NCS)"
8. "total_deaths (TD)"
9. "new_deaths (ND)"
10. "new_deaths_smoothed (NDS)"
11. "reproduction_rate (RR)"
12. "new_tests (NT)"
13. "total_tests (TT)"
14. "stringency_index (SI)"
15. "population (POP)"
16. "median_age (MA)"

The text in parenthesis indicates the code of the field, which we will use in the command line parameters. Apart from the statistics we have in the table, we have to implement a new field, "new_deaths_per_case" (NDPC), equal to new_deaths / new_cases.

We will have 4 filters to implement:

1. The first filter is **"stat:"**, which will be either max or min, depending on input.

2. The second filter is **"limit:"**, an integer between 1 and 100, which will limit the number of data that will be shown on the command line.
3. The third one is **"by:"**, the field on which the statistics will be computed. It can have value NC, NCS, ND, NDS, NT, NDPC.
4. The last one will be **"display:"**, and is the type of data that will be displayed. Allowed values can be DATE, COUNTRY, CONTINENT.

In the command line, we will display all the filters, alongside with the path of the CSV, with a "-" before. We should not use loops, since we have to use lambdas and "Stream API" for iteration purposes. We must use an efficient data structure, such that it supports efficient implementation of the functionality mentioned. Appropriate entities must satisfy data integrity principle accordingly, using the 3NF. Apart from this documentation, we will also submit our Github repository for this assignment.

1.2 Note for the professor

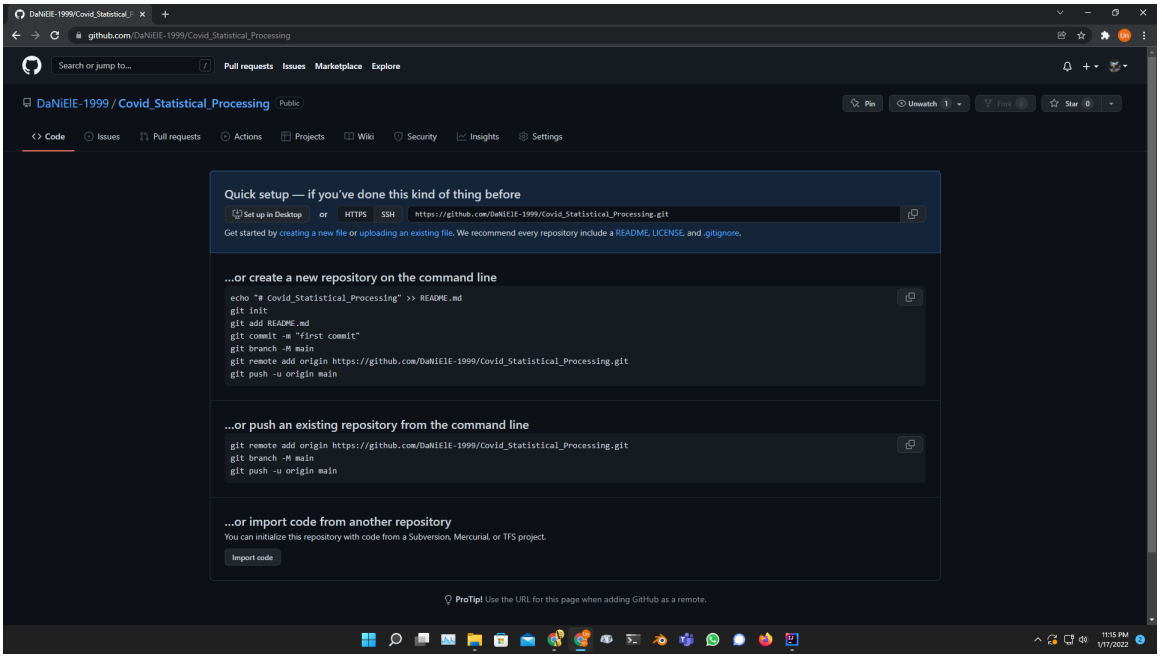
- Please note that lines in the screenshots below are subjects to change, and that reflect the code we wrote prior to formatting.
- Please note that we have used relative path, to get advantage of the .properties file. The examples used an absolute path, but since it was not a requirement, we thought this would still be correct.
- Please also note that we have used "lombok" as external dependency. It makes the code more readable.

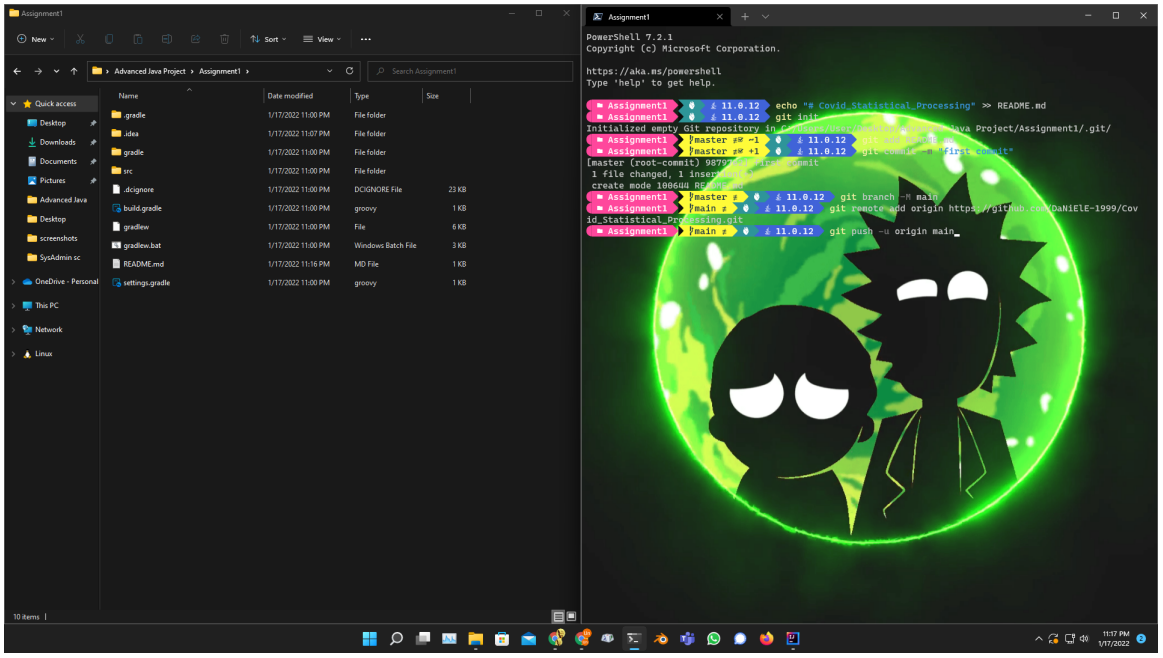
2 Creating the Github Repository

Group assignments are also an opportunity to explore version control tools. The most popular version control tool is Git and Github as a code hosting platform for version control and collaboration. We are going to use them to work together for this assignment.

2.1 Creating Daniele's repository

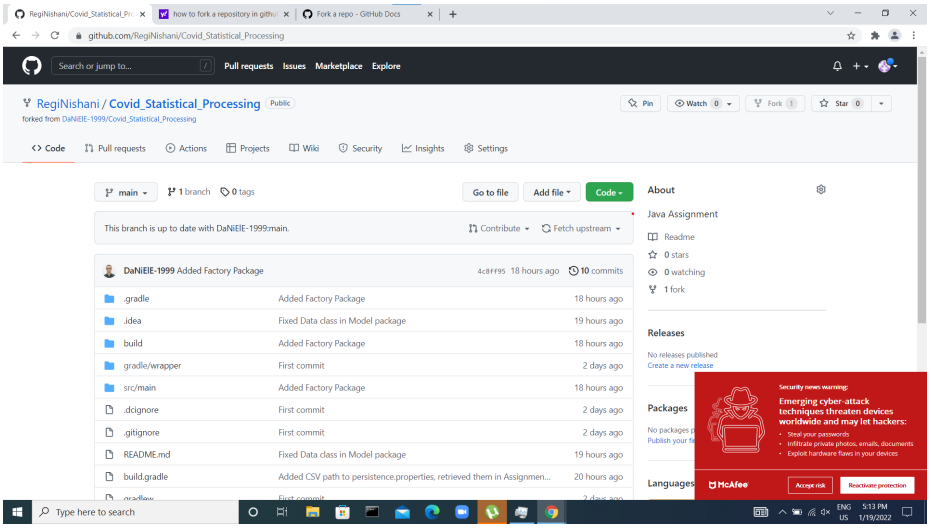
On my account I created a repository named "Covid_Statistical_Processing". On the page that we are redirected, I copy the instructions to my terminal, so that I initialize Git in my working directory. Now I can push the updated code into my repository, or retrieve the code from a previous update. These steps are shown in the figures below.

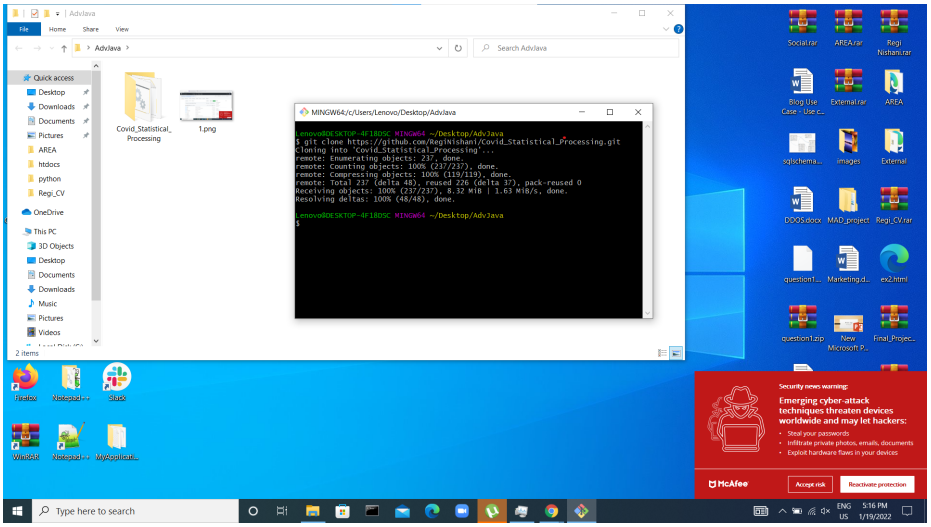




2.2 Creating Regi's repository

I forked Daniele's repository to my Github account. I then cloned my repository to my working directory. These steps are shown below.





Now that we both completed these steps, we may easily work in group. Doing a push request, I may update the code into Daniele's repository.

3 Data Structures

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships." says the inventor of Linux, Linus Torvalds. After reading the data from CSV, we must set them to appropriate entities, and use an efficient data structure. This is the most important task of our project. We have used ArrayList and HashMap for this project.

3.1 3NF

A relation can be in Third normal form if it is in the second normal form and does not have any transitive partial dependency. This form is used to reduce the amount of data duplication. It also increases data integrity. In case there is no transitive dependency for non-prime attributes, it is said to be in the third normal form.

A relation can be said to be in the third normal form if at least one of the following is true for X $\rightarrow$ Y.

1. Y is a prime attribute
2. X is a superkey

3.2 Setting the relations to 3NF

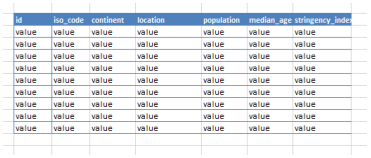
In our case, we get the data from a CSV file, and not a Relational Database. Hence, we will implement the relations in our Java Application. To make our assignment clear, we will divide the column names in an excel table, and than divide this table into two tables, with relation to each other.

iso_code	continent	location	date	total_cases	new_cases	new_cases_smothed	total_deaths	new_deaths	new_deaths_smoothed	reproduction_rate	new_tests	total_tests	stringency_index	population	median_age	new_deaths_per_case
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value	value

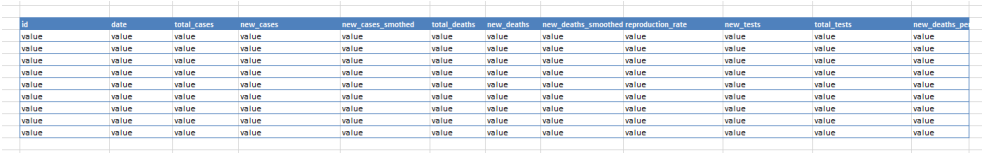
{iso_code, continent, location, population, median_age, stringency_index},
{date, total_cases, new_cases, new_cases_smothed, total_deaths, new_deaths, new_deaths_smoothed reproduction_rate, new_tests, total_tests, new_deaths_per_case}

As we see, we have no candidatee to become a superkey, and all the rest are non prime. This violates the rules of the 3NF. Hence, we will divide this table into 2

relational ones, and on our java application, we will add a Super Key to satisfy the 3NF. This will be the ID, which will be auto-incremented in our application. Our tables should look like this:



id	iso_code	continent	location	population	median_age	stringency_index
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value
value	value	value	value	value	value	value



id	date	total_cases	new_cases	new_cases_smothed	total_deaths	new_deaths	new_deaths_smoothed	reproduction_rate	new_tests	total_tests	new_deaths_pe
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value
value	value	value	value	value	value	value	value	value	value	value	value

Our tables are now in 3NF.

3.3 Java implementation of these tables using POJO

3.3.1 POJO

POJO in Java stands for Plain Old Java Object. It is an ordinary object, which is not bound by any special restriction. The POJO file does not require any special classpath. It increases the readability & re-usability of a Java program.

POJOs are now widely accepted due to their easy maintenance. They are easy to read and write. A POJO class does not have any naming convention for properties and methods. It is not tied to any Java Framework; any Java Program can use it.

The term POJO was introduced by Martin Fowler (An American software developer) in 2000. it is available in Java from the EJB 3.0 by sun microsystem.

Properties:

1. The POJO class must be public.
2. It must have a public default constructor.
3. It may have the arguments constructor.
4. All objects must have some public Getters and Setters to access the object values by other Java Programs.
5. The object in the POJO Class can have any access modifies such as private, public, protected. But, all instance variables should be private for improved

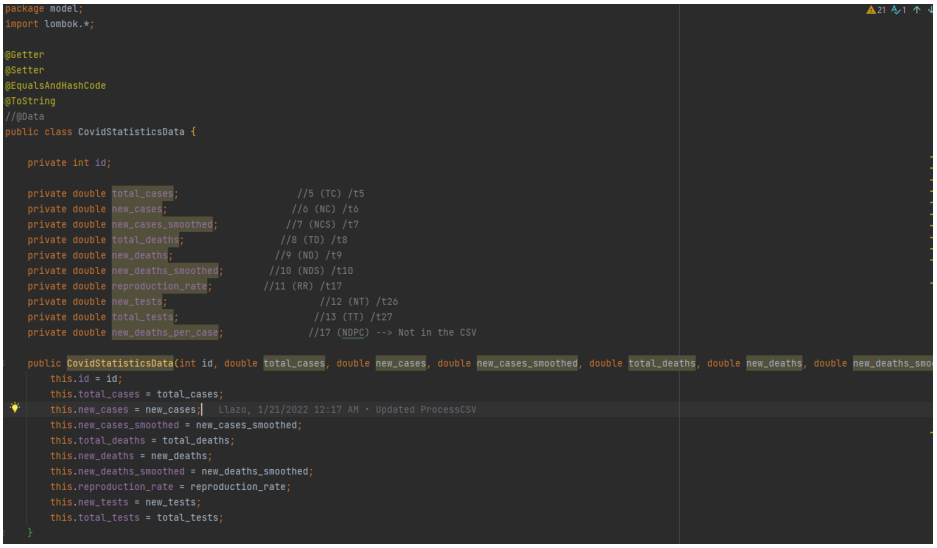
security of the project.

6. A POJO class should not extend predefined classes.
7. It should not implement prespecified interfaces.

3.3.2 Our POJO classes

As mentioned in Section 3.1, we have defined our objects entities. We have a CovidStatisticsData POJO class, and LocationData POJO class. Each object has Getters, Setters, a Constructor, and as a good practice, I have also added EqualsAndHashCode, as well as the ToString Method.

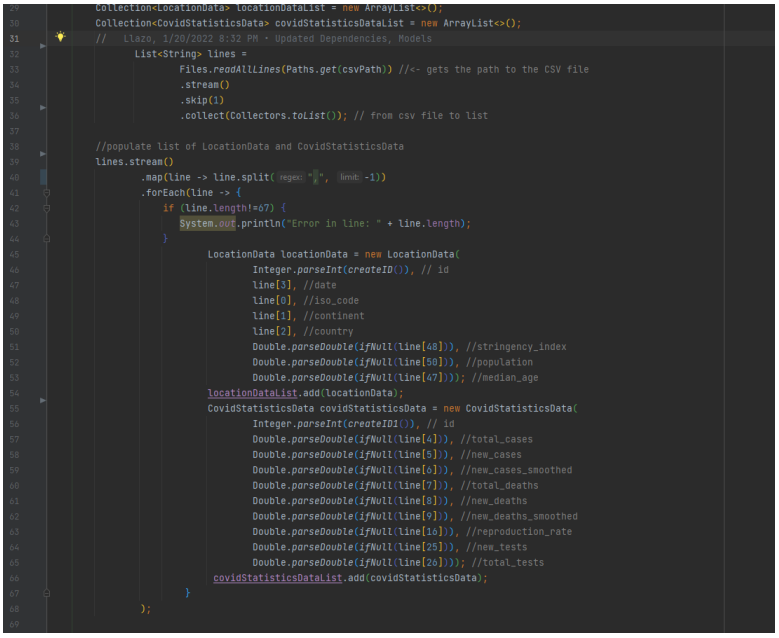
```java
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@EqualsAndHashCode
@ToString

public class LocationData {

    private int id;
    private String Date;                    //1 (COD) /t1
    private String iso_code;                //1 (COD) /t1
    private String continent;               //2 (CNT) /t2
    private String location;                //3 (LOC) /t3
    private double stringency_index;                //14 (SI) /t48
    private double population;                      //15 (POP) /t49
    private double median_age;                      //16 (MA) /t51

    public  LocationData(){

    }
    public LocationData(int id, String Date, String iso_code, String continent, String location, double stringency_index, double population, double median_age) {
        this.id = id;
        this.iso_code = iso_code;
        this.continent = continent;
        this.location = location;
        this.stringency_index = stringency_index;
        this.population = population;
        this.median_age = median_age;
        this.Date = Date;
    }
}
```



```java
package model;
import lombok.*;

@Getter
@Setter
@EqualsAndHashCode
@ToString
//@Data
public class CovidStatisticsData {

    private int id;

    private double total_cases;                    //5 (TC) /t5
    private double new_cases;                     //6 (NC) /t6
    private double new_cases_smoothed;           //7 (NCS) /t7
    private double total_deaths;                //8 (TD) /t8
    private double new_deaths;                 //9 (ND) /t9
    private double new_deaths_smoothed;       //10 (NDS) /t10
    private double reproduction_rate;        //11 (RR) /t17
    private double new_tests;                         //12 (NT) /t26
    private double total_tests;                      //13 (TT) /t27
    private double new_deaths_per_case;             //17 (NDPC) --> Not in the CSV

    public CovidStatisticsData(int id, double total_cases, double new_cases, double new_cases_smoothed, double total_deaths, double new_deaths, double new_deaths_smo
        this.id = id;
        this.total_cases = total_cases;
        this.new_cases = new_cases;  Llazo, 1/21/2022 12:17 AM · Updated ProcessCSV
        this.new_cases_smoothed = new_cases_smoothed;
        this.total_deaths = total_deaths;
        this.new_deaths = new_deaths;
        this.new_deaths_smoothed = new_deaths_smoothed;
        this.reproduction_rate = reproduction_rate;
        this.new_tests = new_tests;
        this.total_tests = total_tests;
    }
```

3.4 Efficient Data Structuring

As seen on the picture below, on lines 29 and 30, we have created two ArrayLists with Objects types, one Object being "LocationData" and the other "CovidStatisticsData". This type of data structure has a time complexity of O(1) in best and worse case scenario. We read all lines from the CSV and then add them to a list, as shown on lines 32-36. On line 39 we initialize a stream of the CSV's lines, then we split it as shown on line 40, using map and lambda expression. We then fill the "locationDataList" and "covidStatisticsDataList" with the respective objects. The objects themselves are created within the loop, using the indexes of the CSV columns.



```
Collection<LocationData> locationDataList = new ArrayList<>();
Collection<CovidStatisticsData> covidStatisticsDataList = new ArrayList<>();
//   Llazo, 1/20/2022 8:32 PM • Updated Dependencies, Models
      List<String> lines =
              Files.readAllLines(Paths.get(csvPath)) //<- gets the path to the CSV file
              .stream()
              .skip(1)
              .collect(Collectors.toList()); // from csv file to list

//populate list of LocationData and CovidStatisticsData
lines.stream()
       .map(line -> line.split( regex: ",", limit: -1))
       .forEach(line -> {
           if (line.length!=67) {
              System.out.println("Error in line: " + line.length);
           }
                 LocationData locationData = new LocationData(
                        Integer.parseInt(createID()), // id
                        line[3], //date
                        line[0], //iso_code
                        line[1], //continent
                        line[2], //country
                        Double.parseDouble(ifNull(line[48])), //stringency_index
                        Double.parseDouble(ifNull(line[50])), //population
                        Double.parseDouble(ifNull(line[47]))); //median_age
                 locationDataList.add(locationData);
                 CovidStatisticsData covidStatisticsData = new CovidStatisticsData(
                        Integer.parseInt(createID1()), // id
                        Double.parseDouble(ifNull(line[4])), //total_cases
                        Double.parseDouble(ifNull(line[5])), //new_cases
                        Double.parseDouble(ifNull(line[6])), //new_cases_smoothed
                        Double.parseDouble(ifNull(line[7])), //total_deaths
                        Double.parseDouble(ifNull(line[8])), //new_deaths
                        Double.parseDouble(ifNull(line[9])), //new_deaths_smoothed
                        Double.parseDouble(ifNull(line[16])), //reproduction_rate
                        Double.parseDouble(ifNull(line[25])), //new_tests
                        Double.parseDouble(ifNull(line[26]))); //total_tests
                  covidStatisticsDataList.add(covidStatisticsData);
               }
       );
```

4 Command Line Arguments

We are required to specify arguments in the command line as:
"-file pathToFile -param1 value1 -param2 value2 ... -paramN valueN."

4.1 Menu object

To be able to get these arguments from the command line, the first thing we did was to create a "Menu" object, which contains all the required parameters, with getters and setters.

```java
import lombok.Setter;

@Getter
@Setter
public class Menu {

    private String status; //max or min
    private Integer limit; // 1 - 100
    private String by; // NC, NCS, ND, NDS, NT, NDPC   Llazo, 1/21/2022 9:58 PM • Tested some functionalities on ProcessCSV
    private String display;
    private String path;
    // DATE, COUNTRY, CONTINENT
    public  Menu() {}

    public Menu(String status, Integer limit, String by, String display, String path) {
        this.status = status;
        this.limit = limit;
        this.by = by;
        this.display = display;
        this.path = path;
    }

    public Integer getLimit() {
        return limit;
    }

    public void setLimit(Integer limit) {
        this.limit = limit;
    }

    @Override
    public String toString() {
        return "-file " + path + " -status " + status + " -limit " + limit + " -by " + by + " -display " + display;
    }
}
```

4.2 Checking the input

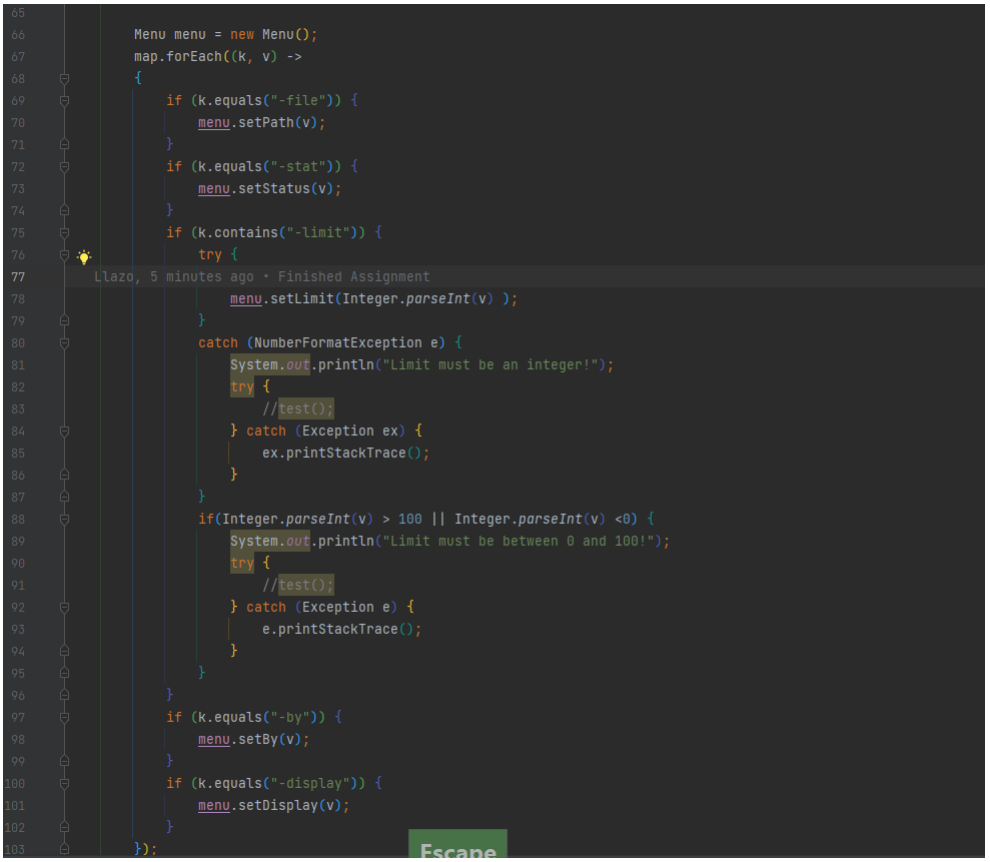
To be able to check the input, we need to consider what can be wrong. We have to check if there are more than 5 parameters, if the format is wrong, or both. All these cases are handled as in the picture below.

```
System.out.println("Please enter the command line parameters\n");
System.out.println("Path must be relative to the Assignment Folder\n");
Scanner sc = new Scanner(System.in);

String s = sc.nextLine();
String[] paramCheck =
        Arrays.stream(s.split( regex: " -"))
                .map(String::trim)
                .toArray(String[]::new);
if (paramCheck.length != 5) {
    System.out.println("Wrong number of parameters!");
}

String[] input =
        Arrays.stream(s.split( regex: " "))
                .map(String::trim)
                .toArray(String[]::new);
if (input.length != 10) {
    System.out.println("Wrong format!");
}

String[] parameters =
        Arrays.stream(s.split( regex: " "))
                //.skip(s.indexOf(1))
                .filter(x -> x.startsWith("-"))
                .map(String::trim)
                .toArray(String[]::new);

List<String> list = new ArrayList<String>(Arrays.asList(parameters));
if (parameters.length != 5)  {
    System.out.println("Wrong number of parameters or incorrect input!");
}
```

4.3 Pairing parameters with their values

We have used a HashMap to pair parameters with their values. This is shown in the first picture below. In the second picture it is shown how we use mapping to insert values in the menu object. We use parameter as key, and value as the inserted value. Here we also catch an exception if limit size is not between 1 and 100, or if a non integer has been entered.

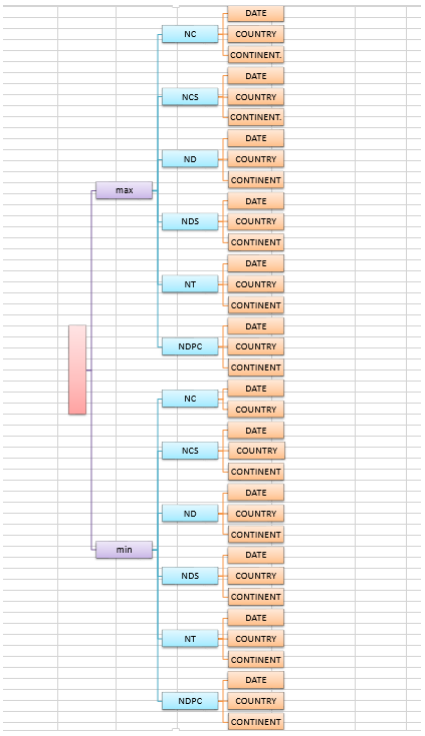
```
HashMap<String, String> map = new HashMap<>();
map.put(list.get(0), input[1]);
map.put(list.get(1), input[3]);
map.put(list.get(2), input[5]);
map.put(list.get(3), input[7]);
map.put(list.get(4), input[9]);
```



```
65
66          Menu menu = new Menu();
67          map.forEach((k, v) ->
68          {
69              if (k.equals("-file")) {
70                  menu.setPath(v);
71              }
72              if (k.equals("-stat")) {
73                  menu.setStatus(v);
74              }
75              if (k.contains("-limit")) {
76                  try {
77      Llazo, 5 minutes ago · Finished Assignment
78                      menu.setLimit(Integer.parseInt(v) );
79                  }
80                  catch (NumberFormatException e) {
81                      System.out.println("Limit must be an integer!");
82                      try {
83                          //test();
84                      } catch (Exception ex) {
85                          ex.printStackTrace();
86                      }
87                  }
88                  if(Integer.parseInt(v) > 100 || Integer.parseInt(v) <0) {
89                      System.out.println("Limit must be between 0 and 100!");
90                      try {
91                          //test();
92                      } catch (Exception e) {
93                          e.printStackTrace();
94                      }
95                  }
96              }
97              if (k.equals("-by")) {
98                  menu.setBy(v);
99              }
100             if (k.equals("-display")) {
101                 menu.setDisplay(v);
102             }
103         });
```

5 Statistics

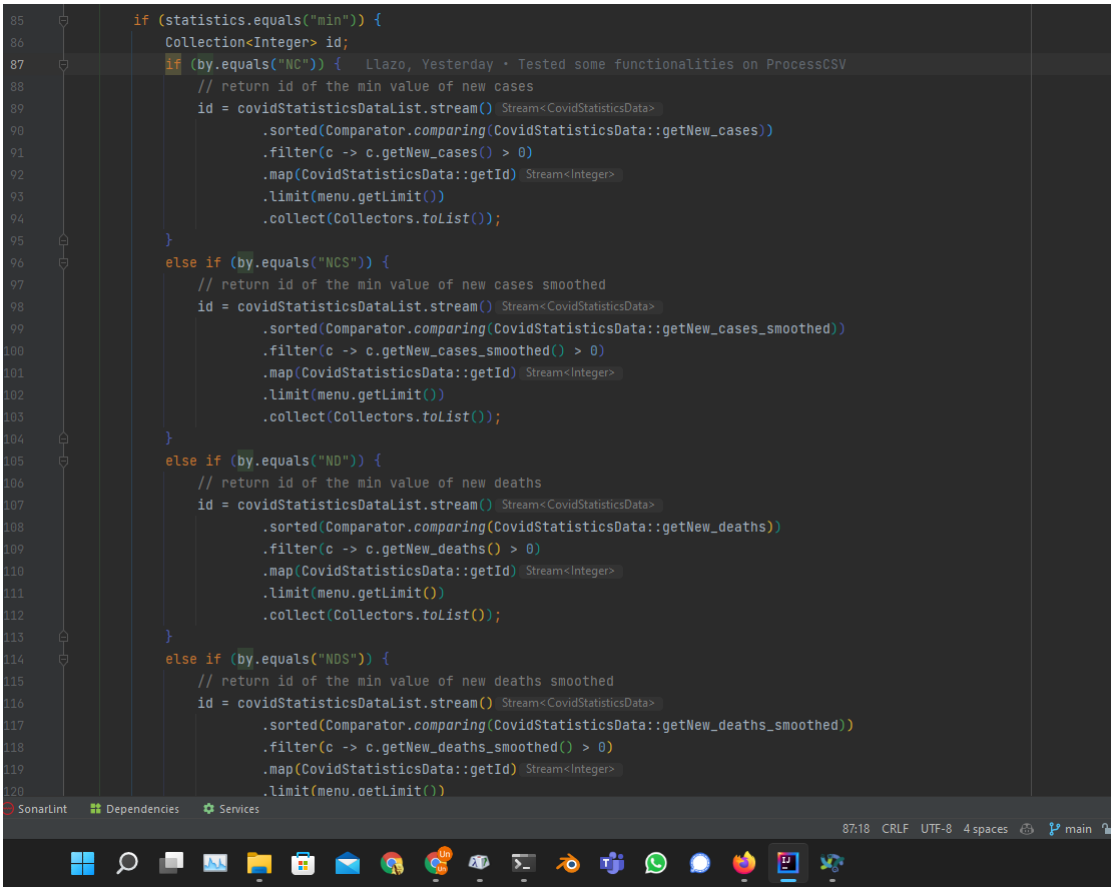
As mentioned in the "Target Application" section in the introduction, we would have to display one of "Date", "Country", "Continent", which have the maximal or minimal value of "NC, NCS, ND, NDS, NT, or NDPC", and then display only between 0 and 100 of the top or bottom values. The relation would be as in the figure below:



5.1 Implementation of the statistics

After user enters the preferences, as shown in the previous section, we have used if/else blocks to display the correct values. A Collection of type Integer is used to store the ID of the requested values to be displayed.

5.1.1 Getting the IDs of minimal statistics



We sort the IDs according to the requested minimal values. We filter the data, so we do not get the negative values, which actually are the CSV fields that do not contain information at all.

5.1.2 Getting the IDs of maximal statistics

```
179
180         else if (statistics.equals("max")) {
181             // return id of the max value of new cases
182             Collection<Integer> id;
183             if (by.equals("NC")) {
184                 // return id of the max value of new cases
185                 id = covidStatisticsDataList.stream() Stream<CovidStatisticsData>
186                         .sorted(Comparator.comparing(CovidStatisticsData::getNew_cases).reversed())
187                         .filter(c -> c.getNew_cases() > 0)
188                         .map(CovidStatisticsData::getId) Stream<Integer>
189                         .limit(menu.getLimit())
190                         .collect(Collectors.toList());
191             }
192             else if (by.equals("NCS")) {
193                 // return id of the max value of new cases smoothed
194                 id = covidStatisticsDataList.stream() Stream<CovidStatisticsData>
195                         .sorted(Comparator.comparing(CovidStatisticsData::getNew_cases_smoothed).reversed())
196                         .filter(c -> c.getNew_cases_smoothed() > 0)
197                         .map(CovidStatisticsData::getId) Stream<Integer>
198                         .limit(menu.getLimit())
199                         .collect(Collectors.toList());
200             }
201             else if (by.equals("ND")) {
202                 // return id of the max value of new deaths
203                 id = covidStatisticsDataList.stream() Stream<CovidStatisticsData>
204                         .sorted(Comparator.comparing(CovidStatisticsData::getNew_deaths).reversed())
205                         .filter(c -> c.getNew_deaths() > 0)
206                         .map(CovidStatisticsData::getId) Stream<Integer>
207                         .limit(menu.getLimit())
208                         .collect(Collectors.toList());
209             }
210             else if (by.equals("NDS")) {
211                 // return id of the max value of new deaths smoothed
212                 id = covidStatisticsDataList.stream() Stream<CovidStatisticsData>
213                         .sorted(Comparator.comparing(CovidStatisticsData::getNew_deaths_smoothed).reversed())
214                         .filter(c -> c.getNew_deaths_smoothed() > 0)
215                         .map(CovidStatisticsData::getId) Stream<Integer>
216                         .limit(menu.getLimit())
217                         .collect(Collectors.toList());
```

We sort the IDs according to the requested maximal values. This is done by sorting them in reverse. We filter the data, so we do not get the negative values, which actually are the CSV fields that do not contain information at all.

5.1.3 Displaying the data

```
241
242             if (typeOfRecords.equals("DATE")) {
243                 locationDataList.stream()
244                         .filter(locationData -> {
245                             assert id != null;
246                             return id.contains(locationData.getId());
247                         })
248                         .map(LocationData::getDate)
249                         .forEach(System.out::println);
250             }
251             else if (typeOfRecords.equals("COUNTRY")) {
252                 locationDataList.stream()
253                         .filter(locationData -> {
254                             assert id != null;
255                             return id.contains(locationData.getId());
256                         })
257                         .map(LocationData::getLocation)
258                         .forEach(System.out::println);
259             }
260             else if (typeOfRecords.equals("CONTINENT")) {
261                 locationDataList.stream()
262                         .filter(locationData -> {
263                             assert id != null;
264                             return id.contains(locationData.getId());
265                         })
266                         .map(LocationData::getContinent)
267                         .forEach(System.out::println);
268             }
269             else {
270                 System.out.println("Invalid input");
271             }
272         }
273         else {
274             System.out.println("Invalid input");
275         }
276
```

Using the IDs we have gathered in the Collection, we use them to display the data that the user has requested us to display. Here we also add the limit value, as the user should only request a number of records between 1 and 100.

6 Conclusion

All in all, we have been able to finish this assignment. We have been able to read the data from the CSV, defined the right entities using POJO objects, by satisfying the 3NF principles of the data integrity. We have used the appropriate data structure, been able to retrieve the data from the command line from the requested input format. All of these has been implemented without loops, as requested, by only using the Stream API and lambdas. We have worked together by using Git and Github, and also provided you the Github repository.

7 References

1. www.javatpoint.com. (n.d.). POJO in Java - Javatpoint. [online] Available at: https://www.javatpoint.com/pojo-in-java

2. GeeksforGeeks. (2019). Third Normal Form (3NF) - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/third-normal-form-3nf/.

3. Basics of the Unix Philosophy. (2003) - Eric Raymond's "Rule of Representation" from 2003. Available at: http://www.catb.org/ esr/writings/taoup/html/ch01s06.html.