Exercise 2: Design Patterns

23.10.2025

Presentation of results in next lab /
Submission in AULIS until 30.10.25, 9:45am

Assignment 1: Abstract Factory in the Order System

This assignment is about extending the order system from exercise sheet 1 in such a way that the `OrderService` class is no longer dependent on concrete item classes (`Product`, `Service`). Basis for this assignment is your (improved) result of exercise sheet 1 (assignment 1).

Your task:

If you solved assignment 1 of exercise sheet 1 as expected, the only dependencies on these classes are for creating new product and service instances. In order to get rid of these, you are asked to make use of the *Abstract Factory* pattern in this assignment: create a factory that "produces" both types of items your order system offers!

Steps:

- Create an interface `ItemFactory` with methods for creating product and service instances and implement a `SimpleItemFactory` that implements this new interface.
- Assign a factory instance to the `OrderService` by invoking a dedicated setter method in `OrderSystemMain` with an instance of a `SimpleItemFactory`.
- Test your solution.

The only references to the `Product` and `Service` classes in the order system should now be in the `SimpleItemFactory`.

Background: Having only one "family" of products and services does not really give reason for the abstract factory pattern. We'll leave it as it is, but you may think of alternative families of products and services that may be ordered (e.g. mobility products like cars with services such as vehicle registration, 24 months inspection & repair package, and insurance, as well as electronic devices like smart phones with services such as trade-in, payment by instalments, and insurance).

Assignment 2: Universal Remote Control

In this assignment you are asked to implement a universal remote control. A universal remote control has buttons which can be configured flexibly to control different functions of a device. The type of device is unknown (hence the remote control is "universal" and needs to be configurable). The remote control has

- 3 *action buttons* which trigger pairs of activate / deactivate functionality: press a button once, and the configured activate functionality is executed. At the next button press, the configured deactivate functionality will be executed (and so on).
- 1 *undo button* which will undo the previously executed action: if some functionality has been activated before, the configured deactivate action will be executed (and vice versa).

The IntelliJ project "UniversalRemoteControl" provides the skeleton of a universal remote control:

- `IRemoteControl`: Interface for the remote control. Needs to be extended by a method for configuring the action buttons with pairs of activate / deactivate functionality.
- `RemoteControl`: Incomplete implementation of the interface (cf. TODOs in source code).

For the two subtasks below, you will have to extend this skeleton by representations of action buttons, undo button / history, and code for the configuration and execution of button behaviour.

Subtask 2.1: Actions for a Universal Remote Control

The universal remote control ought to be implemented and configured for use with a media player (e.g. a CD player or a streaming device). Your task:

- Have a close look at the GoF design patterns catalog and choose a suitable design pattern.
- Complete the given code skeleton such that the design pattern is used.
- Create and configure a remote control for a media player with paired actions "on" / "off", "play" / "pause", and "next" / "previous" in the `Main` class. Upon execution the actions simply print their activity to the console, e.g. "on" or "pause".
- Test the functionality by simulating action button presses in `Main`.

Subtask 2.2: Undoing Actions

Since the functionality assigned to the undo button is determined by the previously executed activate or deactivate action, it can only be configured dynamically, i.e. whenever an action button is pressed. If for example the play/pause button is pressed and the play action (activate action) is executed, the undo button needs to be configured with the pause action (the deactivate action).

Your tasks:

- Conceive a data structure for the undo button and configure it with the appropriate actions.
- Test your solution by extending the series of button presses in `Main` with undo button presses.
- Does your solution work for multiple subsequent undo button presses?

Bonus Task

Assignment 3: Media Presentations

One of the basic GoF ("Gang of Four") patterns is the *Composite* pattern. It is, e.g., used in UI frameworks for defining the hierarchical structure of UIs. If you don't know this pattern, you'll have to do some research!

In this assignment you are asked to apply the Composite pattern for defining and playing media presentations:

- Media presentations ("multimedia slide shows") combine different media objects into one presentation. Our media presentation system supports the media type *text*, *picture*, *video*, and *music*. Each of these media types can be played and prints messages such as "displaying text: <*text*>" or "showing picture no. <*number*>", where <*text*> and <*number*> are placeholders that need to be assigned values upon creation of media objects.
- The presentation system allows to form "complex" media objects by combining media objects through the application of two operators:
  - *Sequence*: media objects that are combined using the sequence operator will be played one after the other. Operators can be "played" as well. When played, the sequence operator prints a message like "playing in sequence" and then plays the combined media objects.
  - *Parallel*: media objects that are combined using this operator will be played in parallel (or concurrently). In order to indicate this concurrent playback, the parallel operator prints a message like "playing in parallel" (followed by the messages printed by playing the combined media objects).
- The operators can be applied recursively, i.e. a sequence operator may combine simple media objects (text, picture, …) and complex media objects (e.g. a parallel combination of music and a video).
- When playing a media presentation, i.e. combined media, the media objects and operators print their messages in the order defined by the operators. Nice to have for better readability: indentation of lines that are printed within the context of a sequence or parallel operator.

Your task:

- Implement the classes for defining complex media presentations using the Composite pattern.
- Create a media presentation with the following structure:
  - Text "A vacation to remember…" as introduction
  - Followed by a series of pictures and videos ("pic1" – "pic5", "vid1", "pic6" – "pic9", "vid2", "vid3"); this series is accompanied by music "Some lovely music" played in the background
  - Afterwards outro text "We'll go there again!"
- Play the media presentation.