Weather Station Analyst

A File I/O Lab for AP Computer Science A

Overview

In this lab, you will practice reading data from a CSV file, parsing it into ArrayLists, performing analysis on the data, and writing the results to an output file. These are foundational skills for working with real-world data in Java.

You are given a file called **weather_data.csv** containing 30 days of weather observations and a starter file called **WeatherAnalyzer.java** with the class structure and boilerplate already in place. Your job is to implement the logic for each method.

Key Concepts

Reading files: Scanner can read from a File object just like it reads from System.in. The expression *new Scanner(new File("data.csv"))* opens a file for reading. Then use *hasNextLine()* and *nextLine()* to process it line by line.

Parsing CSV: Each line of a CSV file is a comma-separated row. Call *line.split(",")* to get a String array of fields. Then convert numeric fields with *Integer.parseInt()* or *Double.parseDouble()*.

Writing files: PrintWriter lets you write text to a file using *println()* and *printf()* just like System.out. Always close the writer when done so data is flushed to disk.

Exception handling: File operations can throw exceptions (e.g., FileNotFoundException). The try-catch blocks are provided in the starter code.

The Data File

The file **weather_data.csv** has the following format. The first row is a header and should be skipped:

date	highTemp	lowTemp	precipitation	conditions
2025-03-01	52	34	0.0	Sunny
2025-03-02	48	30	0.0	Partly Cloudy
...	...	...	...	...

date — YYYY-MM-DD format, stored as a String

highTemp — daily high temperature in °F (int)

lowTemp — daily low temperature in °F (int)

precipitation — rainfall or snowfall in inches (double)

conditions — one of: Sunny, Partly Cloudy, Cloudy, Rainy, or Snowy

Your Tasks

Open **WeatherAnalyzer.java** and find the seven methods you need to complete. Each has a *// --- Your code here ---* comment marking where to write. The boilerplate (constructor, file opening/closing, try-catch, main method) is already written for you.

Task 1 — loadData

This method should read each remaining line from the Scanner (after the header has been skipped), split it by comma using *split(",")*, and add each field to the appropriate ArrayList. Remember:

- The split call gives you a *String[]*. Index 0 is the date, index 1 is the high temp, and so on.
- Use *Integer.parseInt()* to convert the temperature strings to ints before adding them to the Integer ArrayLists.
- Use *Double.parseDouble()* for the precipitation field.
- Use a *while (fileScanner.hasNextLine())* loop to process all rows.

Task 2 — findHottestDayIndex

Traverse the *highTemps* ArrayList and return the index of the maximum value. This is the standard “find max” pattern: start by assuming index 0 is the max, then compare each subsequent element.

Task 3 — findColdestDayIndex

Same approach as Task 2, but find the index of the minimum value in *lowTemps*.

Task 4 — averageHighTemp

Sum all values in *highTemps* and divide by the size of the list. Be careful with integer vs. double division! If you sum ints and divide by an int, Java truncates the result. Use a double accumulator or cast: *1.0 * sum / highTemps.size()*.

Task 5 — totalPrecipitation

Sum all values in the *precipitation* ArrayList and return the total.

Task 6 — countCondition

Count how many entries in *conditions* match the given parameter string. Use **.equals()** to compare Strings, not ==. This is a classic AP exam topic!

Task 7 — writeReport

Call the analysis methods you wrote above and use the PrintWriter to output a formatted report. Match the sample output below exactly. Use *String.format("%.1f", value)* to round doubles to one decimal place.

Expected Output

When you run the program, the console should print:

```
Report written to weather_report.txt
```

And the file **weather_report.txt** should contain:

```
===== WEATHER SUMMARY REPORT =====

Hottest day: 2025-03-26 (70 F)
Coldest day: 2025-03-08 (22 F)
Average high temperature: 56.8 F
Total precipitation: 4.5 inches
Sunny days: 11
Rainy days: 6
Snowy days: 2

===== END OF REPORT =====
```

Tips & Reminders

1. Make sure **weather_data.csv** is in the same directory as your .java file (or project root in an IDE).
2. Test incrementally! After finishing loadData, add a temporary print loop in main to verify the data loaded correctly before moving on.
3. Use **.get(i)** to access ArrayList elements by index and **.size()** instead of .length.
4. Use **.equals()** for String comparison. This is an AP exam favorite!
5. Close your files! The starter code handles this, but understand why: data may not be flushed to disk until the writer is closed.

Extension: Real NOAA Weather Data & Graphing

In this extension, you will download real weather observations from NOAA and generate a temperature line chart as a .png image file using the JFreeChart library.

Part 1: Download Real Data from NOAA

NOAA's Past Weather tool lets you look up historical daily observations for any weather station in the world. The underlying dataset is called GHCN-Daily (Global Historical Climatology Network).

1. Go to **https://www.ncei.noaa.gov/access/past-weather/**
2. Search for a location (try your town, a vacation spot, or a famous city).
3. Select a station from the results, then choose a date range (at least 30 days).
4. Download the CSV. The NOAA file will have columns like STATION, NAME, DATE, TMAX, TMIN, PRCP, and possibly others.
5. Adapt your loadData method (or write a new one) to handle the NOAA CSV format. Key differences:

   NOAA temperatures (TMAX, TMIN) are in tenths of a degree Celsius. To convert to Fahrenheit: *F = (value / 10.0) * 9/5 + 32*

   NOAA precipitation (PRCP) is in tenths of a millimeter. To convert to inches: *inches = value / 254.0*

   Some fields may be empty (missing data). You'll need to handle this with a check before parsing.

   The column positions differ from our lab CSV. Use the header row to identify which column index holds each field, or hard-code the new positions.

Part 2: Generate a Temperature Chart with JFreeChart

JFreeChart is a widely-used open-source Java library for creating charts. You'll use it to plot daily high and low temperatures as a line chart and save it as a PNG image.

Install: Downlad from https://repo1.maven.org/maven2/org/jfree/jfreechart/1.5.3/jfreechart-1.5.3.jar. Copy/drag into your IntelliJ project and right click -- Add as Library. This is the same way we added StdDraw.

Creating the chart: Here is the general approach. You do *not* need to follow this exactly — experiment!

```
import org.jfree.chart.*;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import java.io.File;

// 1. Create a dataset and add your data
DefaultCategoryDataset dataset = new DefaultCategoryDataset();
for (int i = 0; i < dates.size(); i++) {
    dataset.addValue(highTemps.get(i), "High", dates.get(i));
    dataset.addValue(lowTemps.get(i), "Low", dates.get(i));
}
```

```
// 2. Create the chart
JFreeChart chart = ChartFactory.createLineChart(
    "Daily Temperatures",  // title
    "Date",                // x-axis label
    "Temperature (F)",     // y-axis label
    dataset,
    PlotOrientation.VERTICAL,
    true, true, false);

// 3. Save to PNG
try {
  ChartUtils.saveChartAsPNG(
          new File("temperature_chart.png"),
          chart, 800, 400);
} catch (Exception e) {
  e.printStackTrace();
}
```

Your finished chart should show two lines (high and low temperatures) over the date range. Include it in your submission.

Challenge: Customize the chart! Change colors, add a third line for daily average temperature ((high + low) / 2), adjust the date labels so they don't overlap, or add a horizontal reference line at 32°F for the freezing point.