**OOPs Concepts in Java Selenium Framework**

**1. Encapsulation**

What Is It?

Encapsulation means bundling variables (like locators) and methods (like actions) in a class and restricting outside access to internal data using access modifiers (usually private).

Real-Time Selenium Example:

public class SearchPage {

private WebDriver driver;

@FindBy(name = "q") private WebElement searchBox;

@FindBy(name = "btnK") private WebElement searchButton;

public SearchPage(WebDriver driver) {

this.driver = driver;

PageFactory.initElements(driver, this);

}

public void search(String keyword) {

searchBox.sendKeys(keyword);

searchButton.click();

}

}

Interview Explanation:

"In my framework, I encapsulate all WebElements as private and only expose public methods to perform actions. This prevents direct access to locators from test classes, ensuring cleaner, safer, and more maintainable code. If anything changes in the UI, I only update the page class — not the test cases."

**2. Inheritance**

What Is It?

Inheritance lets one class reuse code from another class using the extends keyword. Common use cases in Selenium include sharing setup/teardown logic across tests.

Real-Time Selenium Example:

public class BaseTest {

protected WebDriver driver;

@BeforeMethod

public void init() {

driver = new ChromeDriver();

driver.manage().window().maximize();

}

@AfterMethod

public void tearDown() {

driver.quit();

}

}

public class LoginTest extends BaseTest {

@Test

public void verifyLogin() {

LoginPage login = new LoginPage(driver);

login.login("Ajay", "password123");

}

}

Interview Explanation:

I created a BaseTest class to handle WebDriver setup and teardown. All test classes extend this base class to reuse that logic. This follows the DRY principle and improves test consistency."

**3. Abstraction**

What Is It?

Abstraction hides internal complexity and only shows necessary behavior. In Selenium, it’s often used to abstract actions like click, type, and wait so testers can focus on test flow, not implementation.

Real-Time Selenium Example:

public interface ElementActions {

void click(WebElement element);

void type(WebElement element, String text);

void waitForVisible(WebElement element, int timeoutInSeconds);

}

public class SeleniumActions implements ElementActions {

private WebDriver driver;

public SeleniumActions(WebDriver driver) {

this.driver = driver;

}

public void click(WebElement element) {

waitForVisible(element, 10);

element.click();

}

public void type(WebElement element, String text) {

waitForVisible(element, 10);

element.clear();

element.sendKeys(text);

}

public void waitForVisible(WebElement element, int timeoutInSeconds) {

new WebDriverWait(driver, Duration.ofSeconds(timeoutInSeconds))

.until(ExpectedConditions.visibilityOf(element));

}

}

Interview Explanation:

"Instead of writing wait and click logic repeatedly in tests, I created a utility class implementing an ElementActions interface. This hides the internal complexity and gives testers clean methods to use. It also improves reusability and test stability."

**4. Polymorphism**

Here is the updated Polymorphism section with a different example — we'll demonstrate how multiple browser drivers (like Chrome, Firefox) can be used interchangeably through a common interface, a common real-world use case in Selenium frameworks.

---

4. Polymorphism (Updated Example)

🔷 What Is It?

Polymorphism allows one interface to be implemented in multiple ways. It helps us use different classes through a common reference, enabling flexible and extensible code.

🔷 Real-Time Selenium Example:

public interface DriverManager {

WebDriver getDriver();

}

public class ChromeManager implements DriverManager {

public WebDriver getDriver() {

System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

return new ChromeDriver();

}

}

public class FirefoxManager implements DriverManager {

public WebDriver getDriver() {

System.setProperty("webdriver.gecko.driver", "path/to/geckodriver");

return new FirefoxDriver();

}

}

// Runtime decision based on environment

DriverManager driverManager;

String browser = System.getProperty("browser");

if ("firefox".equalsIgnoreCase(browser)) {

driverManager = new FirefoxManager();

} else {

driverManager = new ChromeManager();

}

WebDriver driver = driverManager.getDriver();

🔷 Interview Explanation:

> "I use polymorphism to dynamically select the browser driver implementation (like Chrome or Firefox) at runtime using a common interface. This allows my test framework to be easily scalable and environment-independent. I just change a property and the underlying driver changes — no test code modifications required."

5. Class and Object

🔷 What Is It?

A class is a blueprint or template; an object is the instance of that class used to perform real actions.

🔷 Real-Time Selenium Example:

@Test

public void verifyCartPage() {

HomePage home = new HomePage(driver);

CartPage cart = new CartPage(driver);

home.searchProduct("Laptop");

cart.addToCart("Laptop");

}

🔷 Interview Explanation:

> "I use classes like HomePage, LoginPage, and CartPage to organize UI elements and actions. In tests, I create objects of these classes to interact with web elements. This makes the test logic readable and object-oriented."

---

🧠 Best Practices for Interviews

Encapsulation: Always keep WebElements private and expose only behavior via public methods.

Inheritance: Use it for common setup like driver init or reporting logic.

Abstraction: Create utility layers for actions/waits/logging to simplify test scripts.

Polymorphism: Use for runtime decision-making (e.g., real vs. mock or web vs. mobile).

Class/Object: Design test scripts using object-oriented principles for modularity.

---

✅ Summary Table

OOP Concept Selenium Example Interview Key Point

Encapsulation Page classes with private locators, public methods "Hides element details and exposes only safe operations."

Inheritance BaseTest for WebDriver setup/cleanup "Promotes reuse and reduces duplication."

Abstraction Utility class for click/type/wait "Hides complex WebDriver logic, improves test readability."

Polymorphism Switching between SeleniumActions and MockActions "Enables runtime behavior change for different test environments."

Class/Object Using LoginPage and CartPage as objects in tests "Builds a modular, readable, object-oriented test framework."