Assignment 1 Documentation

Work by: Anxhelo Kapaj & Fatmir Kondi

Class: Advanced Java – Fall21

The assignment

The assignment was a difficult one which was made specifically to challenge our knowledge on the lessons we have done so far. With a culmination of every knowledge and a little bit of imagination, the challenge was presented in such a way that forced you to think outside the box and reimagine every piece of information placed in a way, in a new puzzle.

When read, few things jumped out at first. The selection of the columns needed, the custom statistic, new\_deaths\_per\_case (NDPC), the input of the user and the non-functional requirements. These conditions, at first created a simple looking puzzle on my head, where I could see different ways to fit the pieces together, but the feeling of missing the links needed was still there. For example, looking at the input I realised that I had to do it using flags or arguments. Flags were difficult so we decided to use arguments (Details will be explained further below). Not using loops meant we had to use streams and find ways to do it like that, rather than our conventional way of thinking. In the end we came up with this code.

The code explanation

There are 2 classes on this Project. The first being the main, and the second being the Object class which is called DataLine.

On the main class, we first declared the variables where we would store the values given from the user. The variables are as follows:

Limit -> here we will put how many lines of data we want to display

By -> here the code of the column will be stored

PathToFile -> the path of the file

Stat -> it will be either min or max

Display -> it will store values DATE, CONTINENT or COUNTRY

Then we have the input of the user. This input is taken using a scanner and then its split into an array of type strings using .split(“ ”). This splits our input into parts every time there is a space in the line.

After this we check if the user has used flags on his input or not. This is done by taking 2 things into consideration:

1. The user inputs the parameters in the same order every time
2. The user can choose to input flags or not.

If the length of the array is more than 5, which means the user has used flags on his input, then we take different parts of the array for the variables. If he has only placed the values he wants, then we use different parts.

Example:

-file D:/data/owid-covid-data.csv -stat max -limit 5 -by NC -display DATE

This line has 10 pieces of parameters on it, and they are ordered

1 2 3 4 5 6 7 8 9 10

Meanwhile this code has 5 parts

D:/data/owid-covid-data.csv max 5 NC DATE

1 2 3 4 5

So, the program checks what type of input you have chosen and then changes our variables according to the case.

Afterwards, we read both the full file, and the first row only. The first row is read because we will use that row which contains the columns as a way to find their index.

The file is placed into an Array List of type String where each element is a full row of the CSV file called inputCSV. The first row is also placed into an Array List of type String but its content is already split and stored into this array.

Then we have a piece of code that stores the indexes of our columns and is flexible in case their order changes. An example would be if columns “Total Cases” is not on the second place so, columns[1] but is on the 4th, the variable colIndTC would still find the correct position and not refer another column of the file.

Afterwards we have a map called newMap where we use our inputCSV list and stream it.

We skip the first row because we have already read that line and we only need the data on this map, not the name of the column where its stored. Then we split every line of this list, because they are not split. Every value of the array inputCSV is a long line of the CSV file all together. In the .split command we have used “,” which means we will split every word whenever there is a comma. We have also used a value “-1”. This value shows that whenever there is an empty space on the CSV file like “Asia,,,,,10,,,” the stream won’t just skip them but actually store them in the stream. This way the order of the columns will be maintained and it will not be broken. Example the 10 value will not be the second on the array but the 15th.

Then we map all of the required parts into our object using the map function. Some of the values have to be parsed into float, while others can be sent over to the object as they are. The values are taken using the indexes from the columns list that we created specifically from the first row only. This way we know that the value we are taking are from the DATE column and not from the New Cases column. Some of the values that we are passing might be empty, and while values that are string like Continent are not a problem (this column being empty will be used later on during filtering) , values that are integers or float have to be 0 instead of empty. This is important because we will use these values in sorting and empty ones will create errors. As for the keys of the map, the iso-code and the date are used together. If it was just the iso code or the date, the values that were the same would overwrite each other and the sorting/filtering would prove useless.

The DataLine class is a class specifically for the object DataLine. This objects takes all of the parameters specified in the assignment part. So,

iso\_code (COD), continent (CNT), location (LOC), date (DT), total\_cases (TC), new\_cases(NC), new\_cases\_smoothed (NCS), total\_deaths (TD), new\_deaths (ND), new\_deaths\_smoothed (NDS), reproduction\_rate (RR), new\_tests (NT), total\_tests (TT), stringency\_index (SI), population (POP), median\_age (MA) and has the appropriate getters and setter plus the constructor.

On the constructor we also have a piece of line of code that is related to the new data,

New deaths per case or (NDPC). On this line we check firstly if the NewCases is 0 then we automatically place 0 otherwise we do the division. This is because division by 0 is not allowed and if there are no new cases then there are no deaths.

Lastly on the code we have the switch code that has all of the different cases related to our input. The first switch is if we use min or max, the second switch is depending on the column we want the data, NewCase, NewCasesSmoothed etc. And the third switch case is wether we want to display DATE, COUNTRY or CONTINENT.

The filters used here are as follows:

On the COUNTRY case, we want to filter out all the rows that the Continent field is empty. This way we will only have the states and not continents or fields like UPPER MIDDLE INCOME etc.

On the CONTINENT case, we have a filter that checks the location column and matches it to one of the 6 continents included in the list (Antarctica does not have covid cases so its not on the list).

As for the sorting of the material we use the comparator to compare the integer values and sort them. If the value is min the compare variable will sort them automatically from low to high, and if we use .reversed() at the end of the compare variable then we can sort them from high to low, which is used on the case that the user inserts max.

As for the things that we can improve from this project, it would be removing unnecessary lines of code like setters which are never used. Using a way to make sure that the flags and their value are correctly positioned in the input, so if the order changes the program still works the same. Maybe use a database so the code is much easier to use and instead of 3 nested switches we can use a simple query so we can get what we want.