**COMMODITY PROFIT ANALYSIS PROJECT**

## Objective

You are given daily profit data of five commodities for a time period of 12 months. As your project you are expected to read the data files given to you and implement 10 query methods that analyze daily profit data of 5 commodities across 12 months. All methods must be robust: they must handle invalid inputs gracefully without throwing exceptions.

## Data

Commodity data is given to you in folder **Data_Files**. In this folder, data is organized as monthly files. Assume each month is 28 days, and day 1 is always Monday. Therefore, week 1 is days 1-7, week 2 is days 8-14, week 3 is days 15-21, and week 4 is days 22-28.

Format of each monthly data is given below:

12 text files: January.txt → December.txt

Format: Day,Commodity,Profit

Example line: 15,Gold,3200

28 days × 5 commodities (Gold, Oil, Silver, Wheat, Copper) per month

The commodities and months are already defined in **Main.java**, you should use those definitions.

You should implement the **loadData** method and read all data to memory using this method. Your **Main.java** should read the data from the given local **Data_Files** folder. You should not change the folder structure, location, or name of any file given to you.

## Rules

You should only use the algorithms and data structures you learn during the course.

- Only primitive arrays and multi-dimensional arrays allowed
- No ArrayList, Set, Map, HashSet, HashMap, etc.
- All 10 methods must be public static
- Methods must return values (no System.out.println)
- Invalid inputs must return special values (never throw exception or System.out.println error messages)

**Required Methods & Exact Return Format**

All the methods required are given to you in **Main.java**. You'll see the **loadData** method and the rest of the ten methods you need to implement. **Main java** also has the definitions of commodities and months. You are expected to implement the 11 methods listed in **Main.java**, but do not modify the methods signatures and existing definitions and do not delete anything from the **Main.java** file.

All methods except **loadData** is expected to return a result as a **String** or **int** value. Check **Main.java** for more details. Also if there is an error in the method, your methods are expected to return a special value designating the error. For each method these values are listed below. Be informed that all input and return parameters are case sensitive and exact match.

0. **loadData**()

You should implement the **loadData** method and read all data to memory using this method.

1. **mostProfitableCommodityInMonth**(int month)

→ month: 0=January, 11=December

→ Return: "Commodity totalProfit" (e.g., "Gold 108542")

→ Invalid month: return "INVALID_MONTH"

2. **totalProfitOnDay**(int month, int day)

→ day: 1–28

→ Return: integer total profit of all commodities on that day

→ Invalid month or day: return -99999

3. **commodityProfitInRange**(String commodity, int fromDay, int toDay)

→ commodity: exact match (case-sensitive)

→ fromDay ≤ toDay, both 1–28

→ Return: total profit in that day range (any month)

→ Invalid commodity, range, or from > to: return -99999

4. **bestDayOfMonth**(int month)

→ Return: day number (1–28) with highest total profit in that month

→ Invalid month: return -1

5. **bestMonthForCommodity**(String commodity)

→ Return: month name (e.g., "March") with highest total profit for that commodity

→ Invalid commodity: return "INVALID_COMMODITY"

6. **consecutiveLossDays**(String commodity)

→ Return: longest streak of negative profit days (across whole year)

→ Invalid commodity: return -1

7. **daysAboveThreshold**(String commodity, int threshold)

→ Return: number of days where profit > threshold

→ Invalid commodity: return -1

8. **biggestDailySwing**(int month)

→ Return: biggest absolute difference between two consecutive days' total profit

→ Invalid month: return -99999

9. **compareTwoCommodities**(String c1, String c2)

→ Return: "C1 is better by X" or "C2 is better by X" or "Equal"

→ Invalid commodity: return "INVALID_COMMODITY"

10. **bestWeekOfMonth**(int month)

→ Week 1: days 1–7, Week 2: 8–14, etc.

→ Return: "Week 3" (with highest total profit)

→ Invalid month: return "INVALID_MONTH"

**Grading**

Grading for the project will be done by an automated test script. A simplified example version of this test script is given to you as **AutoGrader.java**. You don't need to modify this script as part of your project. You are only responsible to implement **Main.java**. **AutoGrader.java** script is given to you in advance so that you can test whether the **Main.java** file you implemented is compatible with the final automated tests.

In order to test your code compile and run **AutoGrader** without any parameters. It will automatically compile your **Main.java,** call **loadData**, and run 10 tests (one per each method) on your **Main** file. At first, if you have not implemented any methods you should see all tests passed, as both the **Main.java** file and the **AutoGrader** tests are designed to emit dummy data. When you implement your methods, the correct calculated results of your methods will be different than the given dummy data and the tests will fail. This is expected and is not a problem. This also means that the **AutoGrader** is able to run all tests on your main file. If you are not seeing the test results and instead presented with another error, this means you have made a change in the **Main.java** that is not compatible with the **AutoGrader**. Therefore, in this case, it is your responsibility to fix this problem.

If you don't like seeing your test fail, or want to add your own tests, you can do this easily within **AutoGrader.java**. Open **AutoGrader.java** and see the 2D array named **TESTS**. Each line represents the method name to call, parameters to send, and the return parameter expected. You can change the dummy return parameters to correct ones to see your tests pass, or you can add new tests as new lines to this **TESTS** array. Notice that changing the **AutoGrader** is optional and you should be extremely careful when modifying this file.

The main grading will be done by a full version of the **AutoGrader**, which will execute 100 tests on your file. The number of tests you pass will affect your final grading in the project. All parameters, return parameters, month names, commodity names are case sensitive and exact match. In case of a mismatch the test will fail, so make sure to follow the guidelines when implementing your project.

Apart from automated tests, there will also be an oral exam for each student. The schedule for the oral exams will be posted later. As always any form of cheating is not allowed in SE 115 projects. All work should be your own and you should be able to explain your code in detail during these oral exams.

Finally, for grading, all work should be done on a personal Github account, and students are required to show their github logs during the oral exams. You are expected to start the project with an empty github import and piece by piece show your following work progress in the logs (ex., after implementing each method, you should commit to github). Developing the entire project outside of github and then dumping the final work to github will not be accepted and will be considered as a type of cheating. **Students that cannot show a log of at least 5 github commits will get a 0 grade for the entire project.**

**Submission – Deadline Dec 21th 23:59**

You should only submit **Main.java** file and nothing else.

Good luck!