OnlineStore

Project description

Task #1. Git

We will store source code of our `OnlineStore` in GitHub. Before start you should read and understand git principals and main git commands.

While doing each task, you should create a separate branch with the name of the topic, e.g. `01_git`, push your task code to this branch and create a pull request from your branch to master branch, and assign it to your trainer.

Do not merge it yourself!

I will look at your projects in the GitHub Classroom. Below is a link to complete the first task and get familiar with Git. At the same time, you will connect to the GitHub Classroom. We have a deadline in a week, but an earlier date is welcome :)

Task #2. Maven

Before start implementation our `OnlineStore`, we need to prepare project structure and set up dependency manager.

To handle our project dependencies and source code build we will use `Maven`.

Please create multi-module maven project in `Idea`, with such modules:

1. parent (this is general store module)
2. domain
3. store

4. consoleApp

Task #3. OOP

Before start creating source code, read carefully all materials about OOP. It is not only 3 principles for interview;)

Store functionality should be based on above principles.

Classes to create:

- `Product` with such attributes as name, rate, price
- `Category` classes with the name attribute, for each store category bike, phone, milk and products list
- `Store` - class that should handle category list
- `RandomStorePopulator` - utility class that will populate out store/category with fake data using `Faker` lib
- `StoreApp` - class with main method to execute our store scenario.

When invoke main method, application should init store with categories and products and `pretty` print this data.

Also, categories should be read dynamically (at runtime), from base category package using `reflections` lib.

Task #4. Collections

Starting extend our store. Please append ability user to interact with our store, while sending commands thru read stream.

Add support of such commands:

- `sort` - products from store according config. In resources folder create xml config file like

XML

```xml
<sort>
    <name>asc</name>
    <price>asc</price>
    <rate>desc</rate>
</sort>
```

Config file can contains from 1 to N fields. Sort should be done using `Comparator`. Sort and print should not modify

default store product lists and their order.

- `top` - print top 5 products of whole store sorted via price desc
- `quit` - exit app

Task #5. Patterns

Read all materials, try to find a `proper` place to your newly learned patterns in our app.

Task #6. Multithreading

Please implement `create order` functionality. Each order should be processed in separate thread. Whe user select product

, generate the random int from 1 to 30, and create thread that will process selected order for selected time, and after it

place the product in another collection (for example, purchased goods).

And create one more thread, that will be executed periodically, e.g. ones in 2 mins, that will clean up purchased collection.

You can implement this in "native" java methods but better and simplier to use java.util.concurrent(https://habr.com/ru/company/luxoft/blog/157273/) package.

Task #7. DB

Please replace `Reflection` and in memory products storage with database.

You should store categories and products for each category in databe tables.

Requirements:

1. use `JDBC`
2. you can select any DB, but for simplicity you can select H2 database in files mode
3. feel free to google)

Task #8. HTTP

The same task as previous, but to store and get store data you should implement HTTP server (use included in java or external lib),

that will handle in memory or DB your categories, but you will receive them via HTTP protocol.

Also, you should implement `add product to cart` logic in this server, and process this request via HTTP.

And finally, your HTTP service should be secured with basic authentication (credentials can be hardcoded).

For Http client side you also can use default (included in java) or use RestAssured.