Develop an application for managing the information about mountain huts in a given region.

In addition to the mountain huts, the application must allow to insert the information about altitude ranges and municipalities.

All classes are inside the package mountainhuts.

(*The goal for this lab is to use the Stream API*)

**R1 - Altitude ranges**

All interactions are through the class Region. The method getName() of Region returns the name of the region as it was specified in the constructor.

Huts are classified according to their altitude range, and such ranges could be freely defined according to the specific needs. Altitude ranges are defined through the method setAltitudeRanges() that gets as a parameter an array of strings. Each string describes an altitude range in the format "[minValue]-[maxValue]". E.g., the range "0-1000" represents altitudes from 0 to 1,000 meters above sea level with the upper level inclusive. Ranges may be assumed non overlapping.

The method getAltitudeRange() gets as a parameter an altitude and returns the string describing the range that contains the altitude among the ranges defined through setAltitudeRanges(). If no range includes the altitude, the method should return the default string "0-INF".

**R2 - Municipalities and mountain huts**

Municipalities are defined using the factory method createOrGetMunicipality() that gets as parameters the unique name of the municipality, the province, and its altitude. The method returns an object of class Municipality. If a municipality with the same name already exists, the method shall return it, ignoring the remaining parameters.

Mountain huts are created using the factory method createOrGetMountainHut() that gets as parameters the unique name of the hut, its category, number of beds, and the municipality where it is located. The method createOrGetMountainHut() also accept an optional parameter that specifies the altitude of the hut. The method returns an object of the class MountainHut. If a hut with the same name already exists, the method shall return it, ignoring the remaining parameters.

The class Municipality and the class MountainHut shall also implement all obvious getters and setters. The method getAltitude() in the class MountainHut returns an Optional that is empty if the altitude of the hut was not specified in createOrGetMountainHut().

The collections containing the names of municipalities and the names huts are available through the methods getMunicipalities() and getMountainHuts(), respectively.

Hints:

* The class Optional is used to explicitly indicate a value that may not exist. The method isPresent() is used to check if a value is available in the optional.
* To create an Optional from a variable that might be null it is possible to use Optional.ofNullable() that returns an Optional wrapping the variable, or an empty Optional if the variable is null.

**R3 - Input from CSV**

The static factory method fromFile() creates an object of class Region using the information stored inside a file whose name is passed as an argument. In more details, the method should populate the region with both the municipalities and the huts, described in a CSV file that is structured as follows:

| **N** | **Columns** | **Municipality** | **MountainHut** |
| --- | --- | --- | --- |
| 0 | Province | ✓ |  |
| 1 | Municipality | ✓ |  |
| 2 | MunicipalityAltitude | ✓ |  |
| 3 | Name |  | ✓ |
| 4 | Altitude |  | ✓ |
| 5 | Category |  | ✓ |
| 6 | BedsNumber |  | ✓ |

Note: the file contains a line for each hut, therefore the information about municipalities may be duplicated.

CSV fields are separated by a semicolon (;). The altitude of a hut is empty if the information is not available.

All data about mountain huts in Piedmont are available in the file: data/mountain\_huts.csv(\*).

Hints:

* To read from a CSV file you can use the provided method readData(), which reads a text file line by line, and returns a list of rows. The first rows contains the headers, while the actual data starts from the second row.

**R4 - Queries**

The method countMunicipalitiesPerProvince() shall return a map with the name of the province as key, and the total number of the municipalities of that province as value.

The method countMountainHutsPerMunicipalityPerProvince() shall return a map with the name of the province as key and as value a second map with the name of the municipality as key, and the number of mountain huts located inside that municipality as value.

The method countMountainHutsPerAltitudeRange() shall return a map with the altitude range returned by getAltitudeRange() as key, and the number of huts in that altitude range (inclusive) as value. When no altitude is specified for the hut, do consider the altitude of the municipality.

The method totalBedsNumberPerProvince() shall return a map with the name of the province as key, and the total number of beds available in all huts located in that province as value.

The method maximumBedsNumberPerAltitudeRange() shall return a map with the altitude range returned by getAltitudeRange() as key, and as value the maximum number of beds available in a single hut in that altitude range (inclusive). When no altitude is specified for the hut, do consider the altitude of the municipality.

The method municipalityNamesPerCountOfMountainHuts() shall return a map with the number of available huts as key, and a list of the municipalities including exactly that number of huts as value. The list should be alphabetically sorted.

To implement the queries, usage of Stream API is recommended; they allow writing more compact and understandable code, with respect to explicit iterations on collections and maps.

(\*) the file contains a simplified version of the data available on the open data portal of the Piedmont region, in particular <https://www.dati.piemonte.it/#/catalogodetail/regpie_ckan_ckan2_yucca_sdp_smartdatanet.it_RifugiOpenDa_2296>