README for Mabaya home exam
written by: Lital Shoshani

Design Approach
When I designed the solution for the home exam, I considered the next principles:
1. SOLID principles
   - Single Responsibility: Every class should have one and only responsibility.
   - Dependency inversion: The program should depend on interfaces.
   - Open/Close: The classes should be open to extension, but closed to changes and modification.
2. Modularity: the program should be divided into smaller components. The components should be independent.

Extended explanations

The model package
I created the model entities to be separated: each entity has its own class. The entities in the model package have one responsibility, and they do not interact with each other directly. That way, deleting, adding or editing the entities does not affect the others. In addition, creating new entities can be done with no effect on the other entities.

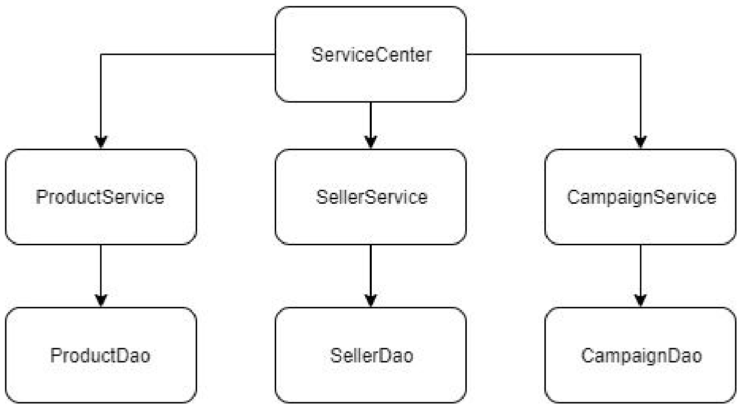
The DAO package
I used DAO pattern to define the data access entities, by using interfaces. The reason for that was to provide a way to move from one database to another. The interfaces provide the mechanism and the functionalities and not the actual implementation, which makes the changes between databases easier. Each database has its own way to implement the functionalities, and the only thing the databases have in common are the functionalities.

In this solution a database was needed to hold the objects of product, of the sellers and of the campaigns, and so I created a “temporary database” based on java hashmap and priority queue.
The reason I used these data structures was because of their complexity: I wanted to be able to put and get the data as fastest as possible.
- in hashmaps the put and get operations are O(1) (amortized).
- in priority queue inserting and deleting is done by O(logn), and peek() method to retrieve the element from the head is done by O(1). This way, for example, the highest bidder campaign can be found very fast.

I created three dao: one responsible to manage the products, one to manage the sellers and one to manage the campaigns. Each dao can make a query and return needed data of the entity.

The service package

For each dao I created a service: each service contains the dao interface (and not the actual class that implements the dao), and its purpose is to access the dao, add values and receive needed data. The services do not interact with each other, all except for one - The service center which is the “brain” of the program.



It can access each service, send data to be added to the daos, request some data and query. This is the class that connect all of the other components: it holds the other services, and uses them to add objects and features, or to pass and get information from one service to the other.

The api package

The controllers in the api package uses the serviceCenter as service (by using spring boot @Service), and they receive from the serviceCenter the data they need.

- A note about ServeAd api:
  this api receives a category and returns a product in the given category (if there is a campaign in that category, and if there is none - a product from the highest bidder campaign is returned).
  I used java Random in order to receive a random product from the chosen campaign. The reason behind that was to vary the shown promoted products of the specified category (in cases where the same campaign is returned in high frequency).

The configuration package

The package configures the starting of the program, by parsing the sellers and products files, and uploading the objects to the daos.
The configuration brain is the ConfigureModule class, that can be edited if the parsing files changes, or if there is no need to add the objects by files (in cases of databases such as sql).

Assumptions:

- I assumed that the input for the api in the program is received with Json.
- For this api there was no need to delete sellers or products. However, there are methods that deals with deleting. This way, for other programs where deleting is essential, using the daos, the services and the entities is easier.