Mandatory requirements

1. The implemented patterns must comply with the GoF definition discussed during the courses and laboratories. Incomplete variations or implementations are not accepted.
2. The pattern must be implemented completely correctly to be taken into account
3. The solution does not contain compilation errors
4. Patterns can be treated separately or can be implemented on the same set of classes
5. Implementations that are not functionally related to the requirements of the subject will NOT be taken into account (taking any example from other sources will not be evaluated)
6. It is NOT allowed to modify the received classes
7. Solutions will be cross-checked using MOSS. Code sharing between students is not allowed. Solutions with a degree of similarity of more than 30% will be canceled.

Mandatory Clean Code requirements (you can lose 2 points for each requirement that is not met) - a maximum of 8 points can be lost

1. For naming classes, functions, attributes and variables, the Java Mix CamelCase naming convention shall be used;
2. Patterns and the class containing the main () method are defined in distinct packages that have the form `cts.name.surname.gNrGroup.pattern.X`, `cts.name.surname.gNrGroup.main` (students in the additional year use "AS"Instead of gNrGroup)
3. Classes and methods are implemented respecting the principles of KISS, DRY and SOLID (attention to DIP)
4. The class names, methods, variables, and messages displayed on the console must be related to the received subject (generic names are not accepted). Functionally, the methods will display messages to the console that simulate the required action or will implement simple processing.

A software application is being developed for a startup that offers an online booking services.

2p. The solution allows customers to book tickets for different events, like concerts, museums, etc, but the current solution is inefficient because new event types are added constantly and their details may change in time. ACME Inc wants a better solution for the creation of different online booking types. Considering that all share the ***OnlineBooking*** abstract definition, implement an efficient solution that will allow them to create bookings for either Concerts, Events or Museums. (Optionally: each ticket should have a uniquely generated number)

2p. To test the solution, by using the received class and by creating at least 3 bookings of different types (Concerts or Events or anything else).

5p. The business is expanding and now is selling holydays trips to the seaside. After a hotel is selected, a client can choose from a wide range of options in order to finish the booking, as: included breakfast, number of persons in the room, type of beds, seaside view, floor level, smoking room, etc. Implement a flexible and efficient way of creating bookings that are able to store all or some of these options (any combination will be allowed). Once a booking is created, is not permitted to change any of its values.

2p. Test the solution by creating at least 3 bookings, with different combinations of options.

7p. The site displays for each trip, image galleries with the surrounding region, described by the ***GalleryImage*** class, to give more details to each offer. Following the performance tests, it was observed that in order to display a list of hotels from the same region, which share the same galleries of images, the memory occupied by the client app increases beyond acceptable limits. Implement a solution (based on a design pattern) that reduces the memory occupied by the application but still allows the display of images. Each image displayed is associated with a specific hotel and the time of the year (summer, winter, spring, autumn) in the hotels list from the same region.

2p. Test the proposed solution by displaying the 3 common images from 2 different hotels galleries. The solution must show that the memory has been used optimally.