**CT3535 Assignment B3**

**Source Code:**

**import** java.util.\*;

**public** **class** WeatherStation {

//Initialising variables city, measurements, & stations

String city = **null**;

ArrayList<Measurement> measurements = **new** ArrayList<Measurement>();

**static** ArrayList<WeatherStation> *stations* = **new** ArrayList<WeatherStation>();

//overloaded constructor for WeatherStation

**public** WeatherStation(String city) {

**this**.city = city;

}

//adds WeatherStation station to the stations arraylist

**void** addWeatherStation(WeatherStation station) {

*stations*.add(station);

}

//adds a measurement class to the measurements arraylist

**void** addMeasurement(**int** time, **double** temperature) {

measurements.add(**new** Measurement(time, temperature));

}

//creating the Measurement class

**static** **class** Measurement{

//Initialising variables

**int** time;

**double** temperature;

//overloaded constructor

**public** Measurement(**int** time, **double** temperature) {

**this**.time = time;

**this**.temperature = temperature;

}

//accessors & mutators

**public** **void** setTime(**int** time) {

**this**.time = time;

}

**public** **void** setTemp(**double** temperature) {

**this**.temperature = temperature;

}

**public** **int** getTime() {

**return** **this**.time;

}

**public** **double** getTemp() {

**return** **this**.temperature;

}

}

//avgTemperature function

**double** avgTemperature(**int** startTime, **int** endTime) {

/\* THOUGHT PROCESS:

creates a double integer, tempList,

which is an addition of all double integers, i,

which is the getTemp accessor (this is integrated using

lambda equations).

it is filtered through all the numbers

larger than or equal to startTime

and smaller than or equal to endTime

(I wasn't sure whether the startTime & endTime was to be included).

the tempList double integer is then divided

by the total count of all measurement instances.

This value is achieved in a similar way, using the

.count() stream function instead of the .sum() function

the .mapToDouble() stream function is

also removed, as the integer counts is all we need.

\*/

//Opens a java 8 stream.

//Filters the stream to all values between the start time

//& end time (done through lambda equations).

//Converts this stream to a double value with .mapToDouble()

//& adds all values i (the individual temperatures) together

//with .sum(). This value is then assigned to tempList

**double** tempList = measurements.stream()

.filter(m->m.getTime()>=startTime

&& m.getTime()<=endTime)

.mapToDouble(i->i.getTemp()).sum();

//Gets the number of results from the stream filter

//and divides this number by the tempList variable.

//The number of results is gotten in a similar way

//to above, but using .count() instead.

**return** (tempList / measurements.stream()

.filter(m->m.getTime()>=startTime

&& m.getTime()<=endTime)

.count());

}

**static** **double** avgTemperatureAcrossAllStations(**int** startTime, **int** endTime) {

//initialises the total temperature & the total count values

**double** totalTempList = 0;

**int** totalCount = 0;

//Runs a for loop for all the WeatherStations e in the stations ArrayList.

//Sums up all temperature values within the measurements list

//(This is the same code as the avgTemperature function).

//Then adds these to the totalTempList variable.

//The totalCount value is incremented by the amount of

//temperatures found in the stations ArrayList.

//Once this code is done, the totalTempList

//value is divided by the totalCount value, & returned.

**for**(WeatherStation e:*stations*) {

**double** tempList = e.measurements.stream()

.filter(m->m.getTime()>=startTime

&& m.getTime()<=endTime)

.mapToDouble(i->i.getTemp())

.sum();

totalTempList+=tempList;

totalCount += e.measurements.stream()

.filter(m->m.getTime()>=startTime

&& m.getTime()<=endTime)

.count();

}

**return** totalTempList / totalCount;

}

//main method

**public** **static** **void** main(String[] args){

//creating a new station, stationA & adding it to the ArrayList stations

WeatherStation stationA = **new** WeatherStation("Galway");

stationA.addWeatherStation(stationA);

//creating arbitrary measurements, and adding them to the measurements ArrayList

stationA.addMeasurement(1, 15.3);

stationA.addMeasurement(2, 12.5);

stationA.addMeasurement(3, 14.7);

//calling the avgTemperature function for the station instance & printing out the results

System.***out***.println("\nStation A average temperature: ");

System.***out***.println(stationA.avgTemperature(1, 3));

//this process is repeated 2 more times for the separate instances

WeatherStation stationB = **new** WeatherStation("Dublin");

stationB.addWeatherStation(stationB);

stationB.addMeasurement(1, 4.3);

stationB.addMeasurement(2, 4.3);

stationB.addMeasurement(3, 4.3);

System.***out***.println("\nStation B average temperature: ");

System.***out***.println(stationB.avgTemperature(1, 3));

//code block for station C

WeatherStation stationC = **new** WeatherStation("Cork");

stationC.addWeatherStation(stationC);

stationC.addMeasurement(1, 17.8);

stationC.addMeasurement(2, 15.6);

stationC.addMeasurement(1, 19.4);

System.***out***.println("\nStation C average temperature: ");

System.***out***.println(stationC.avgTemperature(1, 3));

//calling & printing out the avgTemperatureAcrossAllStations function

System.***out***.println("\nTotal average temperature: ");

System.***out***.println(WeatherStation.*avgTemperatureAcrossAllStations*(1, 3));

}

}

**Screenshot:**

