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 considered
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 considered (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 4 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.model`, `cts.name.surname.gNrGroup.pattern.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 Bicycle shop.

4p. For a store that manufactures and sells bicycles, it is desired to implement an application to help create stickers for gluing on bicycle components. The creation of a Sticker object takes a long time because the dimensions of the bicycle component must be taken into account, as well as the color, to be different from that of the component for which it is created. Since the store sells several bicycles of the same model, it is desired to implement a module that allows the creation of Sticker type objects similar to one already created, without calling the constructor. The constructor of the Sticker class receives an object of the type of a class that implements the IComponent interface. Based on the object received as a parameter of Icomponent type, the Sticker type object is built with the required dimensions.

1p. Test the solution by creating at least four Sticker objects via the implemented module.

4p. The solution must allow the creation of bicycle components of different types: Fork, Top Tube, Wheel, etc. For each type of component, a class related to the respective component is used that implements the IComponent interface. To implement the module that will help the engineers of the bike manufacturing shop in the process of creating objects from the bike component family.

1p. Test the solution by creating at least four objects of at least two different types from the bicycle component family.