

OOPS Concepts in Selenium Framework (Detailed + Practical)

📦 Encapsulation – *Data Hiding & Control*

💎 What it is

Encapsulation means **wrapping data and methods together** and **restricting direct access** to them.

💎 How Selenium Framework Uses Encapsulation

- WebDriver instance is kept **private**
- Page elements are hidden inside **Page Objects**
- Access provided via **public methods**

✅ Selenium Example (Page Object Model)

```
java

public class LoginPage {

    private WebDriver driver;

    private By username = By.id("user-name");
    private By password = By.id("password");
    private By loginBtn = By.id("login-button");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    public void login(String user, String pass) {
        driver.findElement(username).sendKeys(user);
        driver.findElement(password).sendKeys(pass);
        driver.findElement(loginBtn).click();
    }
}
```

Encapsulation ensures test scripts don't directly interact with locators, improving maintainability.

2 Abstraction – *Hiding Implementation Details*

What it is

Abstraction exposes what to do, not how to do it.

Where Selenium Uses Abstraction

- WebDriver is an interface
- Browser-specific drivers are implementations
- Frameworks use abstract base classes for common flows

✓ Selenium Example – WebDriver Abstraction

```
java

WebDriver driver = new ChromeDriver();
```

✓ Framework-Level Abstraction

```
java

public abstract class BaseTest {

    protected WebDriver driver;

    abstract void setUp();

    public void tearDown() {
        driver.quit();
    }
}
```

📖 Inheritance – *Reusability*

💎 What it is

Inheritance allows a child class to reuse properties and methods of a parent class.

💎 Selenium Framework Usage

- **BaseTest** → Test classes
- **BasePage** → All page objects
- **Common utilities** inherited

✅ Selenium Example

java

```
public class BaseTest {  
    protected WebDriver driver;  
  
    public void initDriver() {  
        driver = new ChromeDriver();  
    }  
}
```

java

```
public class LoginTest extends BaseTest {  
  
    @Test  
    public void loginTest() {  
        initDriver();  
        driver.get("https://example.com");  
    }  
}
```



Inheritance avoids code duplication and centralizes WebDriver setup.

4 Polymorphism – *Multiple Forms*

◆ What it is

Polymorphism allows the same method to behave differently at runtime.

◆ Selenium Usage

- Same WebDriver reference → different browser behavior
- Overriding framework methods
- Runtime decision of driver type

✓ Selenium Example

```
java

WebDriver driver;

if (browser.equals("chrome")) {
    driver = new ChromeDriver();
} else {
    driver = new FirefoxDriver();
}
```

WebDriver exhibits runtime polymorphism.

5 Interface – *Contract*

◆ Selenium Uses Interfaces Heavily

- WebDriver
- SearchContext
- JavascriptExecutor
- TakesScreenshot

✓ Selenium Example

```
java

TakesScreenshot ts = (TakesScreenshot) driver;
File src = ts.getScreenshotAs(OutputType.FILE);
```

Interfaces allow Selenium to support multiple browsers consistently.

❏ Composition (HAS-A relationship) – *Preferred over inheritance*

💎 What it is

One class contains another class.

💎 Selenium Framework Best Practice

Page Objects contain WebDriver, not extend it.

✅ Selenium Example

```
java

public class HomePage {
    private WebDriver driver;

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }
}
```

Composition is more flexible and less tightly coupled than inheritance.

❏ Constructor Usage

💎 Why important in Selenium

- Driver injection
- PageFactory initialization
- Dependency management

✅ Selenium Example

```
java

public LoginPage(WebDriver driver) {
    this.driver = driver;
    PageFactory.initElements(driver, this);
}
```

8 Access Modifiers (Framework Design)

Modifier	Usage
<code>private</code>	Locators, driver
<code>protected</code>	Base classes
<code>public</code>	Page actions
<code>default</code>	Package utilities

9 Design Patterns Using OOPS

Selenium frameworks rely on OOPS-based patterns:

- Page Object Model (Encapsulation)
- Factory Pattern (Polymorphism)
- Singleton (Controlled object creation)
- Strategy Pattern (Browser selection)