SDET Course

Design Patterns - Command

3 Types of Design Patterns

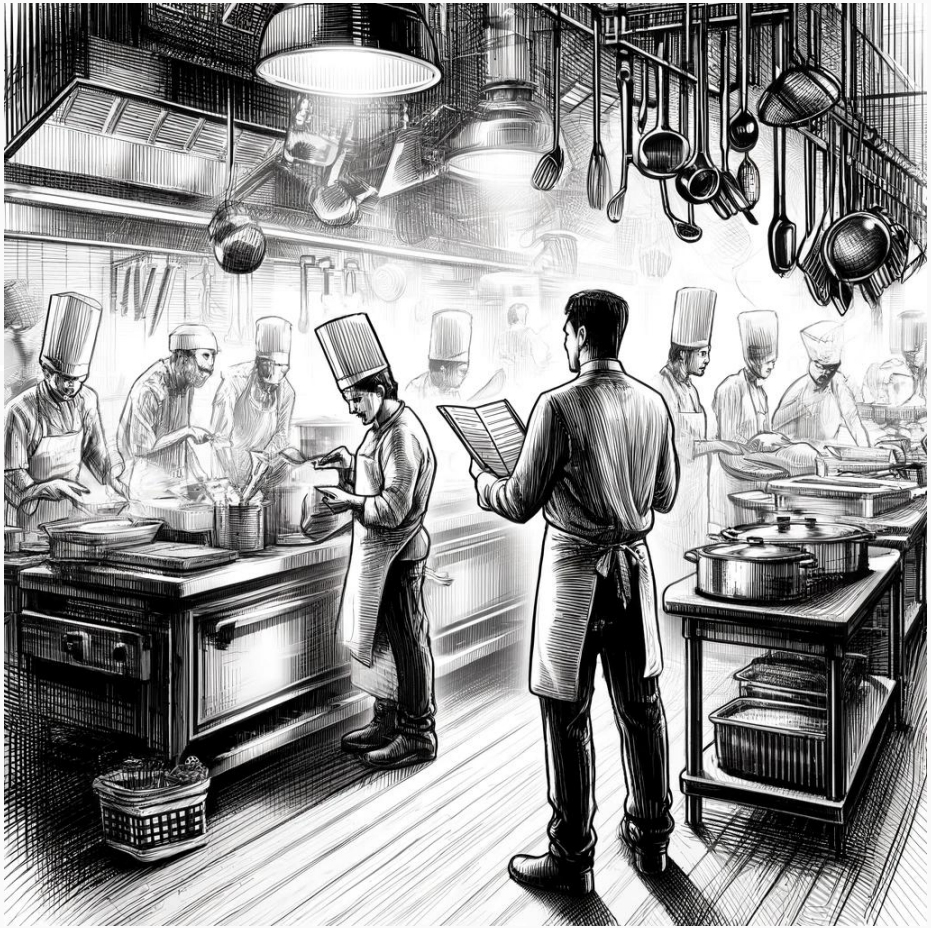
- Creational
  - Singleton
  - Builder
  - Prototype
  - Factory Method
  - Abstract Factory
- Structural
  - Adapter
  - Composite
  - Proxy
  - Flyweight
  - Bridge
  - Facade
  - Decorator
- Behavioral
  - Strategy
  - Observer
  - **Command**
  - Memento
  - State
  - Template Method
  - Mediator
  - Chain of Responsibility
  - Interpreter
  - Visitor
  - Iterator

Agenda

- Description
- Diagram
- Code sample (Java)
- Use cases

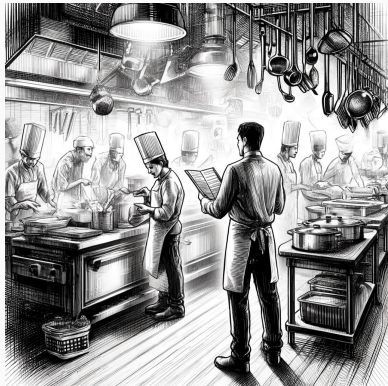
VERISOFT

Descrip
on

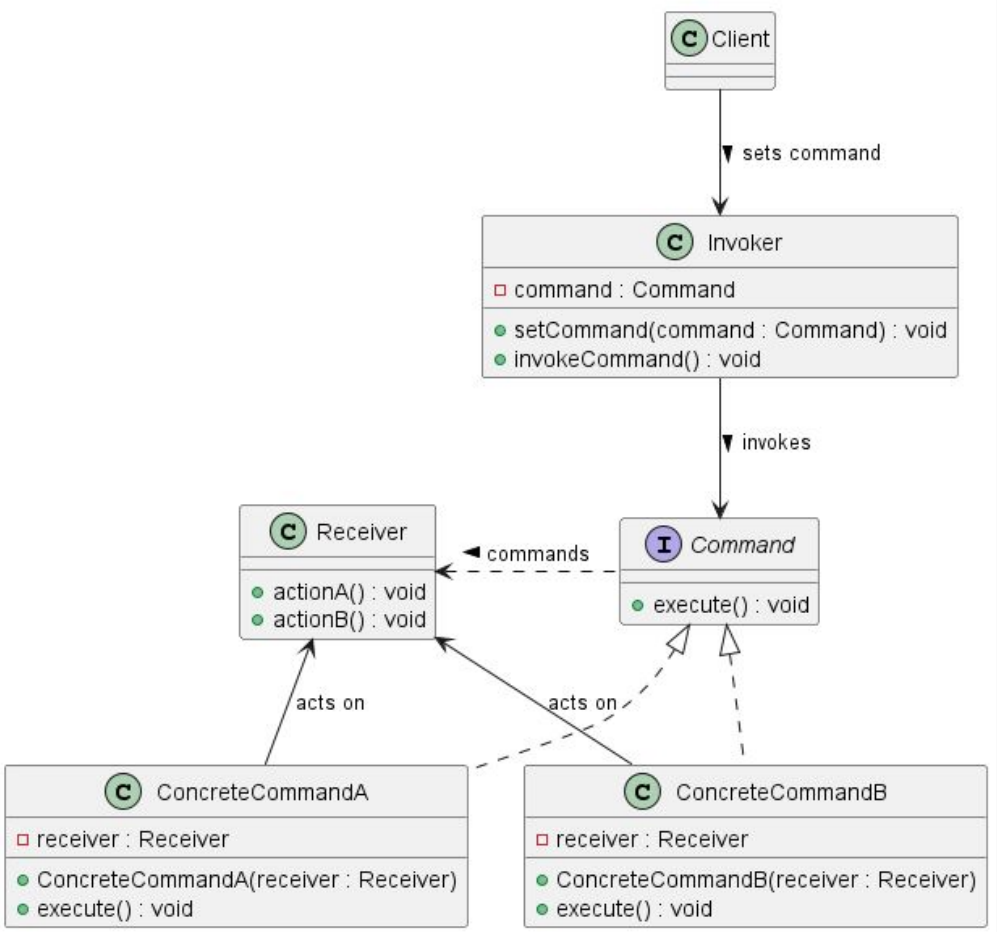


Description

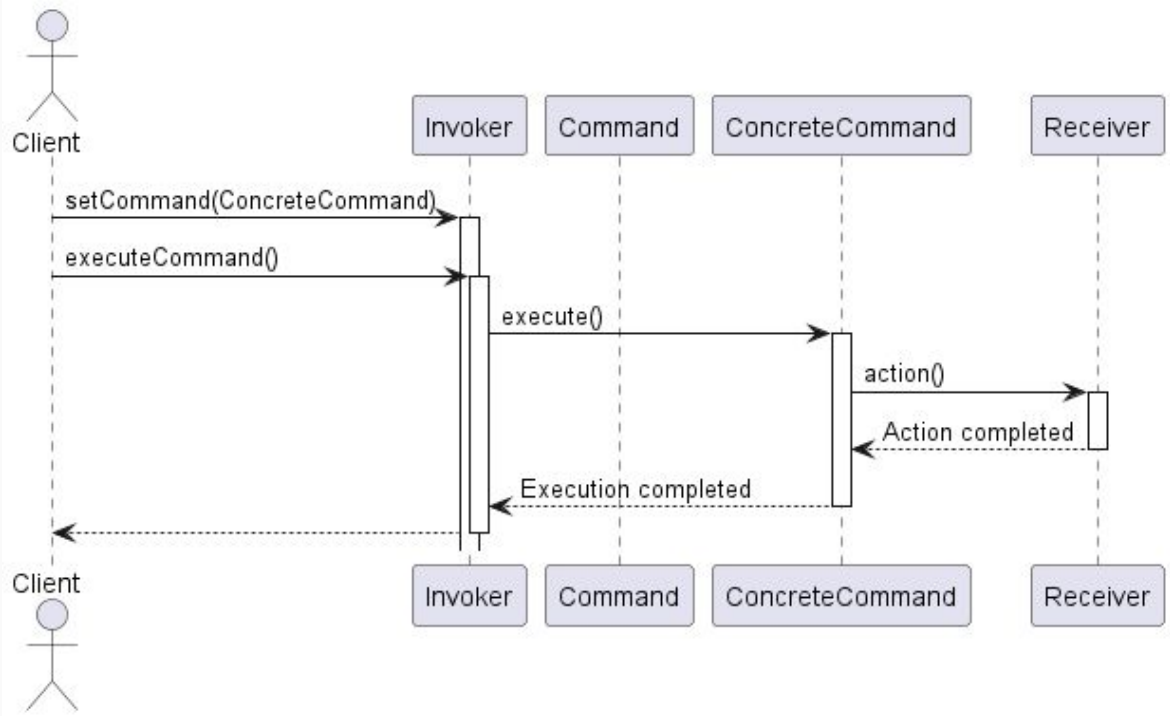
The Command design pattern is a behavioral design pattern that encapsulates a request as an object, thereby allowing users to parameterize clients with queues, requests, and operations. It involves three main components: the command, the receiver, and the invoker. The command component specifies the action to be taken, the receiver is the object that performs the action, and the invoker is the object that knows how to execute the command. This separation of concerns enables decoupling between the objects that send the requests and the objects that receive and execute those requests. By encapsulating a request as an object, it allows for the storage, queuing, and logging of requests, and also provides the capabilities to support undoable operations. The Command pattern is particularly useful in scenarios where actions need to be triggered as a response to a certain event, executed in a specific order, or executed later on.



Class Diagram



Sequence Diagram



Code Sample

Use cases

- General
  - GUI Button Commands
  - Task Scheduling and Background Operations
  - Integration with Smart Home Devices

- In Test Automation
  - Dynamic Test Workflows (Cucumber)
  - Batch Testing
  - API Testing
  - Testing Server

What is Cucumber BDD

- Cucumber BDD is a software development approach that enhances communication, collaboration, and requirements understanding through the use of simple, domain-specific language.
- It enables the creation of executable specifications written in plain language, allowing stakeholders to understand how software behaves without needing to read code.
- Cucumber supports behavior-driven development (BDD) by allowing the definition of application behavior in natural language, bridging the gap between business professionals and developers.

Cucumber BDD Example

```
Feature: Google Search
  Scenario: Searching on Google
    Given I open Google
    When I search for "Cucumber BDD"
    Then I should see results related to "Cucumber BDD"
```

```
public class SearchGoogleSteps {
    WebDriver driver;

    @Given("I open Google")
    public void i_open_google() {
        driver = new ChromeDriver();
        driver.get("https://www.google.com");
    }

    @When("I search for {string}")
    public void i_search_for(String query) {
        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys(query);
        searchBox.submit();
    }

    @Then("I should see results related to {string}")
    public void i_should_see_results_related_to(String query) {
        WebElement resultStats = driver.findElement(By.id("result-stats"));
        assert resultStats.isDisplayed();
        driver.quit();
    }
}
```

Happy Coding