Sahaj software | inspiring brilliance

General guidelines

At Sahaj, we strive to build high-quality software that has strong aesthetics (is readable and maintainable), has extensive safety nets to safeguard quality, handles errors gracefully and works as expected, without breaking down, with varying input.

We are looking for people who can write code that has flexibility built-in, by adhering to the principles of clean coding and Object-Oriented Development, and have the ability to deal with the real-life constraints/trade-offs while designing a system.

It is important to note that we are not looking for a GUI and we are not assessing you on the capabilities around code required to do the I/O. ***The focus is on the overall design****. So, while building a solution, it would be nicer if the input to the code is provided either via unit tests or a file. Using the command line (for input) can be tedious and difficult to test, so it is best avoided.*

Following is a list of things to keep in mind, before you submit your code, to ensure that your code focuses on attributes, we are looking for -

- Is the behaviour of an object distinguished from its state and is the state encapsulated?
- Have you applied SOLID principles to your code?
- Have you applied the principles of YAGNI and KISS (additional info here)?
- Have your unit tested your code or did TDD? If you have not, we strongly recommend you read about it and attempt it with your solution. Having tests is a must and we do appreciate if there is ample test coverage for the given solution, it is a definite plus if you write them.
- Have you looked at basic refactoring to improve the design of your code? Here are some guidelines for the same.
- Finally, and foremost, are the principles applied in a pragmatic way. Simplicity is the strongest of the traits of a piece of code. However, easily written code may not necessarily be simple code.

Sahaj software | inspiring brilliance

Problem Statement
(Average time to write a solution 8-10 hrs)

The transport officials from the city of Nepu want to design a payment system for public metro transport. They have come up with the idea of a prepaid card - TigerCard - which is an NFC enabled card that is to be tapped at entry and exit points of the metro stations.

To make the fares easier to understand for the commuters, the city has been divided into zones. Zone 1 is the central area and Zone 2 forms a concentric ring around Zone 1. Each metro station has been assigned to a zone. In the future, more zones will be added as the metro expands.

The problem statement is to design the fare calculation engine for TigerCard. Below are the definitions and fare rules.

Note: Use of any frameworks like REST APIs, Spring Boot, Django, databases, command line input is unnecessary and will complicate the solution. It will also extend the time taken to solve the problem.

Rules

Time of travel

The fare varies based on the time of the trip. There are two types of fares based on time of travel:

- Peak hours timings
  - Monday - Friday
    - 07:00 - 10:30, 17:00 - 20:00
  - Saturday - Sunday
    - 09:00 - 11:00, 18:00 - 22:00
- Off-peak hours timings
  - All hours except the above peak hours

The below table shows the fare that commuters will have to pay for a single journey from station A to station B. There is no concept of a round-trip journey.

Zones	Peak hours	Off-peak hours
1 - 1	30	25
1 - 2 or 2 - 1	35	30
2 - 2	25	20

Sahaj software | inspiring brilliance

Fare Capping

Fare capping works by rewarding commuters with free rides after they meet the fare equivalent of a daily, weekly, or monthly pass. With fare capping, social equity is achieved by removing upfront cost barriers associated with the recurrent passes. For example, a single ride costs 30 and the daily pass costs 90, the commuter earns a daily pass after the first 3 rides. For the rest of the day, all rides will be free for the commuter. Due to fare capping, the commuter does not have to invest 90 on the daily pass as they get the same features using a regular card.
The following capping categories are available:

- Daily
- Weekly

Capping Limits:

Zones	Daily Cap	Weekly Cap (Monday - Sunday)
1 - 1	100	500
1 - 2 or 2 - 1	120	600
2 - 2	80	400

The cap that is applicable for a day is based on the farthest journey in a day. For example, if a few journeys are within zone 1 and a single journey is between zone 1 & 2, then the daily cap applicable will be the one for zone 1 - 2. The first example later in the document illustrates the same. The weekly cap uses the same logic as the daily cap when determining the zones applicable for the week.

The weekly cap works in conjunction with the daily cap. So, when computing the weekly cap, each day fare might still be capped to that day's daily cap. The second example later in the document illustrates the same.

Problem statement

Given a list of journeys, the program should return the fare applicable.
Input Format: List of journeys for a single commuter, fields include date-time, from-zone, to-zone
Output: Computed Fare for the input journeys data

Below are some sample scenarios to consider and understand the problem further. The explanation column describes the applicable rule.

Sahaj software | inspiring brilliance

Examples

1. Daily cap reached

In this example, there are some trips between zone 1 and 2 in the day. So, the applicable daily cap is 120.

Input:

Day	Time	From Zone	To Zone	Calculated Fare	Explanation
Monday	10:20	2	1	35	Peak hours Single fare
Monday	10:45	1	1	25	Off-peak single fare
Monday	16:15	1	1	25	Off-peak single fare
Monday	18:15	1	1	30	Peak hours Single fare
Monday	19:00	1	2	5	The Daily cap reached 120 for zone 1 - 2. Charged 5 instead of 35

Output: 120

2. Weekly cap reached

Important Note: For easier explanation, the daily journeys have been rolled-up to just show the total fare applicable for a day as this makes it easier to understand how the weekly fare capping will be applicable. In the actual implementation, the input will be a list of individual journeys as shown in the previous scenario.

In this example, there are some trips between zone 1 and 2 in the week. So, the applicable weekly cap is 600. For Monday - Thursday, the daily cap was reached and so the fare is capped at 120 for them.

Input:

Day	Zones	Calculated Daily Fare	Explanation
Monday	1 - 2	120	Daily cap reached
Tuesday	1 - 2	120	Daily cap reached
Wednesday	1 - 2	120	Daily cap reached
Thursday	1 - 2	120	Daily cap reached
Friday	1 - 1	80	Daily cap not reached
Saturday	1 - 2	40	A weekly cap of 600 reached

			before the daily cap of 120
Sunday	1 - 2	0	A weekly cap of 600 reached
Monday (Next week)	1 - 2	100	New week started

Output: 700