CMPS 251 Tutorial 4 – Fall 2019

Exercise 1 – Manipulating a Stream<Invoice>

Create a package named `tutorial4.excercise1` to place all the classes of this exercise.

Create a Class `Invoice` having four attributes: **`itemNumber`** (type String), **`itemDescription`** (type String), **`quantity`** of the item being purchased (type int) and **`price`** (type double). Add getter and setter methods and a constructor to initialize the attributes of this class.

Use the code @ https://gist.github.com/erradi/34d6d26ff4d61c5106979deb8110fcd8 to create the Invoice class.

Create App class with a main method. Inside the main declare **`invoices`** variable to store a list of Invoice objects using the code @ https://gist.github.com/erradi/dc85fe8189c46e30b8c244ce496f7bc0

Perform the following queries on the list of Invoice objects and display the results:

a. Use streams to sort the **`Invoice`** objects by **`itemDescription`,** then display the results.
b. Use streams to sort the **`Invoice`** objects by **`price`,** then display the results.
c. Use streams to map each **`Invoice`** to a string concatenating the **`itemDescription`** and **`quantity`,** sort the results by **`quantity`,** then display the results.
d. Use streams to map each **`Invoice`** to a string concatenating the **`itemDescription`** and the **Invoice amount** (i.e., **quantity * price).** Order the results by **`Invoice`** amount.
e. Modify Part (d) to select the **`Invoice`** amounts in the range $200 to $500.
f. Find any one **`Invoice`** in which the **`itemDescription`** contains the word "saw".

Exercise 2 – Generate and process a list of random numbers

Create a package named `tutorial4.excercise2` to place all the classes of this exercise.

Use **`ints`** method of an instance of **`Random`** class to generate a stream of random numbers in the range 1 to 999.

Tip:

```
Random generator = new Random();
IntStream randomStream = generator.ints(1, 999);
```

Then **`limit`** the stream to 50 elements and **`collect`** the elements into a List on Integers.

Tip:

```
//IntStream.boxed converts an IntStream into a Stream<Integer
List<Integer> numbers =
          randomStream.limit(50).boxed().collect(Collectors.toList());
```

Then compute the followings:

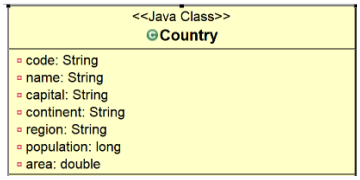
- Count of odds numbers
- Count of evens numbers
- Average of odd numbers
- Average of evens numbers
- Average of all numbers

(note the average values should be `double`).

Exercise 3 – World Atlas

Create a package named `tutorial4.excercise3` to place all the classes of this exercise.

1. First, create a subfolder named **data** under your project's root folder. Then download this file https://www.dropbox.com/s/eyljz0so1mn2wll/countries.json?dl=0 to it.
2. Create Country classes as shown in the class diagram below:



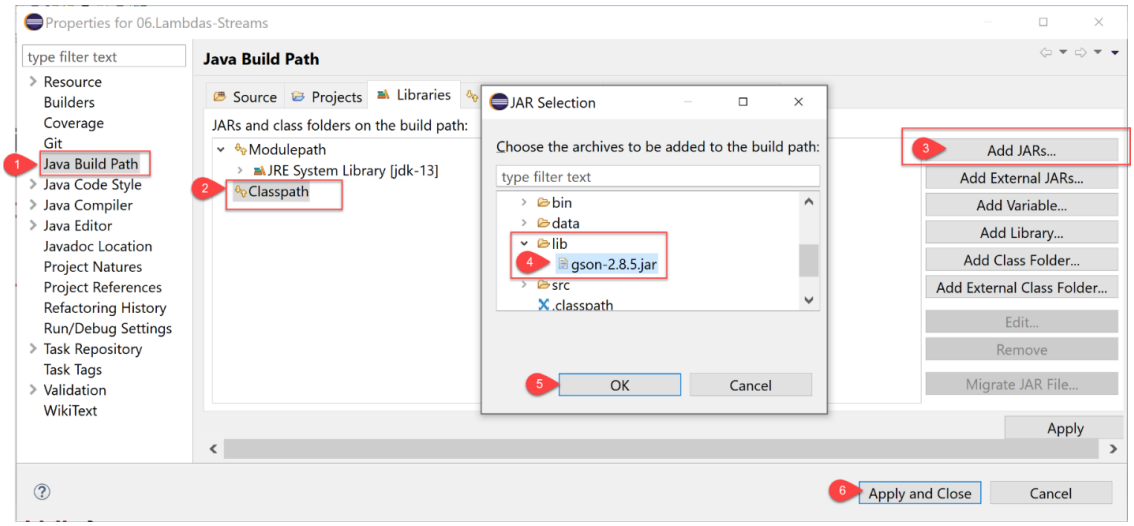
- Generate getters and setters and a constructor for this classes.
- Add toString() method to return:

```
return String.format("%-20s | %-8s |%-18s | People: %,14d | Area: %,10.2f", name, continent, region, population, area);
```

3. Create **CountryAtlas** class and add static private attribute to it `countries : List<Country>`
4. Add **`loadCountries()`** to load the data from `countries.json` file into `countries` list.

To ease implementing these methods you can first complete these steps to add **`Gson`** library to your project:

- First create a subfolder named **lib** under your project folder.
- Download **Gson** library into **lib** subfolder https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.5/gson-2.8.5.jar
- Write-click your project and select **Properties…**
- Select ***Java Build Path***. Click ***Classpath*** then click ***Add JARS…*** select ***gson-2.8.5.jar*** from your project **lib** subfolder (see image below)



Loading the data from a json file to a list requires 2 steps:

(1) Read data from the json file to a string:

```
String filePath = "data/countries.json";
String fileContent = Files.readString(Paths.get(filePath));
```

(2) Convert the json string to an array objects using Gson class:

```
Gson gson = new Gson();
Country[] countriesArray = gson.fromJson(fileContent, Country[].class);
countries = new ArrayList<>(List.of(countriesArray));
```

After loading the countries, remove the ones where the **`continent`** is empty or the **`population`** = 0.

5. Add a main method to call and test **CountryAtlas** methods as you develop them.
6. Implement and test the following **CountryAtlas** methods using lambdas and streams:

`getCountry(name: String): Country`	Return the matching country. [**test it by calling it and displaying the results in the main method**].
`getCountries(continent: String): List<Country>`	Returns countries by continent sorted by population in descending order. Test it by calling it in the main method with `"Asia"` as a parameter, limit the results to 5 countries and display the results.
getCountriesByRegion(String): List<Country>	Returns countries by region sorted by population in ascending order. Test it by calling it in the main method with `"Western Asia"` as a parameter, limit the results to 5 countries and display the results.
getCountryCountByContinent(): Map<String, Long>	Group countries by continent and count countries per continent. Return the results as a Map: the key is the continent name and the value is the count of the countries in the continent. Test it by calling it in the main method and display the results.
getPopulationByContinent(): Map<String, Long>	Group countries by continent and sum the population per continent. Return the results as a Map: the key is the continent name and the value is the sum of the population of the countries in the continent. Test it by calling it in the main method and display the results.
getPopulousCountry(): Country	Get the country with the highest population. Test it by calling it in the main method and display the results.
getLeastPopulatedCountry(): Country	Get the country with the leased population. Test it by calling it in the main method and display the results.
getLargestCountry(): Country	Get the country with the biggest area. Test it by calling it in the main method and display the results.
getSmallestCountry(): Country	Get the country with the smallest area. Only consider the countries where the area is > 0. Test it by calling it in the main method and display the results.