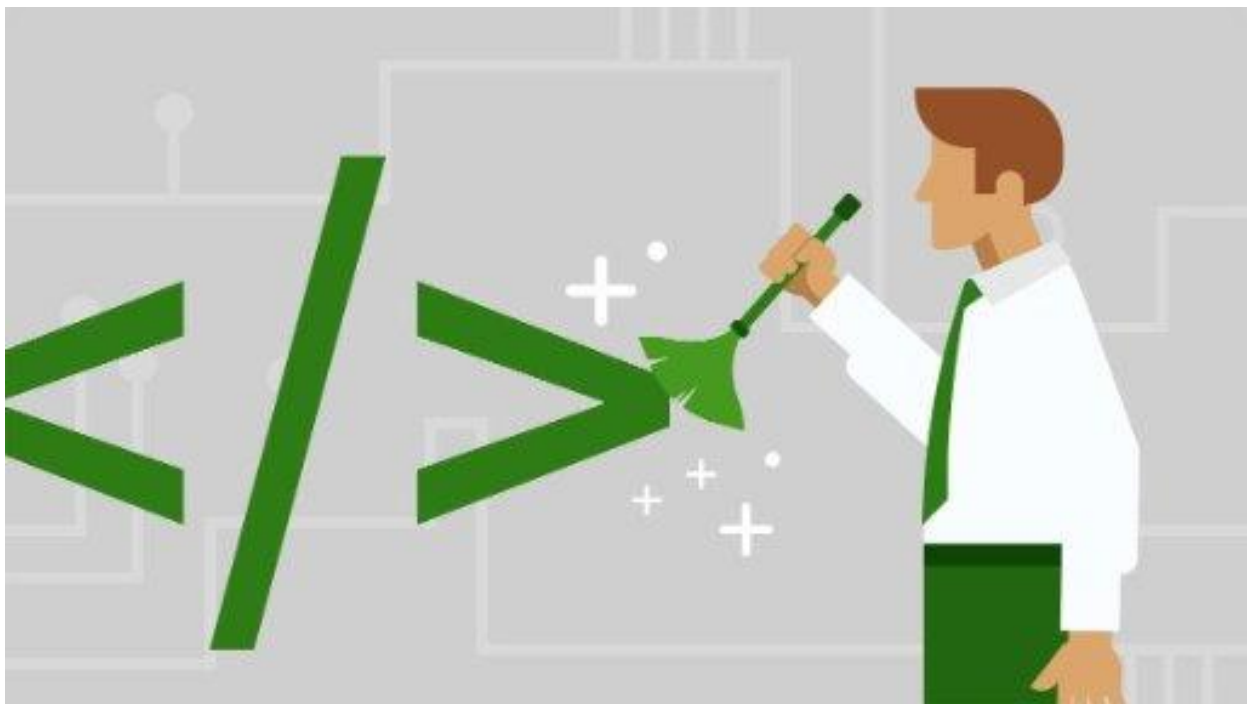


Course: DPR

Date: 10 January 2020

Design Patterns




Class: E-S41

Lecturer: Mariëlle Fransen

Student: Nadya Cheperkova

Nidhi Sharma



CONTENT	1
INTRODUCTION.....	3
BEHAVIORAL PATTERNS.....	3
COMMAND PATTERN	3
APPLICATION.....	4
MAINTAINABILITY	5
REUSABILITY	5
EXTENSIBILITY	5
CONCLUSION	6
UML DIAGRAM	7



INTRODUCTION

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. However, it is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

BEHAVIORAL PATTERNS

Behavioral patterns are responsible for the interaction between the objects. It is made in such a way that they can communicate with each other easily and still are loosely coupled. Loosely coupled system is where the objects do not depend on their concrete classes. Behavioral patterns do not only describe the patterns of the objects or classes but also patterns of communication between them. The patterns are divided into two categories: behavioral class patterns and behavioral object patterns. Behavioral class patterns use inheritance to distribute behavior between classes while behavioral object patterns use object composition rather than inheritance. There are ten different behavioral patterns – **Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method and Visitor**

COMMAND PATTERN

Command pattern is a behavioral pattern where the object encapsulates all information needed to perform an action or trigger an event at a later time. Hence this procedure lets you pass the object as a parameter in methods with different requests, delay or wait on a request's execution as well as support undoable operations. It is also known as Action, Transaction. The key to this pattern is an abstract Command class, which declares an interface for executing operations – different concrete commands. This interface includes an operation to execute the command.

APPLICATION

The purpose of this application is to show how Command pattern can be implemented. Hence, we tried to keep the application as small and simple as possible. The application represents a mini clothing shop with three different products – T-shirt, Pants and Shoes. We created an interface `IProduct` because all of the classes that implement it (T-shirt, Pants and Shoes) have the same methods (Order and Buy) but different return values. Order method returns the name of the product and Buy returns the price of the product so for simplicity we kept them both as string. Moreover, in order to keep the layer for the graphical user interface separate from the layer for the business logic, we introduced `ICommand` interface which is implemented from two different commands `OrderCommand` and `BuyCommand` . Command objects serve as links between the GUI and business logic objects. Therefore, the GUI object does not need to know what business logic object will receive as a request and how it will be processed. The GUI object only triggers the command, which handles all the details. Additionally, we have `PurchaseController` class which is responsible for initiating the requests. Hence, when the button 'Order' is clicked the GUI will receive the name of the product and when the button 'Buy' is clicked the GUI will receive the price of the same product.

The participants in the application are as follows:

- **Abstract Command** (*ICommand*) - declares an interface for executing an operation.
- **Concrete Command** (*OrderCommand, BuyCommand*) - defines a binding between a Receiver object and an action and implements the Execute method by invoking the corresponding operation on the Receiver.
- **Receiver** (*IProduct - Pants, Tshirts, Shoes*) – any class may serve as a receiver as long as it performs some business logic.
- **Invoker** (*PurchaseController*) – triggers the command instead of sending the request directly to the receiver.
- **Client** (*Form1*) - creates a Concrete Command object and sets its receiver.



MAINTAINABILITY

The application can be easily maintained because it is created in such a way that it can be used in the real world. Since every command is an independent class that is inherited from the interface, debugging and maintaining each command is easier. You can maintain each one of them on its own because each of them is responsible for only one operation (single responsibility principle). For instance, you can easily change or remove OrderCommand without influencing BuyCommand or any other class. Furthermore, because of the separation of the graphical user interface from the business logic and the single responsibility principle, the code can be easily tested (for example with unit tests) as we demonstrated in our application.

REUSABILITY

A solution with command design pattern can be considered as a reusable one because we can reuse the code for each command that we are adding. This is because we have ICommand interface, hence, when we decide to add a new command such as ReturnCommand, it will easily implement the interface and reuse all the methods in it.

EXTENSIBILITY

The command class or command interface allows a command to be executed by different receivers. For instance, the commands from our application BuyCommand and OrderCommand are both executed by the T-shirt, Pants and Shoes. This means the application can easily be extended by adding extra receivers such as Leggings or Scarf and still be able to execute the same commands without changing the other classes. Moreover, commands can also be easily added without changing the other commands or the receivers. Hence, we can add ReturnCommand which will implement ICommand interface.



CONCLUSION

To conclude, as all design patterns **Command** has some advantages and disadvantages. Even though it can contain a huge number of classes and objects working together to achieve the goal and the developers need to be careful in developing these classes, it is still a highly used pattern. It decouples the classes that invoke the operation from the object that knows how to execute the operation and allows you to create a sequence of commands by providing a queue system. Furthermore, parameterize of objects by an action to perform is a big advantage. Command is not one of the top five most used design patterns because of its complexity but it has huge applicability in today's world enterprises

UML DIAGRAM

