Exercise 3: Patterns of Enterprise Application Architecture and Refactoring

Presentation of results in next lab /
Submission in AULIS until 10.11.22, 9:45am

Assignment 1: Repository for the Order System

Until now, the current order together with its items is held by the `OrderService` instance. Since this class implements the order system's use cases, it should not have to deal with managing (or even persisting) order and item instances (remember the *Single Responsibility Principle*?).

Your task: Introduce a repository that stores the orders and their associated items. The repository is accessed by the `OrderService` whenever orders are created, manipulated, or retrieved.

Steps:

- Create an *order repository* for storing orders and their associated items. The repository should provide *functionality* for creating and updating orders, for retrieving all orders, and for deleting all orders. You don't have to implement a persistence mechanism (i.e. no database or file access required).
    - Decide whether you design a *collection-oriented* or a *persistence-oriented* repository. Your decision will impact the repository's interface design!
    - Implement an in-memory-repository, but prepare for easily exchanging this simple repository implementation with a solution which is backed, e.g., by a database.
    - Ask yourselves whether it is advantageous to return the original (internal) data structures or a copy thereof.
- The `OrderService` shall access the repository whenever an order is created or changed.
    - When a new order is created, it should be stored in the repository.
    - When a new item is added to an order, the order should be updated in the repository.
    - When the order is finished, the `OrderService` will mark the order instance accordingly and update the changed order in the repository.
- Test your solution!

Assignment 2: User Interface for the Order System

So far, the user interface (probably) is still part of the `OrderService`, which again violates the Single Responsibility Principle. Therefore, the user interface should be extracted from the `OrderService` and moved into a separate class. And wouldn't it be nice, if we could choose between a command line interface and a graphical user interface?

Subtask 2.1: Separating the Command Line Interface from the Order Service

This first subtask is probably rather simple: extract the UI-related code from the `OrderService` (and potentially the `Order` class) and move it to a new class `CLI` (for command line interface). Among others, the `OrderService`'s `finishOrder()` method shall return the submitted order for further handling by the user interface.

Hints:

- Code related to the menu is clearly a matter of user interface.
- Also, the code which gathers input for new products or services should become part of the user interface: we don't want `System.out.println()` or `Input.readXYZ()` in our `OrderService` any longer!
  (The `CLI` will *probably* invoke the `orderXYZ()` method(s) of the `OrderService` and hand over the required parameters.)
- What about formatting prices and sorting items? Is this a task of the `OrderService` (or the `Order` class) or rather a matter of the user interface?

Implement your refactorings and test your solution!

Subtask 2.2: Introduction of a Graphical User Interface

Some of our customers are becoming unsatisfied with the command line interface ☹ Your company therefore decides that the order system needs an alternative graphical user interface. Work never stops…

Your task:

- Implement a *minimal* graphical user interface (e.g. using Swing) that
  - shows the *ordered* list of selected items, i.e. the current order ("shopping cart")
  - provides text fields and buttons for adding new products and services (remember to update the list of items),
  - and a button for finishing the order (only then a summary of all ordered items together with checkout date and the lump sum price will be shown). The ordered list of items ("shopping cart") above will be cleared after dismissing the order summary.
- Permit to choose whether the command line client or the graphical user interface will be used at start time by an argument of the `main()` method. The order system should be started either with the command `java OrderSystemMain cli` or `java OrderSystemMain gui`.
- Test your solution!