Object Oriented Analysis and Design with Java

UE19CS353

Dr. L. Kamatchi Priya
Department of Computer Science and Engineering

PES UNIVERSITY ONLINE

UE19CS353: Object Oriented Analysis and Design with Java

OO Design Patterns & Anti-Patterns with Sample implementation in Java

Dr. L. Kamatchi Priya

Department of Computer Science and Engineering

UE19CS353: Object Oriented Analysis and Design with Java

Unit-5

Behavioral Design Pattern – Command

Agenda

- Introduction to Behavioral design patterns
- Command Design Pattern
  - ✔ Intent
  - ✔ Applicability
  - ✔ Structure
  - ✔ Collaborations
  - ✔ Consequence
  - ✔ Implementation
  - ✔ Related Patterns

Introduction to Behavioral Design Pattern

- **Behavioral design** patterns are concerned with **the interaction** and **responsibility of objects.**
- In these design patterns, the interaction between the objects should be in such a way that they can **easily talk to each other** and still should be **loosely coupled**.

Command Pattern:

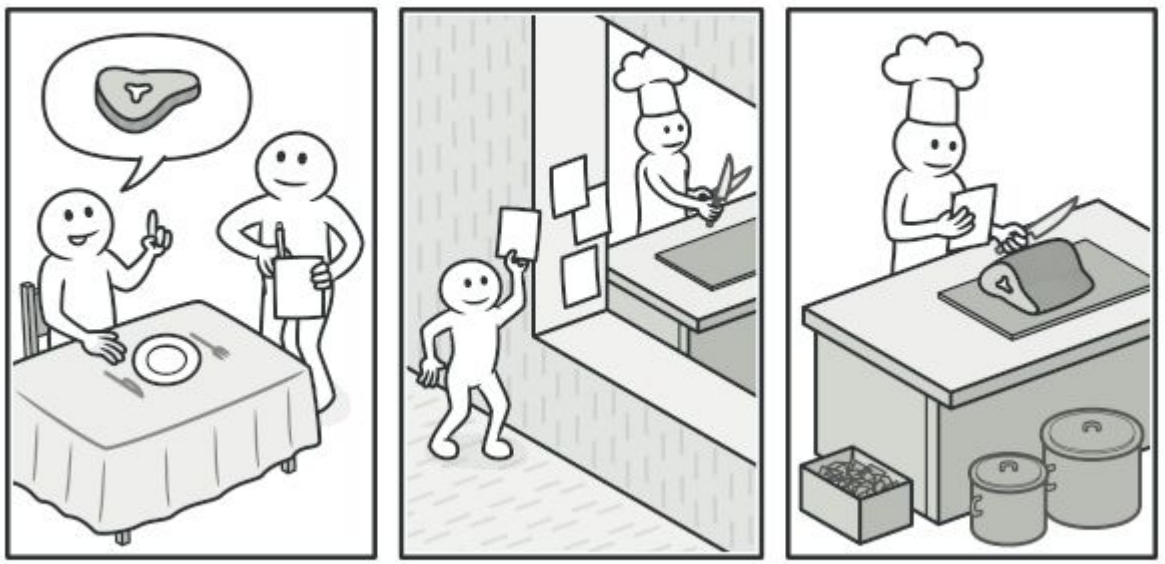
- A Command Pattern says that "encapsulate a request under an object as a command and pass it to invoker object. Invoker object looks for the appropriate object which can handle this command and pass the command to the corresponding object and that object executes the command".
- It is also known as Action or Transaction.

Command Design Pattern

Intent

Command is a behavioural design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request's execution, and support undoable operations.

Real-World Analogy



Making an order in a restaurant.

Applicability

When Would I Use This Pattern?

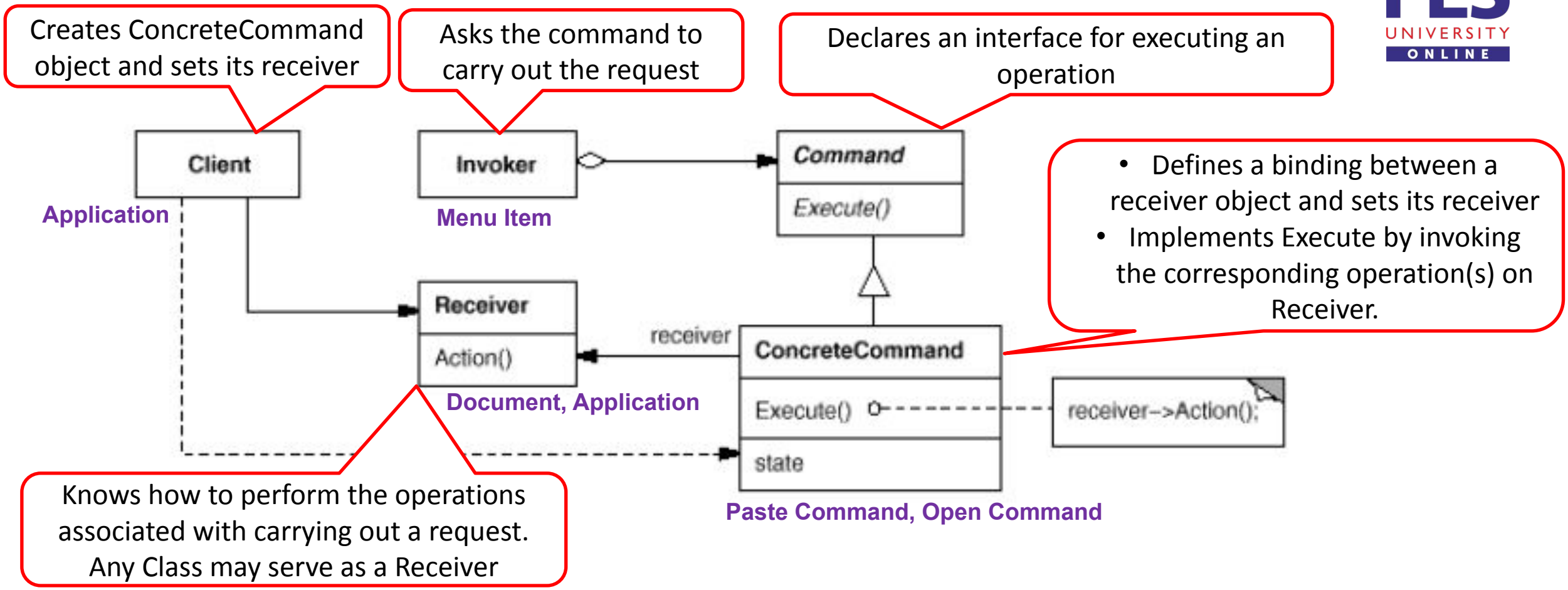
The Command Pattern is useful when:

- ❑ When you need parameterize objects according to an action perform.
- ❑ When you need to create and execute requests at different times.
- ❑ When you need to support rollback, logging or transaction functionality.
- ❑ A history of requests is needed
- ❑ You need callback functionality
- ❑ Requests need to be handled at variant times or in variant orders

The invoker should be decoupled from the object handling the invocation.

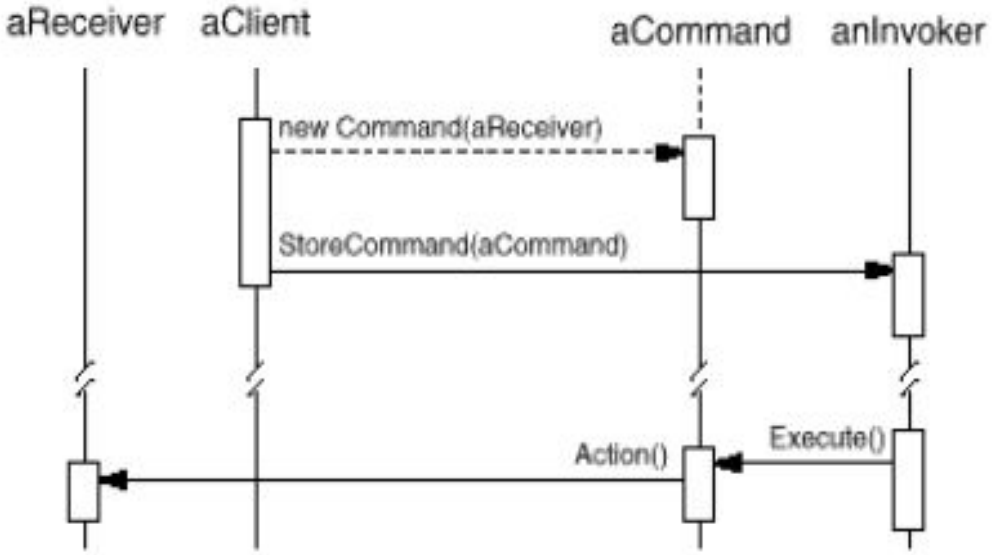
Object Oriented Analysis and Design with Java

Structure



Collaborations

- The client creates a ConcreteCommand object and specifies its receiver
- An Invoker object stores the ConcreteCommand object
- The invoker issues a request by calling Execute on the command. When commands are undoable, ConcreteCommand stores state for undoing the command prior to invoking Execute
- The ConcreteCommand object invokes operations on its receiver to carry out the request



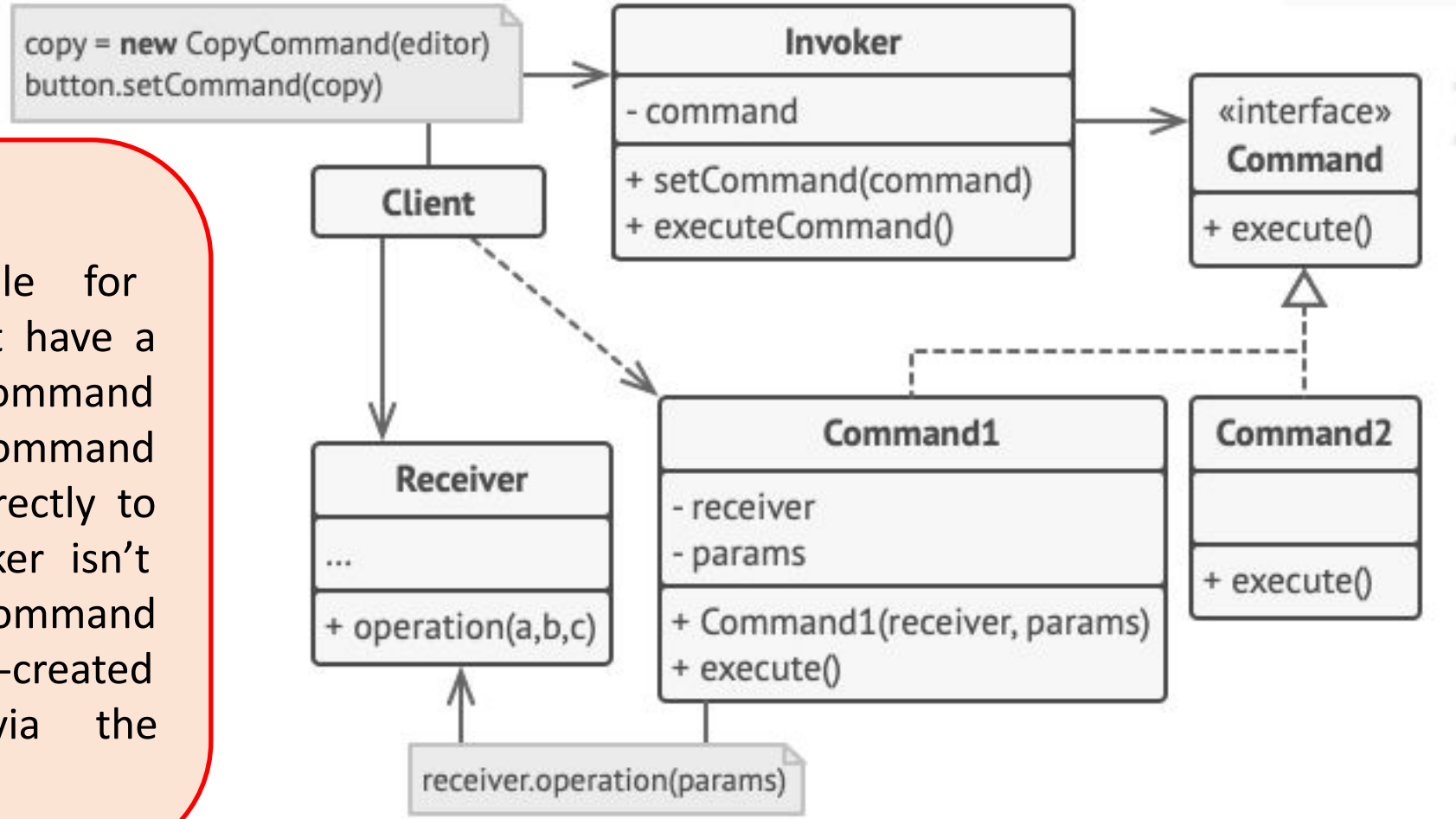
The diagram illustrates how Command decouples the invoker from the receiver (and the request it carries out).

Consequence

- Command decouples the object that invokes the operation from the one that knows how to perform it.
- Commands are first-class objects. They can be manipulated and extended like any other object.
- You can assemble commands into a composite command. In general, composite commands are an instance of the Composite pattern.
- It's easy to add new Commands, because you don't have to change existing classes.

Command: Implementation

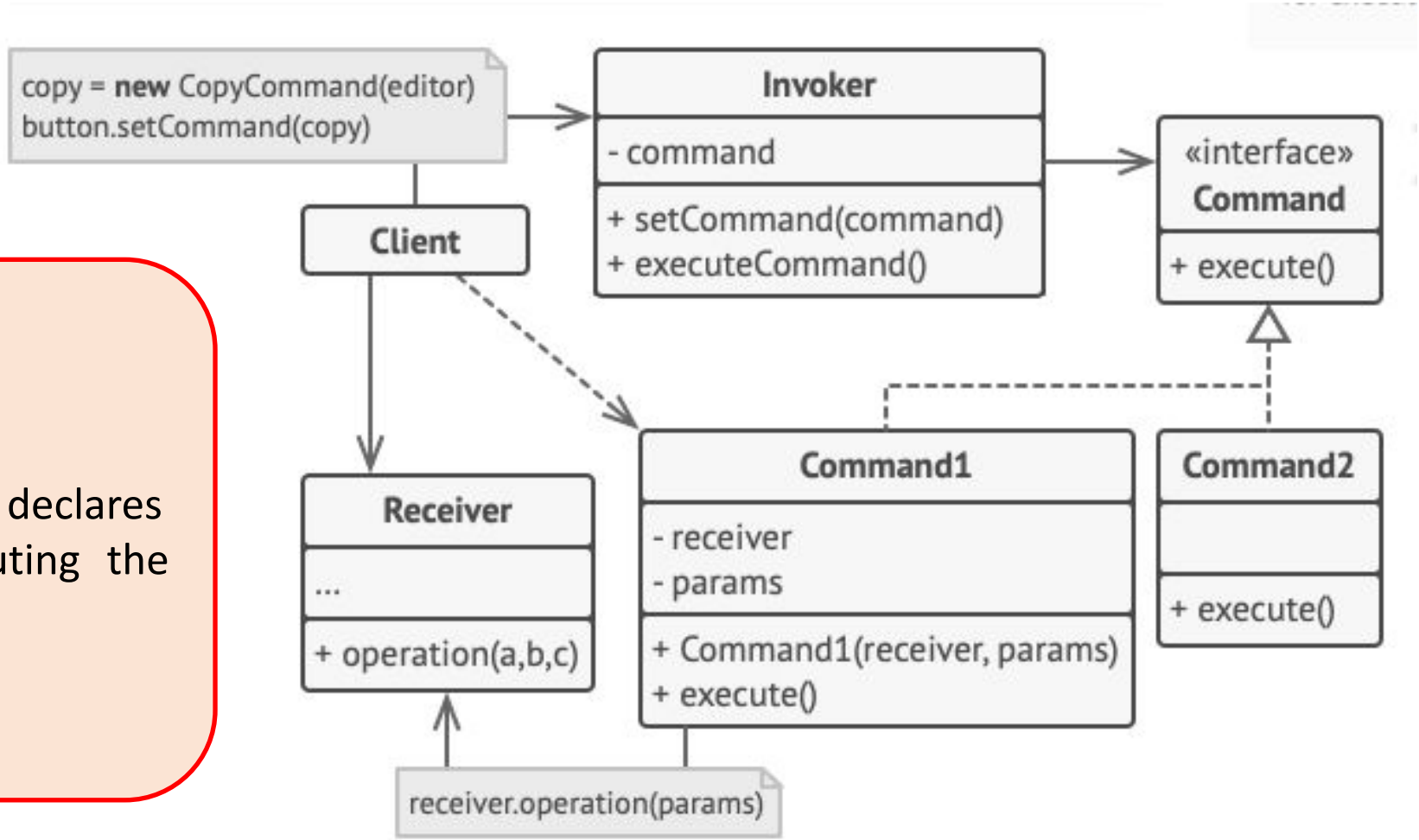
Step1:
The **Invoker** class is responsible for initiating requests. This class must have a field for storing a reference to a command object. The Invoker triggers that command instead of sending the request directly to the receiver. Note that the Invoker isn't responsible for creating the command object. Usually, it gets a pre-created command from the client via the constructor.



Command: Implementation

Step2:

The **Command** interface usually declares just a single method for executing the command.

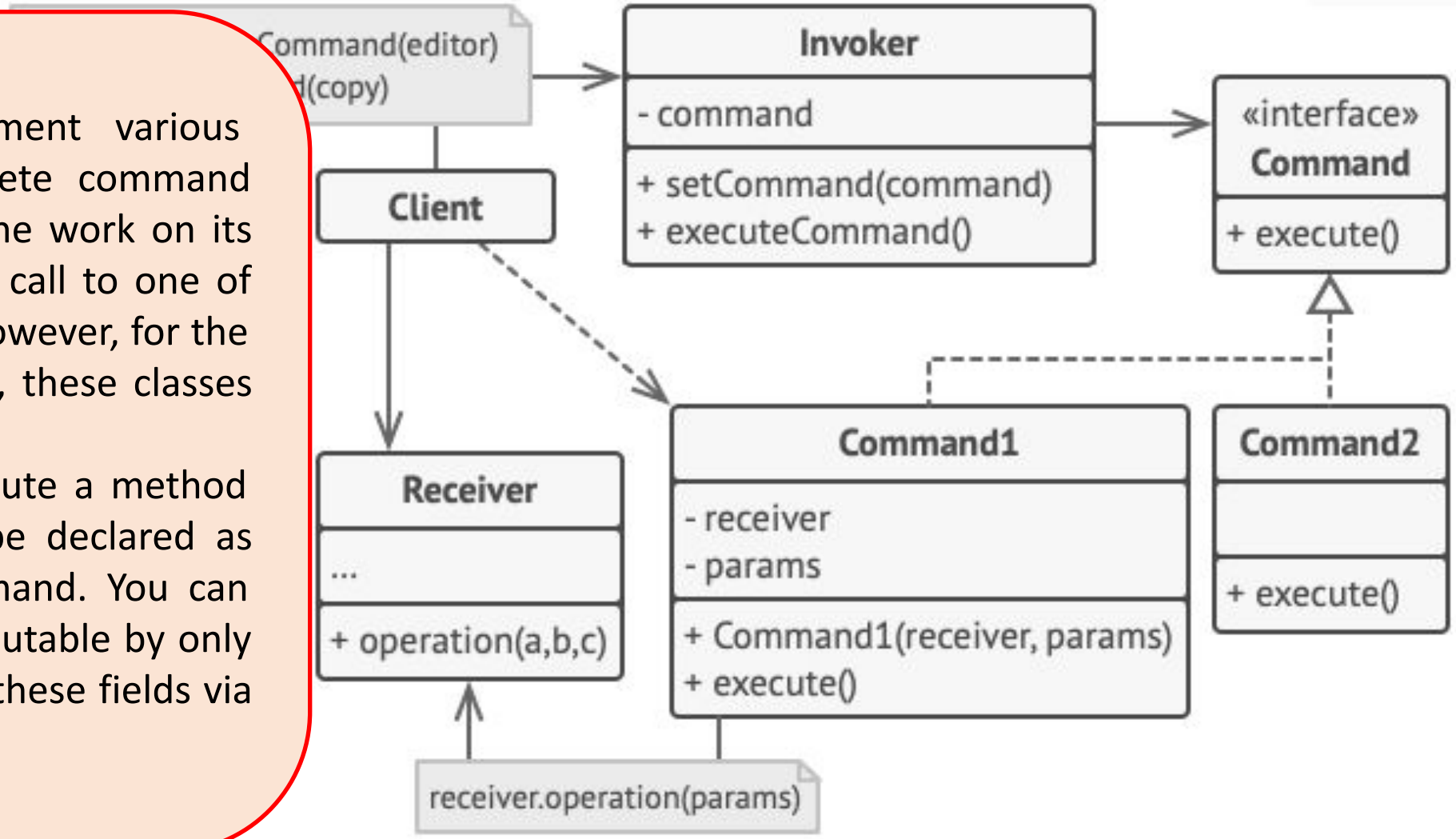


Command: Implementation

Step3:

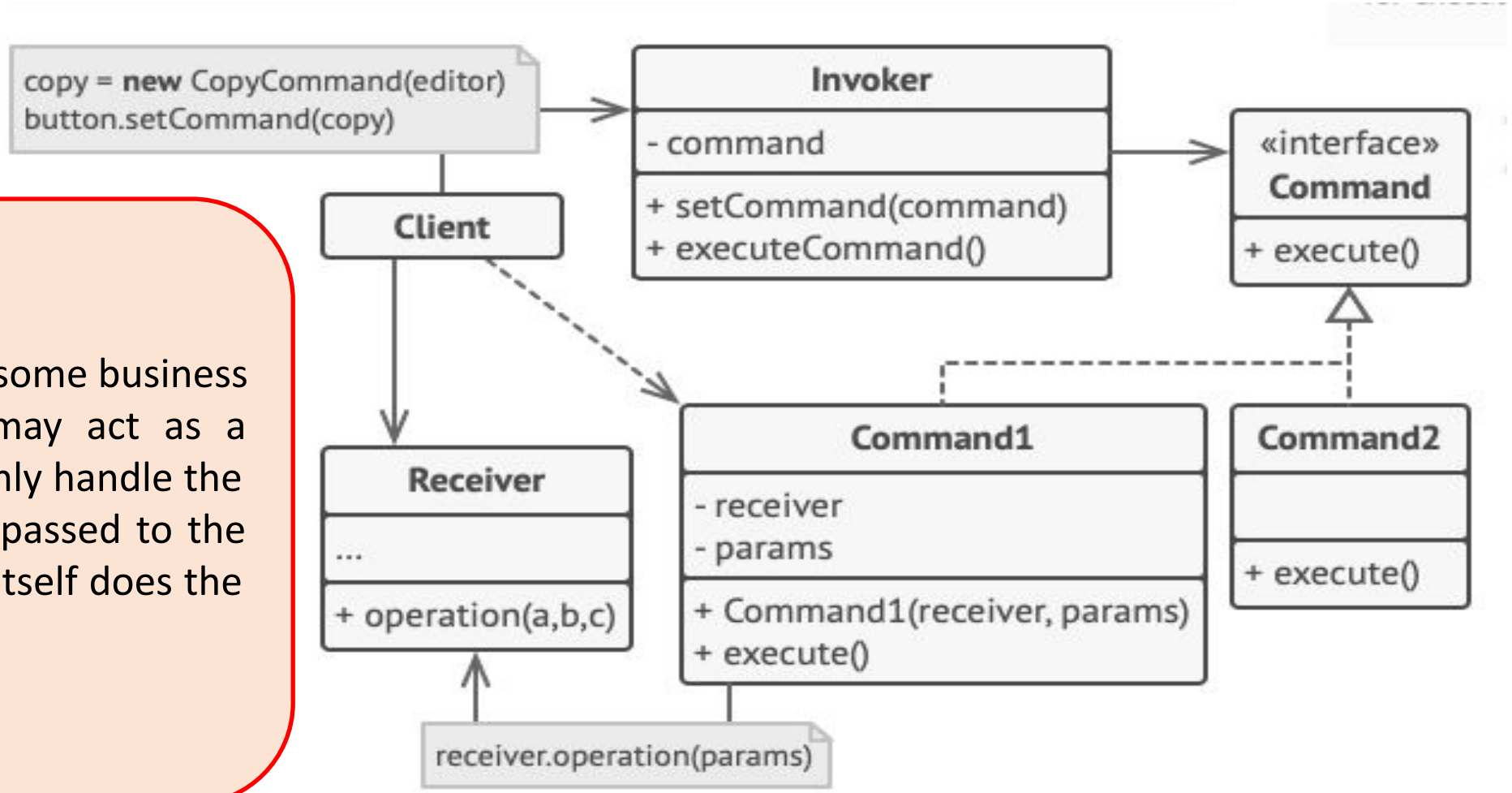
Concrete Commands implement various kinds of requests. A concrete command isn't supposed to perform the work on its own, but rather to pass the call to one of the business logic objects. However, for the sake of simplifying the code, these classes can be merged.

Parameters required to execute a method on a receiving object can be declared as fields in the concrete command. You can make command objects immutable by only allowing the initialization of these fields via the constructor.



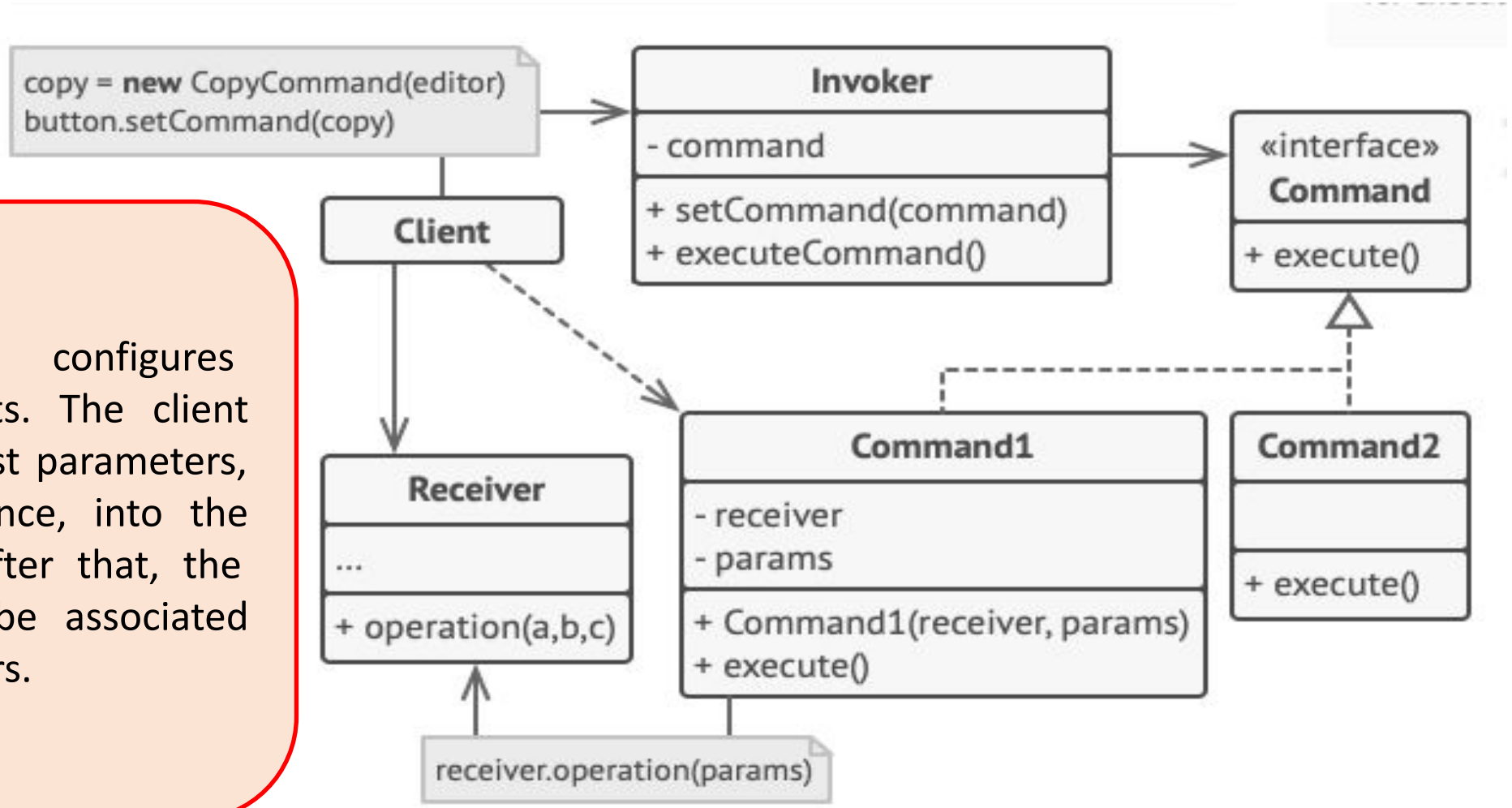
Command: Implementation

Step4:
The **Receiver** class contains some business logic. Almost any object may act as a receiver. Most commands only handle the details of how a request is passed to the receiver, while the receiver itself does the actual work.

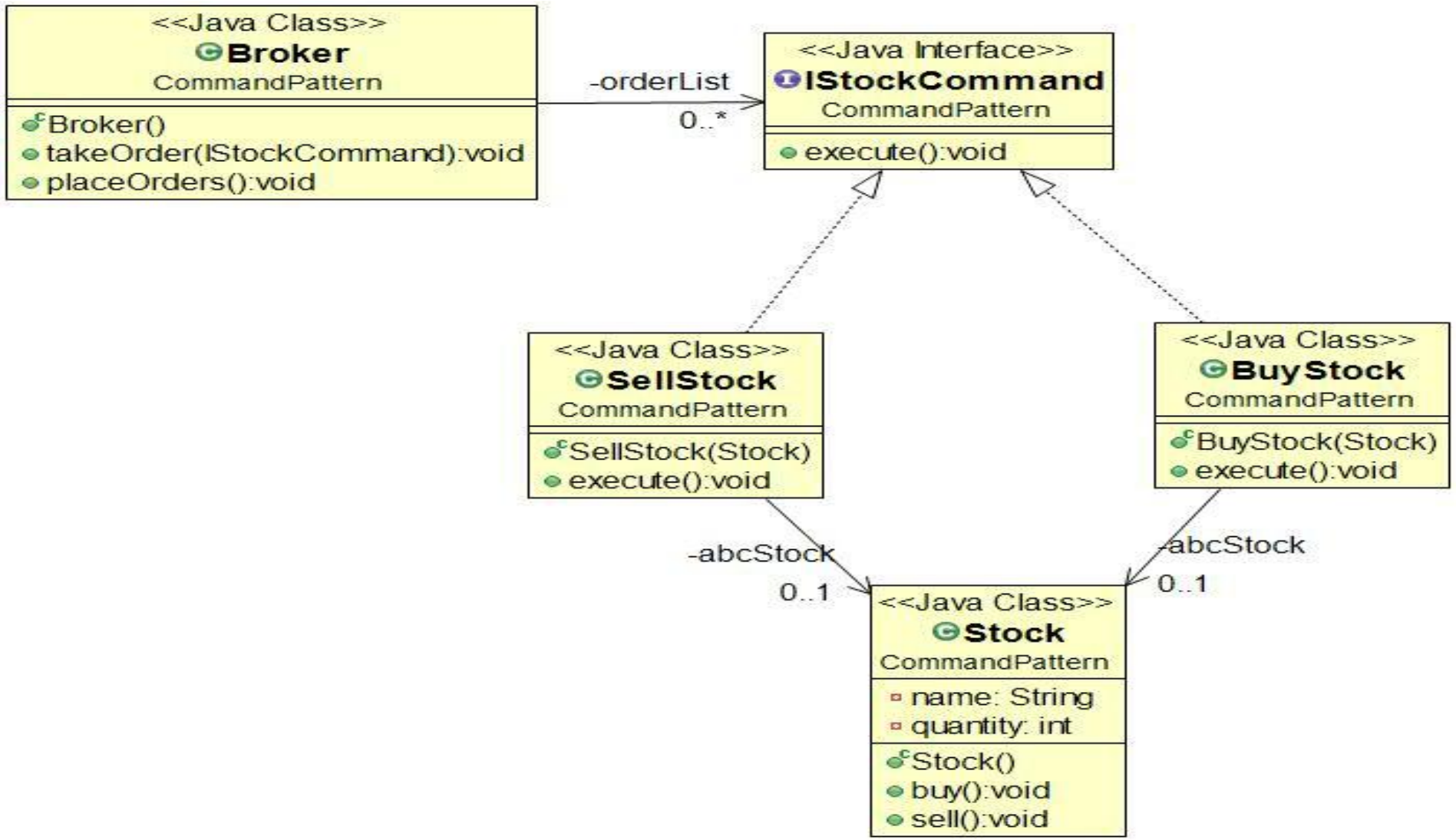


Command: Implementation

Step5:
The **Client** creates and configures concrete command objects. The client must pass all of the request parameters, including a receiver instance, into the command's constructor. After that, the resulting command may be associated with one or multiple invokers.



Command: Implementation



Advantage of command pattern

- It separates the object that invokes the operation from the object that actually performs the operation.
- It makes easy to add new commands, because existing classes remain unchanged.

References

Text Reference

Design Patterns: Elements of Reusable Object-Oriented Software, GOF

Web Reference

https://dzone.com/articles/design-patterns-command
https://refactoring.guru/design-patterns

THANK YOU

Dr. L. Kamatchi Priya

Department of Computer Science and Engineering

priyal@pes.edu