**Java test**

This test is intended to observe your skills in **I/O, Stream, Optional, Collection, Collector, Lambda expressions, and programming logic, focused on data analysis/processing, using Java Core 8**, immersed in an activity that summarizes the main task that the developer faces in this team.

As previously explained, our processes are based on reading files **(.csv, .xml, .xlms, .txt, ...)**, filters and/or modifications to these data, merging data between different files, and at the end as output a new text file (or an update of an existing file) in a specific format. We always have to deliver the return of the work as quickly as possible, applying the best possible logic, consuming as little memory as possible, like a balance game, seeking the best of both worlds.

The initial structure of the project has already been generated. Try to apply the business rules of the tasks using what was developed (this will save time). But you are open to modifying everything and creating new things if you feel it is necessary, but that it makes sense and that you know how to explain why. You can add new Maven dependencies, if you feel the need, or if you have something to present in your solution, but what is already available is sufficient for the task.

**The code was initially developed by a junior developer. He still doesn't know the concepts well, and mistakes may have been made, so revalidate and refactor, if necessary, what has already been developed.** You need to finish developing the code he has already started.

**Remember: We are going to evaluate I/O, Stream, Optional, Collection, Collector, Lambda expressions, and programming logic, focused on data analysis/processing, using Java Core 8, so we expect you to use these concepts as a mid/senior developer!**

Ex:

*List<Object> list = .... ;*

**- Dev Junior**

*for( int i = 0; i < list.size(); i++ ) {*

*System.out.println(list.get(i));*

*}*

**- Dev Mid**

*list.forEach(d -> System.out.println(d));*

**- Dev Senior**

*list.forEach(System.out::println);*

**Starting tasks**

1. ***Read the file “people.csv”***

The **“people.csv”** file can be found at “**src/main/resources/people.csv**”.

The file contains the columns:

**ID,NAME,GENDER,AGE,DATE,COUNTRY**

Were, mandatory:

1. **ID,** and **AGE**: are integers
2. **NAME**, **GENDER, DATE,** and **COUNTRY**: are strings

Follow the mandatory data type (information validation) that must contain each column, and if the data type is invalid, ignore the line. Remember that lines with empty text data (columns with empty information) or with the value “null” must also be ignored. Rows should only have 6 columns, rows with fewer or more columns should be ignored as well. Duplicate lines must be removed, keeping only valid information.

1. ***Read the file “customers.csv”***

The **“customers.csv”** file is located in **“src/main/resources/customers.csv”**.

The file contains the columns:

**Index,Customer Id,First Name,Last Name,Company,City,Country,Phone 1,Phone 2,Email,Subscription Date,Website**

In this case, all data types for every column in that file have already been validated. You'll only need to extract the **Company**, **Country**, and **Email** columns, no worries. All of them are in text format and are not null or empty.

1. ***Merging the files***

We need to merge the two tables generated by reading the files. Return all records from the left table (**people**), where age is greater than or equal to 18, and the corresponding records from the right table (**customers**), where the key is the column **people.COUNTRY = customers.Country**.



Ex:

- Table1 (KEY, NAME):

1, Silva

2, Miranda

3, Oliveira

- Table2 (KEY, JOB):

1, Dev

1, Ops

3, Msg

- JOIN per KEY (KEY, NAME, JOB):

1, Silva, Dev

1, Silva, Ops

2, Miranda, (*or 2, Miranda, “null” / or 2, Miranda, null / or 2, Miranda, ”” )*

3, Oliveira, Msq

1. **Writing the merging of tables**

Now we need to save the new table in an output file **“output.csv”**. It must contain the columns from the **people** **file** and the other 3 mapped from the **customers file**. If the output file contains data, it must be deleted, and new data added. Don't forget the header (use the name **COUNTRY** for the merge key column).

1. ***Getting data from file “people.csv”***

Read (again if necessary) the **“people.csv” file**. Find the first line that in the column **“COUNTRY”** is the value **“France”**, replace the value to **“Portugal”**, and then **print it on screen**.

“**System.out.println(PersonModel.toString())”**

If you don't find any column with the expected value, print “**PersonModel{id=0, name='', gender='', age=0, date='', country=''}”**