Introduction

This test aims at taking a peek at the different strategies candidates might have when implementing the authorization of a credit card transaction.

Besides evaluating the correctness of your solution, we are interested in seeing how you model the domain, organize your code and implement your tests. **You can implement your solution in your favorite language** (although Scala and other functional languages are preferred). Libraries are, of course, allowed, and you can choose the database of your liking.

First, let's guide you through a few basic concepts.

Transaction

A simplified version of a credit card transaction payload is as follows:

```
{
    "account": "123",
    "totalAmount": 100.00,
    "mcc": "5811",
    "merchant": "PADARIA DO ZE               SAO PAULO BR"
}
```

Attributes

- **account** eve
- ry card holder has an account associated to that card
- **totalAmount** the amount to be charged
- **mcc** a numeric code, from `0000` to `9999` that indicates the category of the merchant (wether it's a bar, restaurant, clothes store, online service, etc). In this exercise we'll use a much smaller subset, consisting of the following MCCs:
    - Restaurants: `5811` to `5814`
    - Supermarkets: `5411`
    - Audiovisual media (books, movies and music): `5815`
- **merchant** the name and location of the merchant. *This field has exactly 40 characters. The first 25 refer to the merchant name and the last 15 refer to its location*

Benefits categories

Caju is a benefits platform and therefore our authorization process is slightly more complicated than that of a regular credit card. In order to comply with Brazilian regulations, we need to make sure a certain transaction can be mapped to a benefits category. In this exercise, we'll consider the following categories:

- **MEAL** restaurants, bars, etc
- **FOOD** supermarkets, grocery stores, etc
- **CULTURE** bookstores, streaming services (Spotify, Netflix, etc)

If a transaction cannot be mapped to the above categories, it can't be considered a benefit.

Challenges

Each of the following challenges are steps in the creation of a **full authorizer**. Your authorizer must be an HTTP server that processes the JSON transaction payload using the following rules.

The possible responses are:

- `{ "code": "00" }` if the transaction is **approved**
- `{ "code": "51" }` if the transaction is **rejected** because the account **does not have enough balance**
- `{ "code": "07" }` if any other problem occurs and the transaction cannot be successfully processed

The HTTP Status Code is always `200`

L1. Simple authorizer

The *simple authorizer* should work as follows:

- Receives the transaction
- Uses **solely** the MCC to map the transaction to a benefits category
- Approves or reject the transaction
- If the transaction is approved, the mapped category balance must be decreased by **totalAmount**.

L2. Authorizer with fallback

For *non-benefits expenses*, we have created another category, called **CASH**.

The *authorizer with fallback* should work like the *simple authorizer,* with the following difference:

- If the MCC can't be mapped to a benefits category or the given category balance is not sufficient to pay for the **whole transaction**, checks **CASH** balance and, if sufficient, debits this balance instead.

L3. Merchant-dependent

Sometimes, MCCs are incorrect and a transaction must be processed also taking into account the merchant data. Devise a mechanism for overriding MCCs based on the merchant name. Merchant name has a higher precedence over MCCs.

Examples:

- `UBER TRIP SAO PAULO BR`
- `UBER EATS SAO PAULO BR`
- `PAG*JoseDaSilva RIO DE JANEI BR`

- `PICPAY*BILHETEUNICO GOIANIA BR`

L4. Open question

The following is an open question regarding an important feature of a full authorizer (that you don't need to implement, only discuss in any way you deem fit, like text, diagrams, etc).

- **Concurrent transactions**: given that the same credit card can be used in different online services, there is a small but existent probability that two transactions occur at the same time. What would you do in order to guarantee only one transaction per account is being processed at a given time? Be aware of the fact that all transaction requests are synchronous and must be processed quickly (less than 100ms), or the transaction will timeout.

For this test, try your best to implement a transaction authorizing system considering all the challenges given (L1 to L4) and basic concepts. Remember to add local environment setup instructions.