Verisoft

SDET Course

Design Patterns - Adapter

3 Types of Design Patterns

- Creational
  - Singleton
  - Builder
  - Prototype
  - Factory Method
  - Abstract Factory

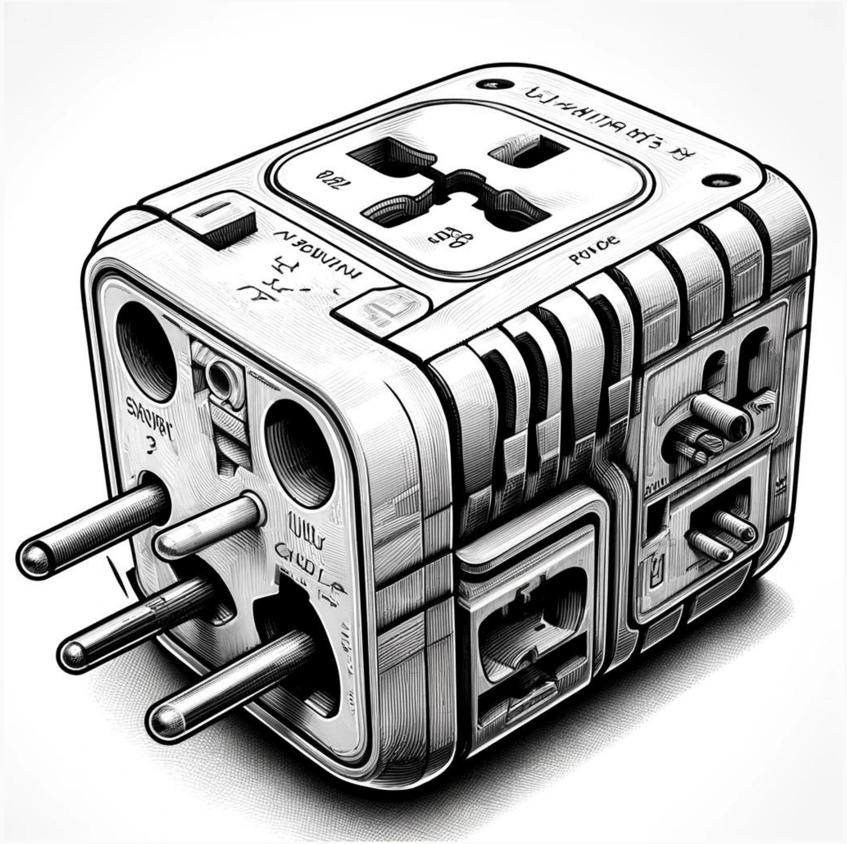
- Structural
  - **Adapter**
  - Composite
  - Proxy
  - Flyweight
  - Bridge
  - Facade
  - Decorator

- Behavioral
  - Strategy
  - Observer
  - Command
  - Memento
  - State
  - Template Method
  - Mediator
  - Chain of Responsibility
  - Interpreter
  - Visitor
  - Iterator

Agenda

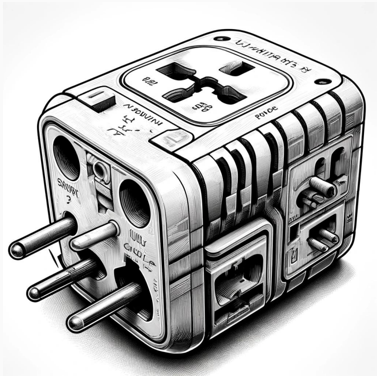
- Description
- Diagram
- Code sample (Java)
- Use cases

The Problem

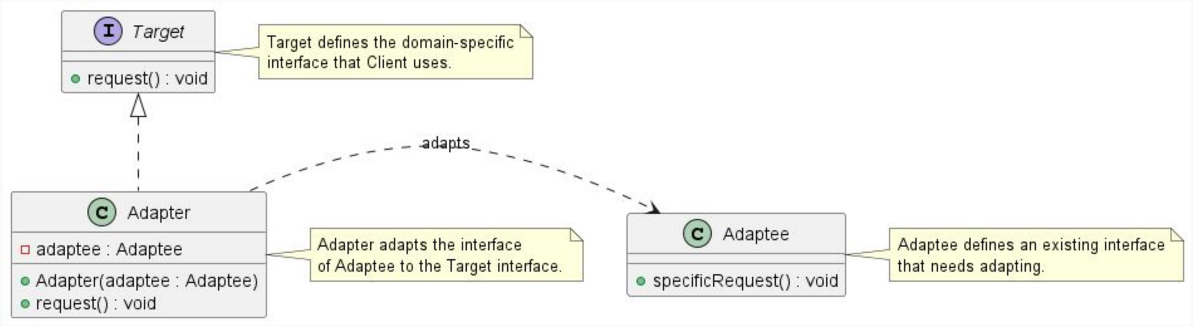


Description

The Adapter design pattern is a structural pattern that allows incompatible interfaces to work together. It acts as a bridge between two incompatible interfaces, converting the interface of a class into another interface that a client expects. By doing so, it enables classes with different interfaces to work together seamlessly. The Adapter pattern is particularly useful when integrating legacy code or third-party libraries into modern systems, as it enables them to collaborate without needing to modify their source code.

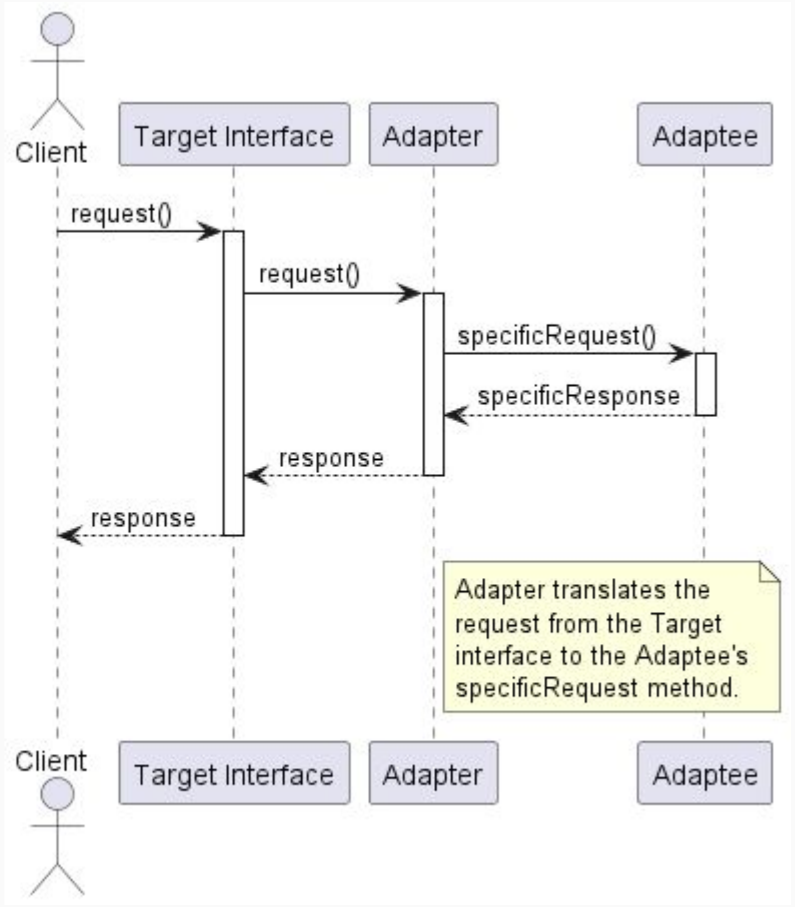


Class Diagram



Target
request() : void
Target defines the domain-specific interface that Client uses.
adapts
Adapter
adaptee : Adaptee
Adapter(adaptee : Adaptee)
request() : void
Adapter adapts the interface of Adaptee to the Target interface.
Adaptee
specificRequest() : void
Adaptee defines an existing interface that needs adapting.

Sequence Diagram



Code Sample

Use cases

- General
  - Legacy Integration
  - Third Party Libraries
  - Cross-Platform applications

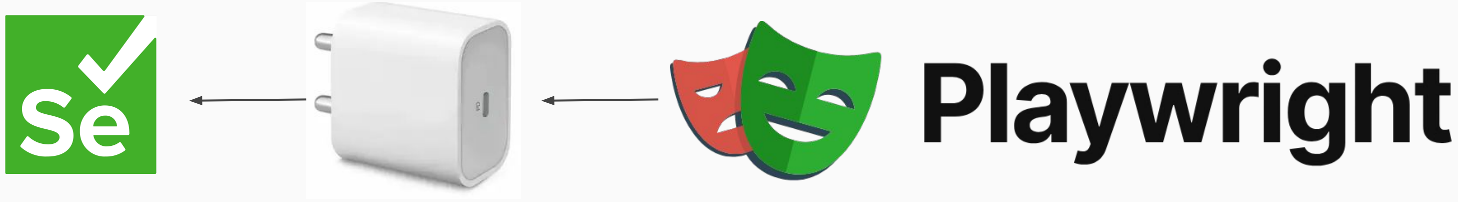
- In Test Automation
  - Legacy test scripts adaptation
  - Data format adaptation (JSON, XML etc.)
  - Appium (vs UIAutomation or XCUITest)

Search Google - Playwright

```
public class GoogleSearchClickPlaywright {
    public static void main(String[] args) {
        try (Playwright playwright = Playwright.create()) {
            BrowserType.LaunchOptions options = new BrowserType.LaunchOptions();
            options.setHeadless(false); // Set to false to see the browser UI
            Browser browser = playwright.chromium().launch(options);

            BrowserContext context = browser.newContext();
            Page page = context.newPage();

            page.navigate("https://www.google.com");
            // Assuming the Google search box has the name attribute "q"
            page.click("input[name='q']");
        }
    }
}
```

Search Google - Selenium

```java
public class GoogleSearchClickSelenium {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        driver.get("https://www.google.com");
        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.click();
    }
}
```



Verisoft

Happy Coding