Milestone overview

The first milestone consists in extracting meaningful information from the dataset. The methods to implement live in the **Extraction.scala** file.

You are given several .csv files containing two kinds of data:

- Weather station's locations (**stations.csv** file) ;
- Temperature records for a year (files **1975.csv**, **1976.csv**, etc.).

Your goal is to merge the data from these sources to get a series of information of the form date × location × temperature.

You can monitor your progress by submitting your work at any time during the development of this milestone. Your submission token and the list of your graded submissions is available on this page.

Data files

The data you will use comes from the National Center for Environmental Information of the United States.

Our files use the comma-separated values format: each line contains an information record, which is itself made of several columns.

Stations

The **stations.csv** file contains one row per weather station, with the following columns:

STN identifier	WBAN identifier	Latitude	Longitude

You might have noticed that there are two identifiers. Indeed, weather stations are uniquely identified by the compound key **(STN, WBAN)**.

Note that on some lines some columns might be empty. Let's illustrate this with the following excerpt:

```
1  010013,,,
2  724017,03707,+37.358,-078.438
3  724017,,+37.350,-078.433
```

Here, the first line describes a station whose STN identifier is 010013, with no WBAN identifier and no GPS coordinates. The second line describes a station whose STN identifier is 724017, WBAN identifier is 03707, latitude is 37.358 and longitude is -078.438. Finally, the third line describes a station whose STN identifier is (again) 724017, WBAN identifier is missing, latitude is 37.350 and longitude is -078.433.

Temperatures

The temperature files contain one row per day of the year, with the following columns:

STN identifier	WBAN identifier	Month	Day	Temperature (in degrees Fahrenheit)

The STN and WBAN identifiers refer to the weather station's identifiers. The temperature field contains a decimal value (or 9999.9 if missing). The year number is given in the file name.

Again, all columns are not always provided for each line. Here is an hypothetical excerpt of such files:

```
1  010013,,11,25,39.2
2  724017,,08,11,81.14
3  724017,03707,12,06,32
4  724017,03707,01,29,35.6
```

Here, the lines respectively indicate that:

1. The average temperature was 39.2 degrees Fahrenheit on November 25th at the station whose STN identifier is 010013.
2. The average temperature was 81.1 °F on August 11th at the station whose STN identifier is 724017.
3. The average temperature was 32 °F on December 6th at the station whose WBAN identifier is 03707.
4. At the same station, the average temperature was 35.6 °F on January 29th.

Data extraction

To make our method signatures as clear as possible, we've introduced the following global type aliases in **package.scala**:

```
1  type Temperature = Double // °C
2  type Year = Int
```

In this project, **Temperature** will always represent a (type **Double**) number of °C. We're also providing you with a **case class** for location, defined in **models.scala** as:

```
1  case class Location(lat: Double, lon: Double)
```

You will first have to implement a method **locateTemperatures** with the following signature:

```
def locateTemperatures(
  year: Year,
  stationsFile: String,
  temperaturesFile: String
): Iterable[(LocalDate, Location, Temperature)]
```

This method should return the list of all the temperature records converted in degrees Celsius along with their date and location (ignore data coming from stations that have no GPS coordinates). You should not round the temperature values. The file paths are resource paths, so they must be absolute locations in your classpath (so that you can read them with getResourceAsStream). For instance, the path for the resource file 1975.csv is **/1975.csv**.

With the data given in the examples, this method would return the following sequence:

```
Seq(
  (LocalDate.of(2015, 8, 11), Location(37.35, -78.433), 27.3),
  (LocalDate.of(2015, 12, 6), Location(37.358, -78.438), 0.0),
  (LocalDate.of(2015, 1, 29), Location(37.358, -78.438), 2.0)
)
```

In order to study the climate we want to remove the variations due to seasons. So, we want to compute the average temperature, over a year, for every station.

To achieve that, you will have to implement the following method:

```
def locationYearlyAverageRecords(
  records: Iterable[(LocalDate, Location, Temperature)]
): Iterable[(Location, Temperature)]
```

This method should return the average temperature at each location, over a year. For instance, with the data given in the examples, this method would return the following sequence:

```
Seq(
  (Location(37.35, -78.433), 27.3),
  (Location(37.358, -78.438), 1.0)
)
```

Note that the method signatures use the collection type **Iterable**, so, at the end, you will have to produce such values, but your internal implementation might use some other data type, if you think that it would have better performance.

Mark as completed