Command Pattern

Michal Moravik, SD20w2

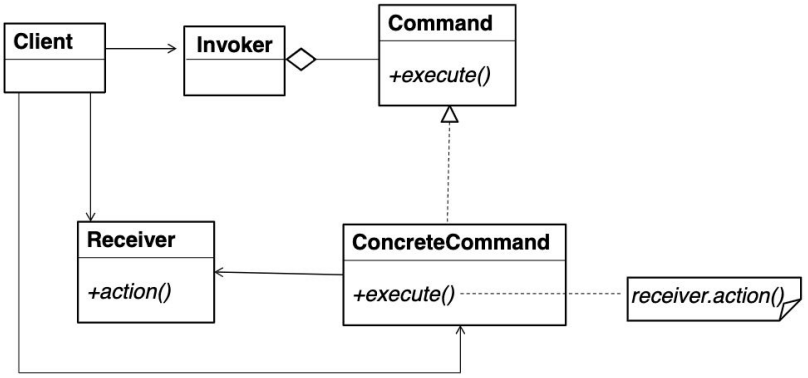
Name: Command Pattern

Category: Behavioral

Intent: Allows us to store requests as command objects (e.g. copy command/functionality is an object) to delay or queue the execution of an action or undo it at a later time.

Motivation:
Used when we want to reduce coupling between the GUI and business logic layers. By using Command Pattern, we can turn a specific method call into a stand-alone object. We use this pattern when we want to implement undo/redo operations. We can also queue operations or schedule their execution.

UML:



Implementation:
Below is an example of undo/redo functionality.

```
public interface ICommand {
    void execute();
    void undo();
}
```

```java
public class ChangeTextCommand implements ICommand {
    public Terminal terminal;

    ChangeTextCommand(Terminal terminal) {
        this.terminal = terminal;
    }

    @Override
    public void execute() {
        terminal.addToStack(" *NEW* ");
    }

    @Override
    public void undo() {
        terminal.removeFromStack();
    }
}
```

```java
public class Terminal {
    private final Deque<String> stack = new LinkedList<>();

    public void addToStack(String text) {
        stack.add(text);
        System.out.println("The current stack: " + stack.toString());
    }

    public void removeFromStack() {
        String last = stack.pollLast();
        System.out.println(last + " was removed. ");
        System.out.println("The current stack: " + stack.toString());
    }
}
```

```java
public class Client {

    public static void main(String[] args) {
        Terminal terminal = new Terminal();
        ICommand changeTextCommand = new ChangeTextCommand(terminal);

        changeTextCommand.execute();
        changeTextCommand.execute();
        changeTextCommand.undo();
    }
}
```

```
The current stack: [ *NEW* ]
The current stack: [ *NEW* ,  *NEW* ]
 *NEW*  was removed.
The current stack: [ *NEW* ]
```

Consequences:

Pros:

- Decoupling classes that call operations from the classes that perform operations
- Creation of undo/redo functionality
- Introducing new commands into the app without breaking existing client code
- Can create more complex commands by assembling multiple simpler ones

Cons:

- Introducing a new layer between senders and receivers, and so, it introduces a new complexity to the system
- More objects in the memory - need to create a command for each request

Known uses:

- undo/redo
- When we want to have commands (actions) separated from “the callers”. E.g. a copy command could be triggered either by a button in GUI or by a shortcut (cmd+C) (both are different “callers”)

Related patterns:

- Chain of Responsibility - commands can be implemented as handlers. You can execute a lot of different operations over the same context object, represented by a request.
- Prototype - storing copy of commands in history