**SD1 - Coursework**

**Antero Duarte: 40211946**

**30/11/2015**

**Table of Contents**

[1 Introduction 3](#__RefHeading__645_1271754275)

[2 Final class 4](#__RefHeading__617_1271754275)

[3 Country class 10](#__RefHeading__623_1271754275)

[4 Week Class 13](#__RefHeading__625_1271754275)

[5 Console Output 15](#__RefHeading__643_1271754275)

# Introduction

This report is the submission of the code for Assignement 2 – Google Trends Data.

The code should consist of three java classes:

* Country
* Week
* Final

The class Final shows has the main method and outputs the answers.

# Final class

import java.io.BufferedReader;

import java.io.FileReader;

import java.time.LocalDate;

import java.time.Month;

import java.util.\*;

*/\*\**

*\* Created by Antero on 26/11/15.*

*\**

*\* The code will be publicly available on https://github.com/afduarte after the submission date, if this poses a problem,*

*\* please inform me not to do so.*

*\**

*\*/*

public class Final {

public static void main(String[] args) throws Exception {

BufferedReader fh =

new BufferedReader(new FileReader("iot.txt")); //Open file iot.txt

//First line contains the language names

String s = fh.readLine();

List<String> langs =

new ArrayList<>(Arrays.asList(s.split("\t")));

langs.remove(0); //Throw away the first word - "week"

//Create a Treemap of Week(See Week class) to a treemap of String(Language) to Integer(Interest)

TreeMap<Week,TreeMap<String,Integer>> interestByWeek = new TreeMap<>();

while ((s=fh.readLine())!=null)

{

String [] wrds = s.split("\t");

TreeMap<String,Integer> interest = new TreeMap<>();

for(int i =0;i<langs.size();i++)

interest.put(langs.get(i),Integer.parseInt(wrds[i+1]));

String [] period = wrds[0].split(" - ");

Week week = new Week(period[0],period[1]);

interestByWeek.put(week,interest);

}

fh.close();

TreeMap<String,Country> world = new TreeMap<>();

//Go through every year file, create a new country for every country there and put it in the TreeMap world

for (int i=2004;i<=2015;i++){

//Temporary holder for the country name and interest values

TreeMap<String,int[]> interestMap = new TreeMap<>();

BufferedReader getInterest =

new BufferedReader(new FileReader(i+".txt"));

String s2 = getInterest.readLine(); //Throw away the first line

while ((s2=getInterest.readLine())!=null)

{

String[] wrds = s2.split("\t");

int[] interest = new int[] {

Integer.parseInt(wrds[1]),

Integer.parseInt(wrds[2]),

Integer.parseInt(wrds[3]),

Integer.parseInt(wrds[4]),

Integer.parseInt(wrds[5])

};

interestMap.put(wrds[0],interest);

}

//Checks if the country exists in world.

//If it doesn't exist, creates it

for(String country:interestMap.keySet()) {

Country c = world.get(country);

if(c==null) {

c = new Country(country);

world.put(country,c);

}

//Sets the year for country c

c.setYear(i,interestMap.get(country));

}

getInterest.close(); //Closes the file

}

// Data structures available are: langs, interestByWeek, world

// langs is a list of languages

// iot is interest over time - a map from week to a map from languages

//interestByWeek is a map of Week(Class created to have start and end dates as LocalDate instead of string)

// to a map of Languages to Interest

// world maps every country in the world.

// an object of type country contains data for every country in the world over the years

//Medium ones:

System.out.println("Medium Ones:");

//Question 6: Which regions have demonstrated interests in exactly two programming language in 2010? (Give the region and the two languages.)

System.out.println("Q6 The following countries have demonstrated interest in exactly 2 languages in 2010: ");

// Stream the values in world

world.values().stream()

// filter for the countries that have the year 2010 and have shown interest in 2 languages

.filter(c -> c.hasYear(2010) && c.getInterestByLang(2010).size() == 2)

// for each country that is still in the stream, print the name and the languages it has shown interest in

.forEach(c ->System.out.printf("\t%s has shown interest in: %s and %s\n",c.getName(),c.getInterestByLang(2010).keySet().toArray()[0],c.getInterestByLang(2010).keySet().toArray()[1]));

//Question 8: Which are the least popular programming languages in the United Kingdom for each of the years 2009 to 2014?

System.out.println("Q8");

// Go through the years 2009 to 2014

for(int i =2009;i<=2014;i++)

// Check if there is data for the specified country in the specified year

if(world.get("United Kingdom").hasYear(i))

// Print the year and get the least popular language by using the method popInYear provided by the country class

System.out.printf("\tLeast popular for year %d is: %s\n",i,world.get("United Kingdom").popInYear(i,"min"));

//Hard ones:

System.out.println("\nHard Ones:");

//Question 11: Which are the top 5 regions that demonstrated significant growth of interests in programming languages in general?

//Significant growth will be 20% more than the interest in first year

System.out.println("\nQ11:");

TreeMap<String,Integer> grownInterest = new TreeMap<>(); //Holds growth against country name

world.values().stream()

//Filter for countries where growth was more than 20%

.filter(c->(float)c.getCombinedInterest(c.getYears().get(c.getYears().size()-1))\*1.20>c.getCombinedInterest(c.getYears().get(0)))

//Put stream results in grownInterest map

.forEach(c-> grownInterest.put(c.getName(),c.getCombinedInterest(c.getYears().get(c.getYears().size()-1))-c.getCombinedInterest(c.getYears().get(0))));

grownInterest.entrySet().stream()

//Sort grownInterest according to values in reverse order (get country with biggest growth first)

.sorted(Collections.reverseOrder(Comparator.comparing(Map.Entry::getValue)))

//Limit to 5 so we get only the 5 biggest growth countries

.limit(5)

//Print the limited stream

.forEach(x -> System.out.println(x.getKey()+" grew "+x.getValue()+" points."));

//Question 12: Which programming language set the record for losing the most interests over a 12 months period?

// When did this happen?

System.out.println("\nQ12:");

//Create a clone of the interestWeek Map that indexes by the starting LocalDate instead of a pair of LocalDates

TreeMap<LocalDate,TreeMap<String,Integer>> interestByStartDate = new TreeMap<>();

for(Map.Entry entry:interestByWeek.entrySet()) {

Week wk = (Week)entry.getKey();

TreeMap<String,Integer> val = (TreeMap<String,Integer>) entry.getValue();

interestByStartDate.put(wk.start(),val);

}

//Week holds two values of type LocalDate. Even though it wasn't created to work like this,

// it can hold values with a 12 month difference, being perfect to hold the LocalDates for this case.

//Initialize values that will be needed to print the answer and keep track of the biggest loss of interest so far

ArrayList<Week> lossPair = null;

String lossLang = "";

int loss = 0;

//Go through interestByStartDate

for(LocalDate wk:interestByStartDate.keySet()) {

// Check every language returned for that week

for (String lang : langs) {

// wk1 holds the interest for the language "lang" in the week "wk"

int wk1 = interestByStartDate.get(wk).get(lang);

// Get list of weeks that exist in the map and are 12 months after the week "wk" (within 1 week tolerance)

ArrayList<LocalDate> closestArray = getCloseWeeks(interestByStartDate, wk);

// If closestArray is not empty and for each Date in said array, get the interest for each language "lang"

// and act accordingly

if (!closestArray.isEmpty()) {

for(LocalDate closest:closestArray) {

int wk2 = interestByStartDate.get(closest).get(lang);

/\* There are 2 different scenarios:

\* 1. The language "lang" is the same as the current lossLang (language with highest loss)

\* and the loss (difference between wk1 and wk2) is the same value.

\* 2. The loss is higher than the current highest and the language is either the same or a

\* different one (doesn't matter because even if it is the same language but there are

\* different loss values, the Array would have to be cleared).

\*/

if(lang.equals(lossLang) && wk1-wk2==loss) {

// If 1 is the case, lossPair ArrayList is initialized if it is null, and the 12 month period

// is added to it as an object of type Week.

if (lossPair == null)

lossPair = new ArrayList<>();

lossPair.add(new Week(wk, closest));

// If 2 is the case, the ArrayList is cleared and the 12 month period is added to it as an

// object of type Week. The current values are updated: lossLang and loss

} else if(wk1-wk2 > loss){

// Even though clearing the existing ArrayList instead of creating a new one would be more

// efficient, not having to check if it was initialized before (null check) simplifies the code.

lossPair = new ArrayList<>();

lossPair.add(new Week(wk, closest));

lossLang = lang;

loss = wk1-wk2;

}

}

}

}

}

//Print the results. Used iterator for loop to easily detect the last time it loops and print a new line character.

System.out.printf("%s lost the most interest(%d points) in the following 12 month periods: ",lossLang,loss);

for (Iterator<Week> iterator = lossPair.iterator(); iterator.hasNext(); ) {

Week wk = iterator.next();

System.out.printf("|%s| ", wk);

if(!iterator.hasNext())

System.out.printf("\n");

}

//Question 13: Languages popular at University may be higher in September and October

// this means the Sept/Oct average is higher than the whole year average.

// Show this trend for the language Java. The example shown has figures for c++.

// You may assume the academic year starts on 1st September every year.

//Ac Yr All Yr Sep/Oct

//2009/10 8.69 9.75 c++

//2010/11 7.67 8.56 c++

//2011/12 7.02 8.00 c++

//2012/13 6.21 6.89 c++

//2013/14 5.94 6.67 c++

//2014/15 5.76 6.50 c++

System.out.println("\nQ13: ");

System.out.printf("Acad Year\tAvg Yr\tAvg Sep/Oct\n");

// Will iterate through the years 2004 to 2014, because 2015 has no data for academic months

for (int i=2004;i<2015;i++){

//Initialize variables

float totalJava =0; // Holds the total interest throughout the year

int totalCount =0; // Holds the number of weeks included

float acadJava = 0; // Holds the total interest for the academic months

int acadCount =0; // Holds the number of weeks included

String lang = "java";// Holds the language to calculate for

for (Week wk:interestByWeek.keySet()) {

// Check if the week is inside the current year

// (week start is after 1st of September of the current year and week end is before 1st of September of the next year)

if (wk.start().isAfter(LocalDate.of(i, Month.SEPTEMBER, 1)) && wk.start().isBefore(LocalDate.of(i + 1, Month.SEPTEMBER, 1))) {

// Accumulates interest for the language and increments the counter

totalJava += interestByWeek.get(wk).get(lang);

totalCount++;

}

// Check if the week is inside the current academic year

// (week start is after 31st of August of the current year and week end is before 1st of November of the current year)

if (wk.start().isAfter(LocalDate.of(i, Month.AUGUST, 31)) && wk.start().isBefore(LocalDate.of(i, Month.NOVEMBER, 1))) {

// Accumulates interest for the language and increments the counter

acadJava += interestByWeek.get(wk).get(lang);

acadCount++;

}

}

// Prints "Academic Year", "Language average for whole year", "Language average for academic months", "Language"

System.out.printf("%s\t%1.2f\t%1.2f\t%s\n",i+"/"+(i+1),totalJava/totalCount,acadJava/acadCount,lang);

}

}

// Method to find a match in the map for the closest week 12 months after the input date

// Returns an ArrayList of all the matches with a tolerance of one week before the date and one week after the date

public static ArrayList<LocalDate> getCloseWeeks(TreeMap<LocalDate,?> map, LocalDate originalDate){

//Takes in originalDate and adds 12 months

LocalDate twelveAfter = originalDate.plusMonths(12);

//Create an ArrayList to hold the LocalDate values that match the criteria

ArrayList<LocalDate> answer = new ArrayList<>();

// Go through the input map and check if it is within the tolerance timespan

for(LocalDate wk: map.keySet()){ // Although a stream API

if(twelveAfter.isAfter(wk.minusWeeks(1)) && twelveAfter.isBefore(wk.plusWeeks(1))) // call could have been

answer.add(wk); // used, there was a

} // performance loss in doing so.

// Returns the ArrayList

return answer;

}

}

# Country class

import java.util.ArrayList;

import java.util.Collections;

import java.util.Map;

import java.util.TreeMap;

*/\*\**

*\* Created by Antero on 12/11/15.*

*\* Class country stores all the data for a country in a TreeMap of Integer(year) to*

*\* a TreeMap of String(language) to Integer(interest in language)*

*\**

*\*/*

public class Country {

//Fields

private String country; // The name of the country

private TreeMap<Integer,TreeMap<String,Integer>> langInterestByYear; // A TreeMap that holds the TreeMap of language to interest against an Integer key (the year)

private String[] langs = new String[]{"java","c++","c#","python","JavaScript"}; // The list of languages

// Methods

public void setYear(int year,int[] interestInLang){

//Takes a year and an array of Integer(interest in each language) and puts them into a TreeMap

//That TreeMap is then put into the TreeMap langInterestByYear of this country

TreeMap<String,Integer> interestByLang = new TreeMap<>();

for(int i=0;i<langs.length;i++)

interestByLang.put(langs[i],interestInLang[i]);

this.langInterestByYear.put(year,interestByLang);

}

// Method to get either the most or the least popular language in a specified year.

public String popInYear(int year,String maxOrmin){

int valOfPop;

switch (maxOrmin.toLowerCase()){

case "max":

valOfPop=(Collections.max(this.getInterestByLang(year).values()));

String mostPop="";

for (Map.Entry<String, Integer> entry : this.getInterestByLang(year).entrySet()) {

if (entry.getValue()==valOfPop) {

mostPop = entry.getKey();

}

}

return mostPop;

case "min":

valOfPop=(Collections.min(this.getInterestByLang(year).values()));

String leastPop="";

for (Map.Entry<String, Integer> entry : this.getInterestByLang(year).entrySet()) {

if (entry.getValue()==valOfPop) {

leastPop = entry.getKey();

}

}

return leastPop;

default:

return "You must pick 'max' or 'min'";

}

}

// Method to return an Integer that represents the interest in a given language for a given year

public int getInterest(int year,String lang) {

return this.langInterestByYear.get(year).get(lang);

}

// Method to return a sum of all the interest values in all the languages for a given year.

public int getCombinedInterest(int year){

// Checks if there is data for this country, if there is no data, returns 0.

if(this.hasYear(year)) {

// If there is data, accumulates the interest for every language

int combined = 0;

for (String lang : langs)

combined += this.getInterest(year, lang);

return combined;

} else return 0;

}

// Checks if the country has the given year by checking against the keys in the TreeMap.

// Returns a boolean value

public boolean hasYear(int year){

return this.langInterestByYear.containsKey(year);

}

// Returns a TreeMap of the interest in the different languages if the interest value is higher than 0

// Organized by year

public TreeMap<Integer,Integer> getInterestByYear(String lang) {

TreeMap<Integer,Integer> interestInLangByYear = new TreeMap<>();

for(int year:this.getYears()) {

int interestInLang = this.langInterestByYear.get(year).get(lang);

if(interestInLang > 0)

interestInLangByYear.put(year,interestInLang);

}

return interestInLangByYear;

}

// Returns a TreeMap of the interest in the different languages if the interest value is higher than 0

// Organized by language

public TreeMap<String,Integer> getInterestByLang(int year) {

TreeMap<String,Integer> interestInLangByYear = new TreeMap<>();

for (String lang : langs) {

int interestInLang = langInterestByYear.get(year).get(lang);

if (interestInLang > 0)

interestInLangByYear.put(lang, interestInLang);

}

return interestInLangByYear;

}

// Returns the name of the country

public String getName() {

return this.country;

}

// Returns an ArrayList of every year for which this country has data for

public ArrayList<Integer> getYears(){

return new ArrayList<>(this.langInterestByYear.keySet());

}

@Override

// Simply provides a toString method.

public String toString(){

String years = this.langInterestByYear.keySet().toString();

return String.format("This country is %s, I have data from the following years: %s",this.country,years);

}

//Constructors

//Takes in a String(country name) and initializes a new empty TreeMap to hold the data for the country

Country(String name){

this.country = name;

langInterestByYear = new TreeMap<>();

}

}

# Week Class

import java.time.LocalDate;

*/\*\**

*\* Created by Antero on 14/11/15.*

*\**

*\* Class Week turns an input of two strings in the format YYYY-MM-DD into LocalDate Objects*

*\* Making it possible to use all the methods in the LocalDate class rather than just having a String.*

*\* e.g.: Week.start().getYear() ---> Returns an integer that represents the year in that week.*

*\* Week.start().isBefore(Week anotherWeek) ---> returns true if the first week is chronologically before the second week*

*\* Week implements Comparable, making it possible to compare two weeks thus, iterating through a structure that uses weeks as key*

*\*/*

public class Week implements Comparable<Week> {

//Week is an array of LocalDate that contains two values, a start date and an end date.

//Fields

private LocalDate[] week = new LocalDate[2];

//Methods

//start() returns the LocalDate of the start of the Week

public LocalDate start(){

return this.week[0];

}

//end() returns the LocalDate of the end of the Week

public LocalDate end(){

return this.week[1];

}

@Override

//toString() Override method returns both dates in a readable format: YYYY-MM-DD - YYYY-MM-DD

public String toString(){

return String.format("%s - %s",this.start(),this.end());

}

// Constructors

Week(LocalDate start, LocalDate end){

this.week[0] = start;

this.week[1] = end;

}

Week(String startString, String endString){

//The input strings are split on the "-" character and turned into a LocalDate.of(YYYY,MM,DD)

String start[] = startString.split("-");

String end[] = endString.split("-");

LocalDate startDate = LocalDate.of(Integer.parseInt(start[0]),Integer.parseInt(start[1]),Integer.parseInt(start[2]));

LocalDate endDate = LocalDate.of(Integer.parseInt(end[0]),Integer.parseInt(end[1]),Integer.parseInt(end[2]));

this.week[0] = startDate;

this.week[1] = endDate;

}

@Override

// Override method to make Week objects comparable. The use of the .start() method makes the comparison between

// two LocalDate values, which already has a well defined compareTo() method.

public int compareTo(Week otherWeek) {

return this.start().compareTo(otherWeek.start());

}

}

# **Console Output**

Medium Ones:

Q6 The following countries have demonstrated interest in exactly 2 languages in 2010:

Algeria has shown interest in: c++ and java

El Salvador has shown interest in: JavaScript and java

Ethiopia has shown interest in: c++ and java

Lebanon has shown interest in: JavaScript and java

Luxembourg has shown interest in: JavaScript and java

Uruguay has shown interest in: JavaScript and java

Q8

Least popular for year 2009 is: c++

Least popular for year 2010 is: c++

Least popular for year 2011 is: c++

Least popular for year 2012 is: c#

Least popular for year 2013 is: java

Least popular for year 2014 is: c++

Hard Ones:

Q11:

China grew 394 points.

Bangladesh grew 34 points.

Bolivia grew 26 points.

Ethiopia grew 20 points.

Nepal grew 20 points.

Q12:

java lost the most interest(22 points) in the following 12 month periods: |2004-10-24 - 2005-10-30|

Q13:

Acad Year Avg Yr Avg Sep/Oct

2004/2005 77.81 85.22 java

2005/2006 62.87 68.33 java

2006/2007 55.13 57.44 java

2007/2008 48.23 51.89 java

2008/2009 42.56 45.75 java

2009/2010 37.12 39.50 java

2010/2011 33.92 35.22 java

2011/2012 33.73 35.00 java

2012/2013 34.38 34.78 java

2013/2014 34.02 34.78 java

2014/2015 30.82 33.38 java