Please paste a screenshot of a successful program run, and copy-and-paste the source code from your .java file(s), here.

Your team member names must be a comment on the first line of each Java file.

// Java Program by Robert Stepp and Freddy Krueger 05/20/2018

/\*\*

\* CS143 Team Project

\* 3. (Based on Gaddis programming challenges 7.1) Rainfall Class

\* Create a text file that has 18 doubles with the average monthly rainfall for Olympia in the last year.

\* Write a RainFall class that stores the total rainfall for each of 18 months into an array of doubles. The program must accept the input out of chronological order and store it in sorted order.

\* Input dates as an integer, stored in the file as YYYYMM.

\* Sample input file:

\* 201802 1.3

\* 201701 2.5

\* 201803 4.0

\* 201711 3.5

\* 201802 1.2

\* . . . etc.

\* The program should have methods that return the following:

\* • Output the rainfall and date (YYYYMM) in order sorted by date

\* • The total rainfall on record

\* • The average monthly rainfall

\* • The month with the most rain

\* • The month with the least rain

\* Demonstrate the class in a complete program. Do not perform input validation.

\* @author Robert Stepp

\* @author Freddy Krueger

\*/

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.PriorityQueue;

import java.util.Stack;

import javax.swing.JOptionPane;

public class Main {

private static HashMap<Integer, Double> rainfall = new HashMap<Integer, Double>();

static HashMap<Integer, Character> rainfallGraphFill = new HashMap<Integer, Character>() {

private static final long serialVersionUID = 1L;

{

put(500, '#');

put(100, '\*');

put(50, '|');

put(25, '¦');

put(10, '!');

put(5, ':');

put(1, '.');

}

};

static HashMap<Integer, Integer> graphValues = new HashMap<Integer, Integer>() {

private static final long serialVersionUID = 1L;

{

put(500, 0);

put(100, 0);

put(50, 0);

put(25, 0);

put(10, 0);

put(5, 0);

put(1, 0);

}

};

// Used to sort the date into a descending order.

private static PriorityQueue<Object> pq = new PriorityQueue<Object>();

// Takes the output from Priority Queue in order.

private static ArrayList<Integer> dateArray = new ArrayList<Integer>();

static String[] menuOptions = { "All months", "Total rainfall", "Average monthly rainfall", "Month with the most rain",

"Month with the least rain", "All previous choices" };

static String[] months = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November",

"December" };

static Object[] keys = graphValues.keySet().toArray();

static String fileName = "";

static Object menuChosen = null; // User input on the what data to display.

static double totalMonthlyRainfall = 0.0; // Used for totalRainfall.

/\*

\* Max and min use these to search. I believe this would be O(N) as it uses the

\* import function as parent source and sets these during.

\*

\*/

static int monthMax = 0;

static int monthMin = 0;

static double monthMaxRainfall = 0.0;

static double monthMinRainfall = 100.0;

/\*\*

\* Takes the chosen output method and passes it to the associated method.

\*

\*/

public static void chosenOutput() {

if (menuChosen.equals(menuOptions[0])) { // Return output for every month.

buildTitle((String) menuChosen);

returnRainfallMonthly();

} else if (menuChosen.equals(menuOptions[1])) { // Return total rainfall for all months combined.

buildTitle((String) menuChosen);

System.out.printf("The total rainfall was %.2f inches over %d months.", totalMonthlyRainfall, dateArray.size());

} else if (menuChosen.equals(menuOptions[2])) { // Return average monthly rainfall.

buildTitle((String) menuChosen);

System.out.printf("The average rainfall was %.2f inches per month over %d months.\n", averageRainfall(), dateArray.size());

} else if (menuChosen.equals(menuOptions[3])) { // Return month with the most rain.

buildTitle((String) menuChosen);

mostRain();

} else if (menuChosen.equals(menuOptions[4])) { // Return month with the least rain.

buildTitle((String) menuChosen);

leastRain();

} else if (menuChosen.equals(menuOptions[5])) { // Return all above options.

buildTitle((String) menuChosen);

buildTitle((String) menuOptions[0]);

returnRainfallMonthly();

buildTitle((String) menuOptions[1]);

System.out.printf("The total rainfall was %.2f inches over %d months.\n", totalMonthlyRainfall, dateArray.size());

buildTitle((String) menuOptions[2]);

System.out.printf("The average rainfall was %.2f inches per month over %d months.\n", averageRainfall(), dateArray.size());

buildTitle((String) menuOptions[3]);

mostRain();

buildTitle((String) menuOptions[4]);

leastRain();

}

}

/\*\*

\* Parses the month digits YYYYMM to the text version of the month.

\*

\* @param date

\* from multiple sources

\* @return outputDate (String of Month YYYY)

\*/

public static String parseMonth(Integer date) {

String month = "";

String outputDate = "";

char[] dateNumArray = String.valueOf(date).toCharArray();

int monthNum = Integer.parseInt(dateNumArray[4] + "" + dateNumArray[5]);

month = months[monthNum - 1];

outputDate = String.format("%s %s", month, (dateNumArray[0] + "" + dateNumArray[1] + "" + dateNumArray[2] + "" + dateNumArray[3]));

return outputDate;

}

/\*\*

\* Builds the title from the chosen option and adds pizzazz around it.

\*

\* @param option

\* Is the chosen option and is converted to lower case and appended

\* to.

\*/

public static void buildTitle(String option) {

option = " Outputting " + option.toLowerCase() + " ";

int sizeNeeded = option.length() \* 2;

for (option.length(); option.length() < sizeNeeded;)

option = "◄" + option + "►";

System.out.println(option);

}

/\*\*

\* Returns the amount of rainfall for every month.

\*

\*/

public static void returnRainfallMonthly() {

System.out.printf("%-40s║%s\n", "Data", "Graph of data");

for (int j = 0; j < dateArray.size(); j++) {

parseRainfall(rainfall.get(dateArray.get(j)));

System.out.printf("%s had %.2f inches of rain.\t║%s\n", parseMonth(dateArray.get(j)), rainfall.get(dateArray.get(j)), graphRainfall());

}

printGraphLegend();

}

/\*\*

\* Graphs the output of the rainfall per month. \*

\*

\* @return graph

\*/

public static String graphRainfall() {

String builtGraph = "";

for (int s = 0; s < keys.length; s++)

for (int t = 0; t < graphValues.get(keys[s]); t++)

builtGraph += rainfallGraphFill.get(keys[s]);

return builtGraph;

}

/\*\*

\* Prints the legend of what the symbols on the graph mean

\*/

public static void printGraphLegend() {

int i = 0;

int j = 0;

int max = 0;

String test = String.format("║ %2s = %s ║\n",

rainfallGraphFill.get(keys[i]),

intToDouble((int) keys[i]) + " inches ");

if (test.length() > max)

max = test.length();

while (j < max) {

if (j == 0)

System.out.print("╔");

else if (j != max - 1)

System.out.print("═");

else if (j == max - 1)

System.out.print("╗");

j++;

}

String title = "Graph Legend";

for (title.length(); title.length() < max;) {

title = "║ " + title + " ║";

}

System.out.println("\n" + title);

j = 0;

while (j < max) {

if (j == 0)

System.out.print("╠");

else if (j != max - 1)

System.out.print("═");

else if (j == max - 1)

System.out.print("╣\n");

j++;

}

j = 0;

while (i < keys.length) {

System.out.printf("║ %2s = %s ║\n",

rainfallGraphFill.get(keys[i]),

intToDouble((int) keys[i]) + " inches ");

i++;

}

while (j < max) {

if (j == 0)

System.out.print("╚");

else if (j != max - 1)

System.out.print("═");

else if (j == max - 1)

System.out.print("╝");

j++;

}

System.out.println();

}

/\*\*

\* Changes keys int to double to display inches of rain.

\*

\* @param i

\* keys[i] to convert.

\* @return String with a double formatted.

\*/

public static String intToDouble(int i) {

double legend = 0;

legend = (double) i / 100;

String legendParsed = String.format("%.2f", legend);

return legendParsed;

}

/\*\*

\* Uses the total and dateArray to calculate the average rainfall.

\*

\* @return Average Monthly Rainfall

\*/

public static double averageRainfall() {

double averageMonthlyRainfall = totalMonthlyRainfall / dateArray.size();

return averageMonthlyRainfall;

}

/\*\*

\* Outputs the month with the least rainfall. Includes an if statement for no

\* rain.

\*

\*

\*/

public static void leastRain() {

if (monthMinRainfall == 0.0) {

System.out.printf("%s had no measureable rainfall.\n", parseMonth(monthMin));

} else

System.out.printf("%s had the least rainfall at %.2f inches.\n", parseMonth(monthMin), monthMinRainfall);

}

/\*\*

\* Outputs the month with the most rainfall.

\*

\*

\*/

public static void mostRain() {

System.out.printf("%s had the most rainfall at %.2f inches.\n", parseMonth(monthMax), monthMaxRainfall);

}

/\*\*

\* Gets input from user of the filename and how they want the output.

\*

\*/

public static void input() {

fileName = JOptionPane.showInputDialog(null, "Please enter filename:", "./src/Rainfall.csv");

menuChosen = JOptionPane.showInputDialog(null, "Choose output:", "Choose wisely", JOptionPane.QUESTION\_MESSAGE, null, menuOptions, null);

importFile(fileName);

}

/\*\*

\* Imports the file

\* https://www.mkyong.com/java/how-to-read-and-parse-csv-file-in-java/

\*

\* @param name

\* From input user supplied

\*/

public static void importFile(String name) {

BufferedReader br = null;

String line = "";

try {

br = new BufferedReader(new FileReader(name));

while ((line = br.readLine()) != null) {

String[] tempStorage = line.split(",");

String[] trimStorage = new String[tempStorage.length];

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

trimStorage[i] = tempStorage[i].trim();

}

rainfall.put(

Integer.parseInt(trimStorage[0]),

Double.parseDouble(trimStorage[1]));

findMax(Double.parseDouble(trimStorage[1]),

Integer.parseInt(trimStorage[0]));

findMin(Double.parseDouble(trimStorage[1]),

Integer.parseInt(trimStorage[0]));

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} finally {

if (br != null) {

try {

br.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

populatePQ();

}

/\*\*

\* Populates the priority queue to sort the dates.

\*

\*/

public static void populatePQ() {

Object[] tempArray = rainfall.keySet().toArray(); // Puts the keys from the hashmap into a temporary array.

for (int i = 0; i < rainfall.keySet().size(); i++) {

pq.add(tempArray[i]);

}

populateDA();

}

/\*\*

\* Populates the date arraylist with sorted dates from priority queue (Early to

\* present).

\*

\*/

public static void populateDA() {

int pqSize = pq.size();

for (int i = 0; i < pqSize; i++) {

dateArray.add((Integer) pq.remove());

}

orderedGraphKeys();

totalRainfall();

}

/\*\*

\* Finds the max rainfall and stores it and the date.

\*

\* @param rain

\* from importFile tempArray

\* @param date

\* from importFile tempArray

\*/

public static void findMax(double rain, Integer date) {

if (rain > monthMaxRainfall) {

monthMaxRainfall = rain;

monthMax = date;

}

}

/\*\*

\* Finds the min rainfall and stores it and the date.

\*

\* @param rain

\* from importFile tempArray

\* @param date

\* from importFile tempArray

\*/

public static void findMin(double rain, Integer date) {

if (rain < monthMinRainfall) {

monthMinRainfall = rain;

monthMin = date;

}

}

/\*\*

\* Calculates the total rainfall.

\*

\*

\*/

public static void totalRainfall() {

for (int t = 0; t < dateArray.size(); t++) {

totalMonthlyRainfall += rainfall.get(dateArray.get(t));

}

}

/\*\*

\* Parses rainfall and fills graphValues

\*

\* @param rain

\* Used to convert from double \* 100 to integer for graph creation.

\*

\*/

public static void parseRainfall(double rain) {

emptyGraphValues();

double rain100 = Math.round(rain \* 100);

int rainChanged = (int) rain100;

for (int rC = 0; rC < keys.length; rC++) { // rC = Rain Changed

int keyValue = (int) keys[rC];

while (rainChanged >= keyValue) {

if (rainChanged <= 4) {

graphValues.put(keyValue, rainChanged);

rainChanged -= rainChanged;

} else {

double d = rainChanged / keyValue;

graphValues.put(keyValue, (int) Math.floor(d));

rainChanged -= (graphValues.get(keyValue) \* keyValue);

}

}

}

}

/\*\*

\* Empty the HashMap graphValues to be used again.

\*/

public static void emptyGraphValues() {

Object[] keys = graphValues.keySet().toArray();

for (int gV = 0; gV < graphValues.size(); gV++) {

graphValues.put((int) keys[gV], 0);

}

}

/\*\*

\* Puts the keys from graphValues in order to display in descending order.

\*/

public static void orderedGraphKeys() {

PriorityQueue<Object> keysOrdered = new PriorityQueue<Object>();

Stack<Integer> reverseKeys = new Stack<Integer>();

for (int kO = 0; kO < keys.length; kO++) // kO = Keys Ordered

keysOrdered.add(keys[kO]);

for (int kR = 0; kR < keys.length; kR++) // kR = Key Reverse

reverseKeys.push((int) keysOrdered.remove());

for (int kT = 0; kT < keys.length; kT++) // kT = Key Transfer

keys[kT] = reverseKeys.pop();

}

public static void main(String[] args) {

input();

chosenOutput();

}

}

