

Selenium

1. What are the challenges you have faced during testing?

Some of the common challenges in automation testing include:

- **Dynamic Elements:** Many web elements have dynamic attributes (e.g., dynamic IDs), making them difficult to locate.
 - **Handling Pop-ups & Alerts:** Different types of pop-ups (JavaScript alerts, authentication pop-ups) require different handling strategies.
 - **Synchronization Issues:** Some elements load asynchronously, requiring proper wait strategies (Implicit, Explicit, Fluent Waits).
 - **Cross-browser Compatibility:** Ensuring tests work on different browsers (Chrome, Firefox, Edge) with varying behaviors.
 - **Test Data Management:** Maintaining valid and reusable test data for automation scripts.
 - **Element Staleness:** Elements become stale when the page refreshes, leading to `StaleElementReferenceException`.
 - **Flaky Tests:** Tests sometimes pass and sometimes fail due to inconsistent test environments, timing issues, or dependency on external data.
 - **CI/CD Integration:** Ensuring automation scripts run seamlessly in Continuous Integration pipelines like Jenkins.
 - **API Testing Integration:** Often, UI automation needs to be integrated with API tests for full test coverage.
-

2. What strategies did you follow while building a Selenium framework from scratch?

Building a Selenium framework from scratch requires careful planning and structuring. The key strategies include:

- **Selecting a Design Pattern:**
 - **Page Object Model (POM):** To keep locators and test logic separate.
 - **Page Factory:** To initialize elements efficiently.
 - **Singleton Design Pattern:** To manage a single WebDriver instance.
- **Choosing an Appropriate Framework:**
 - **TestNG/JUnit:** For test execution and reporting.
 - **Maven/Gradle:** For dependency management.
 - **Cucumber:** If using BDD for business-readable test cases.
- **Implementing Utilities:**
 - **Logger (Log4j/SLF4J):** For detailed logging.
 - **Configuration Manager:** Reading properties from external config files.
 - **Custom Wait Mechanism:** Handling synchronization effectively.
- **CI/CD Integration:**
 - Using **Jenkins/GitHub Actions** to run automation in pipelines.

- **Parallel Execution:**
 - **Grid/Selenoid/Cloud Testing (Sauce Labs, BrowserStack):** To run tests in different environments.

3. Where do you perform the Singleton Design Pattern? If you don't use it, do you have an idea about this?

Singleton Design Pattern ensures that only one instance of a class is created and provides a global point of access to it.

- In **Selenium Framework**, Singleton Pattern is often used for **WebDriver instance management**. Instead of creating multiple WebDriver instances, we create one shared instance for the entire test execution.

Example of Singleton WebDriver Implementation:

```
public class WebDriverSingleton {
    private static WebDriver driver;

    private WebDriverSingleton() { } // Private constructor to restrict instantiation

    public static WebDriver getDriver() {
        if (driver == null) {
            driver = new ChromeDriver(); // Or fetch browser type from properties
        }
        return driver;
    }
}
```

Calling it in the test:

```
WebDriver driver = WebDriverSingleton.getDriver();
```

4. Difference between Implicit, Explicit, and Fluent Waits in Selenium?

Wait Type	Definition	Behavior
Implicit Wait	Global wait that applies to all elements.	Throws <code>NoSuchElementException</code> if the element is not found within the time.
Explicit Wait	Waits for a specific condition before proceeding.	Used for specific elements using <code>ExpectedConditions</code> .
Fluent Wait	Similar to Explicit Wait but allows polling frequency and ignores exceptions.	Provides more control over polling time.

5. Pros and Cons of Implicit Wait and Explicit Wait

Type	Pros	Cons
Implicit Wait	- Simple to implement - Applies globally to all elements	- Can cause unnecessary delays - Doesn't handle conditions like visibility
Explicit Wait	- Efficient and works for specific elements - Reduces script execution time	- Must be used carefully to avoid long execution time

6. Why do we prefer Explicit Wait instead of Fluent Wait? What are the disadvantages of Fluent Wait?

- **Explicit Wait is preferred** because:
 - It is simple and widely used.
 - It provides predefined conditions in `ExpectedConditions`.
- **Disadvantages of Fluent Wait:**
 - Requires manual handling of polling intervals.
 - Increases script complexity.

7. Without Implicit Wait, will the Selenium script work?

Yes, the script will work, but it might fail if elements take time to load. In such cases, Explicit Wait should be used.

8. What is the default polling time in Explicit and Implicit Wait?

- **Implicit Wait:** It continuously checks for an element within the specified time.
- **Explicit Wait:** Default polling time is **500 milliseconds**.

9. Explain synchronization in Selenium?

Synchronization ensures that the test script waits for the web application to be in the correct state before interacting with elements. It can be:

- **Implicit Wait** (global wait)
- **Explicit Wait** (condition-based wait)
- **Thread.sleep()** (not recommended)

10. Which concept is implemented in Explicit and Fluent Wait?

- **Explicit Wait:** Implements `ExpectedConditions` class.
 - **Fluent Wait:** Implements `Function<T, R>` interface from `java.util.function`.
-

11. Explain Abstraction and Interface with respect to Selenium.

- **Abstraction:** Hides the implementation details and only exposes necessary functionalities.
- **Interface in Selenium:** `WebDriver` is an interface, and classes like `ChromeDriver`, `FirefoxDriver` implement it.

Example:

```
WebDriver driver = new ChromeDriver(); // Abstraction
```

12. Difference between Factory Design and Singleton Framework?

Feature	Factory Design Pattern	Singleton Pattern
Purpose	Creates objects based on conditions	Ensures a single instance of a class
Example	Browser Factory	WebDriver Singleton

13. What is Page Object and Page Factory Model?

- **Page Object Model (POM):** Separates UI elements from test scripts.
- **Page Factory:** Uses `@FindBy` annotations to initialize elements efficiently.

Example:

```
public class LoginPage {
    @FindBy(id = "username")
    WebElement username;

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

14. Have you used Interface in your framework other than Selenium Interfaces?

Yes, for example:

- **Custom Reporting Interface**
 - **Database Connection Interface**
 - **Custom Utility Interface**
-

15. How do you achieve Inheritance in your framework?

By creating a **Base Class** and inheriting it in test classes:

```
public class BaseTest {  
    WebDriver driver;  
    public void setup() { driver = new ChromeDriver(); }  
}  
  
public class LoginTest extends BaseTest {  
    @Test  
    public void testLogin() {  
        driver.get("https://example.com");  
    }  
}
```

16. What is WebDriver? Name methods without implementation.

- WebDriver is an **interface** in Selenium.
 - Methods without implementation:
 - `get()`
 - `findElement()`
 - `findElements()`
-

17. Fastest Locator in Selenium?

- **ID** is the fastest locator.
-

18. What does :: (double colon) in XPath represent?

- Used in **Axes** (e.g., `following-sibling::`).
-

19. Explain `driver.manage().window().maximize()`

- Uses the **Options Interface** in Selenium to manage browser settings.
-

20. Difference between `get()` and `navigate().to()`

Method	Behavior
<code>get()</code>	Loads a new web page.
<code>navigate().to()</code>	Allows forward and backward navigation.

21. How would you check for broken links on a webpage?

To check for broken links in Selenium, follow these steps:

1. Fetch all anchor (`<a>`) tags.
2. Extract the `href` attribute.
3. Send an HTTP request to the extracted URL.
4. Verify the HTTP response code.

Example Code:

```
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;

public class BrokenLinksChecker {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        List<WebElement> links = driver.findElements(By.tagName("a"));
        for (WebElement link : links) {
            String url = link.getAttribute("href");
            checkBrokenLink(url);
        }
        driver.quit();
    }

    public static void checkBrokenLink(String url) {
        try {
            HttpURLConnection conn = (HttpURLConnection) new
            URL(url).openConnection();
            conn.setRequestMethod("HEAD");
            conn.connect();
            int responseCode = conn.getResponseCode();
            if (responseCode >= 400) {
                System.out.println(url + " is a broken link");
            }
        } catch (Exception e) {
            System.out.println(url + " is a broken link");
        }
    }
}
```

```
}  
}  
}
```

- Valid links return HTTP 200.
- Broken links return HTTP 400+ or throw exceptions.

22. Difference between `submit()` and `click()` in Selenium?

Method	Usage	Works On
<code>click()</code>	Clicks any clickable element	Buttons, links, radio buttons
<code>submit()</code>	Submits a form	Only on <code><form></code> elements

Example:

```
driver.findElement(By.id("submitButton")).click(); // Works everywhere  
driver.findElement(By.id("searchBox")).submit(); // Only works inside a  
form
```

23. Difference between absolute XPath (/) and relative XPath (//)?

Type	Syntax	Usage
Absolute XPath	<code>/html/body/div[1]/table/tr[2]/td[1]</code>	Starts from root, rigid structure
Relative XPath	<code>//table[@id='example']/tr/td[1]</code>	Can start from any node, flexible

Example:

```
// Absolute XPath  
driver.findElement(By.xpath("/html/body/div[1]/input")).click();  
  
// Relative XPath  
driver.findElement(By.xpath("//input[@id='search']")).click();
```

- **Relative XPath is recommended** since it is **more robust**.
-

24. Difference between `findElement()` and `findElements()`?

Method	Return Type	Exception if Element Not Found
<code>findElement()</code>	<code>WebElement</code> (Single element)	<code>NoSuchElementException</code>
<code>findElements()</code>	<code>List<WebElement></code> (Multiple elements)	Returns an empty list

Example:

```
WebElement element = driver.findElement(By.id("uniqueID")); // Throws exception if not found
```

```
List<WebElement> elements = driver.findElements(By.className("someClass"));  
// Returns empty list if not found
```

25. Difference between Frames and iFrames?

Feature	Frame	iFrame
Definition	A separate document inside a webpage	An inline frame inside another webpage
Usage	Less common now (old HTML <code><frame></code> tag)	Commonly used in modern web apps
Selenium Handling	<code>driver.switchTo().frame()</code>	<code>driver.switchTo().frame()</code>

Example:

```
driver.switchTo().frame("frameName");
```

26. Return Type of `findElement()` and `findElements()`?

- `findElement()` → Returns a **single WebElement**.
 - `findElements()` → Returns a **List of WebElements**.
-

27. What error will be thrown when no element is found for `findElement()` and `findElements()`?

Method	Exception/Output
<code>findElement()</code>	Throws <code>NoSuchElementException</code>
<code>findElements()</code>	Returns an empty list (<code>List<WebElement></code>)

28. Exceptions Faced in Selenium (Java Exceptions Included)

Selenium Exceptions:

- **`NoSuchElementException`** – Element not found.
- **`TimeoutException`** – Explicit wait timeout.
- **`StaleElementReferenceException`** – Element is no longer valid.
- **`ElementClickInterceptedException`** – Another element is covering the clickable element.

Java Exceptions:

- **`NullPointerException`** – Accessing an object that is not initialized.
- **`ArithmeticException`** – Divide by zero error.
- **`ArrayIndexOutOfBoundsException`** – Accessing an invalid index in an array.

29. How to handle `StaleElementReferenceException`?

Solution:

1. **Re-locate the element** before interacting.
2. **Use Explicit Wait** to wait for element reload.

Example:

```
java
CopyEdit
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement element =
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("someID")));
```

30. Interfaces used in Selenium?

Some important interfaces:

- **`WebDriver`**
- **`JavascriptExecutor`**

- TakesScreenshot
 - Alert
 - WebElement
-

31. Where is Inheritance used in Selenium?

Inheritance is used in:

- **BaseTest Class:** Common setup/teardown for all tests.
- **Page Object Model (POM):** Pages inherit reusable methods.

Example:

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

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

32. How do you initialize web elements in POM? What happens if not initialized?

- Use `PageFactory.initElements()` in the constructor.

Example:

```
java
CopyEdit
public class LoginPage {
    @FindBy(id="username") WebElement username;

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

- **If not initialized,** `NullPointerException` occurs.
-

33. If both Implicit and Explicit Waits are used, which takes priority?

- **Explicit Wait takes priority** over Implicit Wait.
 - **Not a good practice** to mix both.
-

34. Difference between `close()` and `quit()` in Selenium?

Method	Behavior
<code>close()</code>	Closes only the current window
<code>quit()</code>	Closes all browser windows

35. How do you handle Alerts in Selenium?

```
java
CopyEdit
Alert alert = driver.switchTo().alert();
System.out.println(alert.getText());
alert.accept(); // Clicks OK
```

36. How to handle pop-ups appearing at unknown times?

Use **Explicit Wait with ExpectedConditions.alertIsPresent()**:

```
java
CopyEdit
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
alert.accept();
```

37. How to handle file upload when `<input>` does not have `type="file"`?

- Use **Robot Class**:

```
Robot robot = new Robot();
robot.keyPress(KeyEvent.VK_ENTER);
robot.keyRelease(KeyEvent.VK_ENTER);
```

- Or use **AutoIt/Sikuli** for non-Selenium file uploads.
-

38. How to perform keyboard operations in Selenium?

Use **Actions Class**:

```
java
CopyEdit
Actions actions = new Actions(driver);
actions.sendKeys(Keys.ENTER).perform();
```

39. What does this snippet mean?

```
WebDriver driver = new ChromeDriver();
```

- `WebDriver` → Interface from Selenium.
- `ChromeDriver` → Class that implements `WebDriver`, used for Chrome browser.
- `new ChromeDriver()` → Initializes Chrome browser.

This line means:

1. A `WebDriver` reference (`driver`) is created.
 2. A new instance of `ChromeDriver` (Chrome browser) is assigned to `driver`.
 3. Since `WebDriver` is an interface, this is **Dynamic Polymorphism** (runtime polymorphism).
-

40. Where can Dynamic Polymorphism be observed in Selenium WebDriver?

Dynamic Polymorphism (Method Overriding) occurs when:

```
WebDriver driver = new ChromeDriver();
driver = new FirefoxDriver();
```

- **Method calls are resolved at runtime** (e.g., `driver.get()` will call `ChromeDriver` or `FirefoxDriver` methods accordingly).
 - **WebDriver is an interface**, and `ChromeDriver` and `FirefoxDriver` override its methods.
-

41. Difference between / and // in XPath?

Symbol	Meaning	Example
/	Absolute XPath (starts from root)	<code>/html/body/div[1]/table/tr[2]/td[1]</code>
//	Relative XPath (searches anywhere)	<code>//table[@id='example']/tr/td[1]</code>

Example:

```
// Absolute XPath
driver.findElement(By.xpath("/html/body/div/input")).click();

// Relative XPath
driver.findElement(By.xpath("//input[@id='search']")).click();
```

- **Relative XPath (//) is recommended** since it is more flexible.
-

42. If proper XPath, CSS Selector, and ID are not available, how do you identify an object?

- **Use other attributes** like name, class, text(), contains(), preceding-sibling, parent, child, etc.
- **Use JavaScript Executor** to directly interact with elements.

Example: Using JavaScript Executor

```
JavascriptExecutor js = (JavascriptExecutor) driver;
WebElement element = (WebElement) js.executeScript("return
document.querySelector('input[name=username]')");
```

43. Attributes of CSS Selector?

- **ID Selector:** #id
- **Class Selector:** .classname
- **Tag & Attribute Selector:** tagname[attribute=value]
- **Child Selector:** parent > child
- **Descendant Selector:** ancestor child
- **Sibling Selector:** element1 + element2

Example:

```
input[name='username']
driver.findElement(By.cssSelector("input[name='username']"));
```

44. Which is faster, XPath or CSS?

- **CSS is faster** than XPath because:
 - It uses a browser's native engine.
 - XPath requires traversal from root (especially for complex queries).
 - **XPath is more flexible** (allows parent-to-child and sibling navigation).
-

45. How to get the nth element using XPath and CSS?

XPath:

```
// Select 3rd element
driver.findElement(By.xpath("//div[@class='example'] [3]"));
```

CSS Selector:

```
// Select 3rd element
driver.findElement(By.cssSelector("div.example:nth-of-type(3)"));
```

46. How to find the parent/grandparent using only CSS?

CSS does **not** support traversing backward (no parent selector), but you can use **JavaScript**:

```
JavascriptExecutor js = (JavascriptExecutor) driver;
WebElement parent = (WebElement) js.executeScript("return
document.querySelector('child-selector').parentNode;");
```

47. Will driver.findElements() throw an exception?

- **No.** If no elements are found, it returns an **empty list** (List<WebElement>).
 - Unlike findElement(), which throws NoSuchElementException.
-

48. What is returned by driver.manage()?

Returns "**Options**" interface which helps in:

- **Window handling:** driver.manage().window().maximize();
 - **Cookies handling:** driver.manage().getCookies();
 - **Timeouts handling:** driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
-

49. How to find an element with dynamic ID?

Use **XPath functions**:

```
driver.findElement(By.xpath("//*[contains(text(), 'This element has an ID')]"));
```

or

Use **CSS Wildcard Matchers**:

```
driver.findElement(By.cssSelector("div[class*='dynamic-id']"));
```

50. How to initialize elements in POM? (signupPage.java & driver)

Use PageFactory.initElements():

```
public class SignupPage {
    WebDriver driver;

    @FindBy(id = "username")
    WebElement username;

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

// Test class
SignupPage signup = new SignupPage(driver);
signup.username.sendKeys("testuser");
```

13. Get values from a dropdown and print them in ascending order

```
Select select = new Select(driver.findElement(By.id("dropdown")));
List<WebElement> options = select.getOptions();

// Extract text and sort
List<String> dropdownValues = new ArrayList<>();
for (WebElement option : options) {
    dropdownValues.add(option.getText());
}

Collections.sort(dropdownValues);
System.out.println(dropdownValues);
```

51. Using TreeSet to find unique elements

```
TreeSet<String> uniqueElements = new TreeSet<>();
List<WebElement> elements = driver.findElements(By.className("items"));

for (WebElement element : elements) {
    uniqueElements.add(element.getText());
}

System.out.println(uniqueElements);
```

- **TreeSet automatically sorts elements** and removes duplicates.
-

52. Is TakesScreenshot an Interface or Class?

TakesScreenshot is an **Interface** in Selenium.

Example usage:

```
TakesScreenshot ts = (TakesScreenshot) driver;
File src = ts.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(src, new File("./screenshot.png"));
```

53. Why use Selenium despite many third-party libraries?

- Open-source.
 - Supports multiple browsers.
 - Large community support.
 - Cross-language compatibility (Java, Python, C#).
 - Easily integrates with TestNG, JUnit, Cucumber.
-

54. Why use build() and perform() with Actions class?

- build() **compiles** multiple actions into a single step.
- perform() **executes** the action sequence.

Example:

```
Actions actions = new Actions(driver);
actions.moveToElement(element).click().build().perform();
```

- Without build(), perform() works for single-step actions.
 - For multi-step actions, **always use build() before perform()**.
-

55. Can we use perform() alone in scripting without build()?

Yes, we can use perform() alone if we have only one action in Actions class.

```
Actions action = new Actions(driver);
action.click(element).perform(); // Works fine without build()
```

However, if multiple actions are chained together, use build() before perform().

```
Actions action = new Actions(driver);
action.moveToElement(element).click().build().perform(); // Ensures all
actions are executed together
```

56. Difference between `build()` and `perform()` in Selenium?

Method	Description
<code>build()</code>	Compiles multiple actions into a single step but does not execute them.
<code>perform()</code>	Executes the compiled actions or a single action.

Example:

```
Actions action = new Actions(driver);  
action.moveToElement(element).click().build().perform();
```

- **Without `build()`** → The script might execute one action at a time instead of together.
-

57. Return type of `getWindowHandle()` and `getWindowHandles()`?

Method	Return Type	Description
<code>getWindowHandle()</code>	String	Returns the current window's unique ID .
<code>getWindowHandles()</code>	Set<String>	Returns all window IDs in a Set.

Example:

```
String parentWindow = driver.getWindowHandle();  
Set<String> allWindows = driver.getWindowHandles();
```

58. How to switch back to the parent window in Selenium?

```
String parentWindow = driver.getWindowHandle(); // Get parent window ID  
Set<String> allWindows = driver.getWindowHandles();
```

```
for (String window : allWindows) {  
    if (!window.equals(parentWindow)) {  
        driver.switchTo().window(window); // Switch to new window  
        driver.close(); // Close new window  
    }  
}
```

```
driver.switchTo().window(parentWindow); // Switch back to parent window
```

59. If a button is disabled, how to check using `getAttribute()`?

```
WebElement button = driver.findElement(By.id("submit"));
String isDisabled = button.getAttribute("disabled");
```

```
if (isDisabled != null) {
    System.out.println("Button is disabled");
} else {
    System.out.println("Button is enabled");
}
```

- If the `disabled` attribute is present, the button is disabled.
-

60. Explain method overloading with Selenium and an example.

Method Overloading → Same method name with different parameters.

Example:

```
public void clickElement(WebElement element) {
    element.click();
}

public void clickElement(WebElement element, int waitTime) {
    WebDriverWait wait = new WebDriverWait(driver,
        Duration.ofSeconds(waitTime));
    wait.until(ExpectedConditions.elementToBeClickable(element)).click();
}
```

- Same method (`clickElement`) is used with different parameters.
 - This is **compile-time polymorphism**.
-

61. How do you read Excel in Selenium?

You can use **Apache POI** to read an Excel file.

Code to read an Excel file:

```
FileInputStream file = new FileInputStream(new File("data.xlsx"));
Workbook workbook = new XSSFWorkbook(file);
Sheet sheet = workbook.getSheet("Sheet1");
Row row = sheet.getRow(0);
Cell cell = row.getCell(0);
System.out.println(cell.getStringCellValue());
workbook.close();
```

Counter-questions you may face:

- **How to handle different data types?**

Use `getCellType()` to determine if the cell contains a `String`, `Numeric`, or `Boolean` value.

- **How to write data into Excel?**

Use `setCellValue()` and `FileOutputStream()`.

62. Do you use a property file in your framework? Which Java concept is used?

Yes, we store configuration data (URLs, credentials) in a `.properties` file.

Java concept used: Java Collections (HashMap, Properties Class)

Example:

```
Properties prop = new Properties();
FileInputStream file = new FileInputStream("config.properties");
prop.load(file);
String url = prop.getProperty("app.url");
```

63. How to handle unpredictable pop-ups in Selenium?

Use `WebDriverWait` and `try-catch`:

```
try {
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    Alert alert = wait.until(ExpectedConditions.alertIsPresent());
    alert.accept();
} catch (Exception e) {
    System.out.println("No pop-up appeared.");
}
```

64. How to re-execute failed test cases in a different browser?

1. **Use TestNG RetryAnalyzer**

Create a class:

```
public class RetryLogic implements IRetryAnalyzer {
    private int retryCount = 0;
    private static final int maxRetryCount = 2;

    public boolean retry(ITestResult result) {
        if (retryCount < maxRetryCount) {
            retryCount++;
            return true;
        }
        return false;
    }
}
```

```
}  
}
```

Use in the test:

```
@Test(retryAnalyzer = RetryLogic.class)  
public void testMethod() {  
    // Test script  
}
```

65. How to handle dynamic web elements?

- **Use contains() or starts-with() in XPath:**
 - `driver.findElement(By.xpath("//input[contains(@id, 'dynamic_')]"));`
 - **Use Thread.sleep(3000) (not recommended, but sometimes required).**
-

66. Click the last option in a dropdown.

```
Select select = new Select(driver.findElement(By.id("dropdown")));  
List<WebElement> options = select.getOptions();  
options.get(options.size() - 1).click();
```

67. How to count links on a page?

```
List<WebElement> links = driver.findElements(By.tagName("a"));  
System.out.println("Total links: " + links.size());
```

68. What is Page Factory in POM?

Page Factory helps initialize elements using **lazy loading**.

```
public class LoginPage {  
    WebDriver driver;  
  
    @FindBy(id = "username")  
    WebElement username;  
  
    public LoginPage(WebDriver driver) {  
        this.driver = driver;  
        PageFactory.initElements(driver, this);  
    }  
}
```

69. How to achieve POM if the application has 10 pages?

- Create a **separate class for each page**.
- Store reusable methods inside a **Base Class**.

- Use `PageFactory.initElements(driver, this)` in each page.
-

70. Annotation used in Page Object Model?

- `@FindBy` → Used to locate web elements.
 - `@BeforeMethod` & `@AfterMethod` → Used for setup and teardown.
-

71. How to find broken links in Selenium?

```
List<WebElement> links = driver.findElements(By.tagName("a"));

for (WebElement link : links) {
    String url = link.getAttribute("href");
    HttpURLConnection connection = (HttpURLConnection) new
URL(url).openConnection();
    connection.setRequestMethod("HEAD");
    connection.connect();

    if (connection.getResponseCode() >= 400) {
        System.out.println(url + " is a broken link.");
    }
}
```

72. How to handle frames in Selenium?

```
driver.switchTo().frame("frameName");
driver.switchTo().defaultContent(); // Switch back to the main page
```

73. Different Types of Navigation Commands in Selenium

Selenium provides the `navigate()` method to move between web pages:

```
driver.navigate().to("https://example.com"); // Opens a URL
driver.navigate().back(); // Goes to the previous page
driver.navigate().forward(); // Moves to the next page
driver.navigate().refresh(); // Refreshes the page
```

74. How to Handle Alerts in Selenium?

Selenium provides `switchTo().alert()` to handle JavaScript alerts:

```
Alert alert = driver.switchTo().alert();
alert.accept(); // Clicks OK
alert.dismiss(); // Clicks Cancel
String alertText = alert.getText(); // Gets alert message
alert.sendKeys("Sample Text"); // Sends input to alert (if applicable)
```

75. Difference Between Assert and Verify?

Feature	assert	verify
Purpose	Stops test execution if condition fails	Continues execution even if condition fails
Used In	JUnit, TestNG	Selenium test scripts
Example	<code>Assert.assertEquals(actual, expected);</code>	<code>SoftAssert softAssert = new SoftAssert();</code>

Soft Assert Example:

```
SoftAssert softAssert = new SoftAssert();
softAssert.assertEquals(actual, expected);
softAssert.assertAll(); // This executes all soft assertions
```

76. How to Download a File Using Selenium?

Selenium itself cannot handle file downloads, but we can modify browser preferences:

For Chrome:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("download.default_directory",
"/path/to/download/folder");
```

For Firefox:

```
FirefoxProfile profile = new FirefoxProfile();
profile.setPreference("browser.helperApps.neverAsk.saveToDisk",
"application/pdf");
```

77. How to Manage Data Tables in Selenium?

To handle dynamic tables, iterate through rows and columns:

```
List<WebElement> rows = driver.findElements(By.xpath("//table[@id='data-table']/tbody/tr"));

for (WebElement row : rows) {
    List<WebElement> cols = row.findElements(By.tagName("td"));
    for (WebElement col : cols) {
        System.out.println(col.getText());
    }
}
```

78. How to Automate Localization Testing?

- **Use browser language settings:**
 - `ChromeOptions options = new ChromeOptions();`
 - `options.addArguments("--lang=fr");` // French language
 - **Store translations in a properties file** for assertion checks.
-

79. How to Avoid `NoSuchElementException`?

Without try/catch block:

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementID")
));
```

With try/catch block:

```
try {
    WebElement element = driver.findElement(By.id("elementID"));
} catch (NoSuchElementException e) {
    System.out.println("Element not found!");
}
```

80. How to Handle Dynamic Web Tables?

Find the column dynamically:

```
List<WebElement> cells = driver.findElements(By.xpath("//table[@id='data-
table']/tr[last()]/td"));
```

81. Check If an Element Is Enabled Without `isEnabled()` ?

Use `getAttribute()`:

```
String disabled = element.getAttribute("disabled");
if (disabled == null) {
    System.out.println("Element is enabled.");
} else {
    System.out.println("Element is disabled.");
}
```

82. Why Is CSS Faster Than XPath?

- CSS **directly** interacts with the browser engine, making it faster.
- XPath requires **traversing the DOM**, which slows execution.

83. Why Do Companies Still Use XPath?

- XPath can locate elements based on text, which CSS cannot.
 - XPath supports traversing parent elements, which CSS doesn't support.
-

84. Why Is an Element Not Found Even Though It's Visible?

Possible reasons:

1. **Element is inside an iframe** → Use `switchTo().frame()`.
 2. **DOM updates dynamically** → Use `Explicit Wait`.
-

85. How to Execute Selenium in Headless Mode?

For Chrome:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless");
```

For Firefox:

```
FirefoxOptions options = new FirefoxOptions();
options.setHeadless(true);
```

86. How to Get Text from a Textbox if `getText()` Is Not Working?

Use `getAttribute("value")`:

```
String text = driver.findElement(By.id("textbox")).getAttribute("value");
```

87. How to Solve **Element Not Found** Error Even with Correct Locator?

1. **Use Explicit Wait:**
 2. `WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));`
 3. `wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementID")));`
 4. **Scroll into View:**
 5. `((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", element);`
-

88. First Steps in Writing LoginPage.java in POM?

1. Create locators using @FindBy.
2. Use PageFactory.initElements() to initialize elements.
3. Write reusable methods for login actions.

```
public class LoginPage {  
    WebDriver driver;  
  
    @FindBy(id = "username")  
    WebElement username;  
  
    public LoginPage(WebDriver driver) {  
        this.driver = driver;  
        PageFactory.initElements(driver, this);  
    }  
}
```

89. How to Handle Windows Pop-ups in Selenium?

Use AutoIT or Robot Class.

Using Robot Class:

```
Robot robot = new Robot();  
robot.keyPress(KeyEvent.VK_ENTER);  
robot.keyRelease(KeyEvent.VK_ENTER);
```

90. Different Ways to Handle Hidden Elements?

- **JavaScript Executor:**
((JavascriptExecutor) driver).executeScript("arguments[0].click();", element);
 - **Modify CSS using JavaScript:**
((JavascriptExecutor) driver).executeScript("arguments[0].style.visibility='visible';", element);
-

91. Difference Between click() in WebElement vs Actions Class?

Feature	WebElement.click()	Actions.click()
Execution	Directly clicks on the element	Uses low-level interactions
Use Case	Works for normal clickable elements	Useful for hidden/hover elements

```
Actions action = new Actions(driver);  
action.moveToElement(element).click().perform();
```

92. Can We Change Test Behavior at Runtime?

Yes, using **Data Providers in TestNG**:

```
@DataProvider(name="loginData")
public Object[][] dataProvider() {
    return new Object[][] {{"user1", "pass1"}, {"user2", "pass2"}};
}
```

93. Handling iFrames, Windows, Tables, and Alerts

- **Switch to an iframe:**
 - `driver.switchTo().frame("frameName");`
 - **Handle Windows:**
 - `Set<String> windows = driver.getWindowHandles();`
 - `for (String window : windows) {`
 - `driver.switchTo().window(window);`
 - `}`
 - **Handle Tables:**
 - `List<WebElement> rows = driver.findElements(By.xpath("//table//tr"));`
 - **Handle Alerts:**
 - `driver.switchTo().alert().accept();`
-

94. How to Perform Right Click in Selenium?

```
Actions action = new Actions(driver);
action.contextClick(element).perform();
```

95. How to Scroll Down a Page?

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("window.scrollTo(0,500)");
```

96. What Will `getWindowHandles()` Return?

It returns a **Set of window IDs** (`Set<String>`).

97. How to Automate Windows Applications?

Selenium cannot handle Windows applications directly.
Use **AutoIT**, **Sikuli**, or **Robot Class**.

TestNG

1. What is the Importance of TestNG Framework?

TestNG (**Test Next Generation**) is a widely used testing framework for Java that enhances Selenium automation. Its key benefits include:

- **Annotations** (`@Test`, `@BeforeClass`, `@AfterMethod`, etc.) for better test structure.
 - **Parallel execution** for faster testing.
 - **Grouping and dependencies** to organize test execution.
 - **Data-driven testing** using `@DataProvider`.
 - **Reporting and logging** for better debugging.
-

2. Why Do We Use TestNG in Frameworks?

- **Structured execution** using annotations.
 - **Flexible test configuration** via `testng.xml`.
 - **Parallel execution** improves speed.
 - **Retry mechanism** for failed tests.
 - **Supports assertion methods** like `Assert.assertEquals()`.
-

3. What is the Purpose of `testng.xml`?

`testng.xml` allows us to:

- Run **specific test cases**.
- Execute **tests in parallel**.
- Group test cases (`<groups>`).
- Pass **parameters** to test methods.

Example:

```
<suite name="Suite1">
  <test name="LoginTest">
    <classes>
      <class name="tests.LoginTest"/>
    </classes>
  </test>
</suite>
```

4. What is the Purpose of Listeners?

Listeners in TestNG **listen to test execution events** and help in:

- Capturing **screenshots on failure**.
- Logging test results to reports.

- Generating custom test reports.

Example: Implementing `ITestListener`

```
public class CustomListener implements ITestListener {  
    public void onTestFailure(ITestResult result) {  
        System.out.println("Test Failed: " + result.getName());  
    }  
}
```

Use in `testng.xml`:

```
<listeners>  
    <listener class-name="listeners.CustomListener"/>  
</listeners>
```

5. How to Run the Same Method 100 Times in TestNG?

Use `invocationCount` in `@Test` annotation:

```
@Test(invocationCount = 100)  
public void testMethod() {  
    System.out.println("Executing test 100 times");  
}
```

6. Reporting Tool in TestNG?

- **Default TestNG Reports** (HTML/XML).
- **Extent Reports** for detailed logging, screenshots, and visual charts.
- **Allure Reports** for interactive UI-based reporting.

Example of **Extent Reports**:

```
ExtentReports extent = new ExtentReports("report.html");  
ExtentTest test = extent.startTest("Login Test");  
test.log(LogStatus.PASS, "Login successful");  
extent.endTest(test);  
extent.flush();
```

7. TestNG Annotations and Their Order

Annotation	Execution Order
@BeforeSuite	Runs before all tests in a suite
@BeforeTest	Runs before any test in <test> tag
@BeforeClass	Runs before methods in a class

Annotation	Execution Order
@BeforeMethod	Runs before each test method
@Test	Executes the test case
@AfterMethod	Runs after each test method
@AfterClass	Runs after all methods in a class
@AfterTest	Runs after all test cases
@AfterSuite	Runs after the entire test suite

Example:

```

@BeforeMethod
public void setup() {
    System.out.println("Setup before test");
}

@Test
public void loginTest() {
    System.out.println("Executing login test");
}

@AfterMethod
public void teardown() {
    System.out.println("Cleanup after test");
}

```

8. What is @DataProvider?

It is used for **data-driven testing** to provide multiple sets of data to a test method.

Example:

```

@DataProvider(name = "loginData")
public Object[][] provideData() {
    return new Object[][] {
        {"user1", "pass1"},
        {"user2", "pass2"}
    };
}

@Test(dataProvider = "loginData")
public void loginTest(String username, String password) {
    System.out.println("Logging in with " + username + " and " + password);
}

```

9. Difference Between @Factory and @DataProvider?

Feature	@Factory	@DataProvider
Purpose	Used for creating multiple instances of a test class	Provides multiple data sets to a single test
Use Case	Creating different test objects	Running the same test with different input data
Example	@Factory public Object[] createTests() { return new Object[]{ new TestClass(1), new TestClass(2) }; }	@DataProvider(name="data") public Object[][] provideData() { return new Object[][]{ {"data1"}, {"data2"} }; }

10. Parallel Execution in TestNG

To run tests in **parallel**, use testng.xml:

```
<suite name="ParallelSuite" parallel="tests" thread-count="2">
  <test name="ChromeTest">
    <classes>
      <class name="tests.ChromeTest"/>
    </classes>
  </test>
  <test name="FirefoxTest">
    <classes>
      <class name="tests.FirefoxTest"/>
    </classes>
  </test>
</suite>
```

11. Running Only Failed Test Cases in TestNG

- First execution generates a testng-failed.xml.
- Re-run the failed tests using:

```
java -cp bin:testng.jar org.testng.TestNG testng-failed.xml
```

12. Taking Screenshot for Failed Test Cases

Using ITestListener:

```
public class ScreenshotListener implements ITestListener {
    public void onTestFailure(ITestResult result) {
        File src = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(src, new File("screenshot.png"));
    }
}
```

```
}
```

13. How to Run the Same Tests 10 Times?

```
@Test(invocationCount = 10)
public void testMethod() {
    System.out.println("Executing test 10 times");
}
```

14. Types of Listeners in TestNG

Listener	Purpose
ITestListener	Captures test start, success, failure events
ISuiteListener	Executes before and after a test suite
IReporter	Generates custom reports
IAnnotationTransformer	Modifies test execution at runtime

15. Difference Between @BeforeSuite and @AfterSuite

- **@BeforeSuite** → Runs **once** before all tests in a suite.
- **@AfterSuite** → Runs **once** after all tests in a suite.

Example:

```
@BeforeSuite
public void setupSuite() {
    System.out.println("Setup before suite");
}

@AfterSuite
public void cleanupSuite() {
    System.out.println("Cleanup after suite");
}
```

16. How to Add/Remove Test Cases in testng.xml?

To **add** a test:

```
<test name="Login Test">
    <classes>
        <class name="tests.LoginTest"/>
    </classes>
</test>
```

To **remove**, simply delete or comment out the <test> section.

17. How Many Suites Can Be There in TestNG?

In TestNG, you can have **multiple test suites** inside a single XML file or separate XML files. There is no hard limit on the number of suites.

If you want to run all suites, you can create a master `testng.xml` and include all other suites:

```
<suite name="MasterSuite">
  <suite-files>
    <suite-file path="suite1.xml"/>
    <suite-file path="suite2.xml"/>
  </suite-files>
</suite>
```

Run it using:

```
java -cp "bin:testng.jar" org.testng.TestNG master-suite.xml
```

18. Syntax to Perform Parallel Testing in TestNG

To perform **parallel execution**, use `parallel="methods"`, `parallel="tests"`, or `parallel="classes"` in `<suite>` tag.

Example of parallel execution (methods):

```
<suite name="ParallelSuite" parallel="methods" thread-count="3">
  <test name="Test1">
    <classes>
      <class name="tests.LoginTest"/>
      <class name="tests.HomePageTest"/>
    </classes>
  </test>
</suite>
```

Options for `parallel=""`:

Value	Description
methods	Runs test methods in parallel
classes	Runs test classes in parallel
tests	Runs test cases in parallel
suites	Runs test suites in parallel

19. Can We Have Multiple Suites in One XML?

Yes, you can define multiple `<suite>` tags in one XML file.

Example:

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite1">
  <test name="LoginTest">
    <classes>
      <class name="tests.LoginTest"/>
    </classes>
  </test>
</suite>

<suite name="Suite2">
  <test name="SearchTest">
    <classes>
      <class name="tests.SearchTest"/>
    </classes>
  </test>
</suite>
```

To run all suites, execute the XML file with TestNG.

20. What is invocationCount in TestNG?

invocationCount is used to run the same test **multiple times**.

Example: Run a test 5 times

```
@Test(invocationCount = 5)
public void loginTest() {
    System.out.println("Running login test multiple times");
}
```

Cucumber, and BDD

1. What is Cucumber, and Why is it Used in BDD?

Cucumber is an **open-source tool** used for **Behavior-Driven Development (BDD)** that allows writing test cases in **plain English** using the **Gherkin language**. It enables collaboration between **developers, testers, and business stakeholders**.

✓ Why use Cucumber in BDD?

- Improves **test readability** using natural language.
- Bridges the gap between **business analysts, developers, and testers**.
- Allows **test automation** with Selenium, Appium, and APIs.
- Supports **data-driven** and **parallel execution**.

★ Example:

Feature: Login Functionality

Scenario: Successful Login

Given User is on the login page

When User enters valid credentials

Then User should be logged in successfully

2. What are Gherkin Keywords?

Gherkin is a simple language that defines test cases in **Cucumber**.

✓ Common Gherkin Keywords:

Keyword	Purpose
Feature:	Describes the high-level feature
Scenario:	Defines a test case
Given	Sets up the initial context (Preconditions)
When	Specifies an action (User interaction)
Then	Verifies the expected outcome
And / But	Adds additional steps
Background:	Reusable precondition for all scenarios
Scenario Outline:	Runs the same test multiple times with different inputs
Examples:	Data table for Scenario Outline

3. How Do You Structure Feature Files in Cucumber?

Best Practices for Feature Files:

1. **One feature per file** (e.g., login.feature).
2. **Follow Given-When-Then format.**
3. **Avoid UI-specific details.**
4. **Use meaningful step definitions.**

★ Example Feature File Structure:

Feature: Search Functionality

Scenario: Search for a product

Given User is on Home Page

When User searches for "Laptop"

Then Search results should display "Laptop"

4. Difference Between Scenario and Scenario Outline?

Feature	Scenario	Scenario Outline
Purpose	Defines a single test case	Runs the same test multiple times with different inputs
Data Handling	Hardcoded values	Uses <placeholders> with Examples
Example	✓ Scenario: Login with valid credentials	✓ Scenario Outline: Login with multiple credentials

★ Example of Scenario Outline (Data-Driven Test):

```
Scenario Outline: Login with multiple credentials
  Given User is on Login Page
  When User enters "<username>" and "<password>"
  Then User is logged in successfully
```

Examples:

```
| username | password |
| user123  | pass123  |
| user456  | pass456  |
```

🔗 Use Scenario Outline when testing multiple data sets!

5. How Are Step Definitions Mapped to Feature File Steps?

- Cucumber **matches** steps in feature files to **Java methods** using **regular expressions**.
- Step definitions must be in **glue code** (inside `stepDefinitions` package).

★ Example:

■ Feature File (`login.feature`)

```
Scenario: Login with valid credentials
  Given User is on Login Page
  When User enters "admin" and "admin123"
  Then User is logged in
```

■ Step Definition (`LoginSteps.java`)

```
public class LoginSteps {

    @Given("User is on Login Page")
    public void userIsOnLoginPage() {
        driver.get("https://example.com/login");
    }

    @When("User enters {string} and {string}")
    public void userEntersCredentials(String username, String password) {
        driver.findElement(By.id("username")).sendKeys(username);
        driver.findElement(By.id("password")).sendKeys(password);
        driver.findElement(By.id("login")).click();
    }
}
```

```
@Then("User should be logged in")
public void userShouldBeLoggedIn() {
    Assert.assertTrue(driver.findElement(By.id("logout")).isDisplayed());
}
}
```

6. How Do You Parameterize Steps in Cucumber?

✓ Using Placeholders ({string}, {int}):

```
Scenario: Login with multiple users
    When User logs in with "admin" and "admin123"
@When("User logs in with {string} and {string}")
public void userLogsIn(String username, String password) {
    driver.findElement(By.id("username")).sendKeys(username);
    driver.findElement(By.id("password")).sendKeys(password);
}
```

7. What Happens If Multiple Step Definitions Match the Same Step?

- **Cucumber throws an error:**
"Ambiguous step definitions for..."
 - **Solution:**
 - Ensure **unique regex patterns** for step definitions.
 - Use **precise regex matching**.
-

8. What Are Tags in Cucumber and How Are They Used?

Tags help in **categorizing and running specific tests**.

✓ Example:

```
@smoke
Scenario: Verify Login
    Given User is on Login Page
    When User enters "valid" credentials
    Then User should be logged in successfully
```

✓ Run tests with tags in Runner class:

```
@CucumberOptions(tags = "@smoke")
```

9. What Are Hooks (@Before, @After) in Cucumber?

Hooks allow executing **setup and teardown** logic.

✓ Example (Hooks.java)

```
@Before
public void setUp() {
    driver = new ChromeDriver();
    driver.manage().window().maximize();
}

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

10. How Do You Handle Test Data in Cucumber Scenarios?

- **Using Scenario Outline with Examples**
- **Using DataTable for complex data**

★ Example: Using DataTable in Step Definition

```
Scenario: User registration
    When User enters the following details
        | Name | Email | Age |
        | John | john@email.com | 25 |
    @When("User enters the following details")
    public void enterUserDetails(DataTable dataTable) {
        List<Map<String, String>> data = dataTable.asMaps(String.class,
String.class);
        String name = data.get(0).get("Name");
        String email = data.get(0).get("Email");
    }
```

11. How Do You Integrate Cucumber with Selenium WebDriver?

- **Define Cucumber Feature Files**
 - **Create Step Definitions using WebDriver**
 - **Configure Runner Class with @CucumberOptions**
-

12. Can You Run Cucumber Tests in Parallel?

✓ Enable Parallel Execution in testng.xml

```
<suite name="ParallelSuite" parallel="methods" thread-count="3">
    <test name="CucumberTest">
        <classes>
            <class name="runner.TestRunner"/>
        </classes>
    </test>
</suite>
```

✓ Use @DataProvider(parallel = true)

```
@DataProvider(parallel = true)
public Object[][] scenarios() {
    return super.scenarios();
}
```

13. What Is the Difference Between Scenario and Scenario Outline?

Feature	Scenario	Scenario Outline
Purpose	Single test case	Runs same test with multiple inputs
Data Handling	Hardcoded values	Uses <placeholders> with Examples: table
Example	✓ Scenario: Login with valid credentials	✓ Scenario Outline: Login with multiple credentials

★ Example of Scenario Outline (Data-Driven Test)

Scenario Outline: Login with multiple users
 Given User is on Login Page
 When User logs in with "<username>" and "<password>"
 Then User is redirected to dashboard

Examples:

username	password
user1	pass1
user2	pass2

14. How Do You Handle Dynamic Locators in Cucumber?

- Use **contains()**, **starts-with()**, **ends-with()** in XPath.
- Use **CSS selectors with partial matches**.

★ Example:

```
@When("User clicks on product containing {string}")
public void userClicksOnProduct(String productName) {
    driver.findElement(By.xpath("//a[contains(text(), '" + productName +
    "')]").click();
}
```

15. How to Handle Windows Popup in Selenium?

Selenium **cannot handle OS-based popups** directly. Use **Robot class**, **AutoIT**, or **Sikuli**.

✓ Using Robot Class (Java)

```
Robot robot = new Robot();
robot.keyPress(KeyEvent.VK_ENTER);
robot.keyRelease(KeyEvent.VK_ENTER);
```

✓ Using AutoIT (For file upload popups)

1. **Write AutoIT Script (upload.au3)**
 2. `ControlFocus("Open","", "Edit1")`
 3. `ControlSetText("Open","", "Edit1", "C:\file.pdf")`
 4. `ControlClick("Open","", "Button1")`
 5. **Execute in Selenium**
 6. `Runtime.getRuntime().exec("C:\\path\\to\\upload.exe");`
-

16. How Do You Run Only Failed Test Cases in Selenium?

✓ Using TestNG `testng-failed.xml`

1. **After execution, TestNG generates `testng-failed.xml`**
2. **Run the failed tests using:**
3. `java -cp bin:testng.jar org.testng.TestNG test-output/testng-failed.xml`

✓ Using `retryAnalyzer` in TestNG

```
public class RetryAnalyzer implements IRetryAnalyzer {
    private int count = 0;
    private static final int MAX_RETRIES = 2;

    public boolean retry(ITestResult result) {
        if (count < MAX_RETRIES) {
            count++;
            return true;
        }
        return false;
    }
}
```

Apply in TestNG:

```
@Test(retryAnalyzer = RetryAnalyzer.class)
public void testCase() {
    Assert.assertEquals(false, true);
}
```

17. How to Handle Alerts in Selenium?

Use the **Alert interface** to switch and handle alerts.

✓ Handle JavaScript Alert (OK Button Only)

```
Alert alert = driver.switchTo().alert();
alert.accept(); // Clicks OK
```

✓ Handle Confirmation Alert (OK/Cancel Button)

```
Alert alert = driver.switchTo().alert();
alert.dismiss(); // Clicks Cancel
```

✓ **Handle Prompt Alert (Enter Text in Alert Box)**

```
Alert alert = driver.switchTo().alert();
alert.sendKeys("Hello");
alert.accept();
```

18. How to Scroll Down a Page in Selenium?

✓ **Using JavaScriptExecutor**

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("window.scrollTo(0,1000)"); // Scroll down 1000px
```

✓ **Scroll to a Specific Element**

```
WebElement element = driver.findElement(By.id("footer"));
js.executeScript("arguments[0].scrollIntoView();", element);
```

✓ **Using Actions Class**

```
Actions actions = new Actions(driver);
actions.sendKeys(Keys.PAGE_DOWN).perform();
```

19. What Is PageFactory in Selenium POM?

PageFactory is a class in Selenium that initializes WebElements at runtime.

✓ **Steps to Implement POM Using PageFactory**

■ **LoginPage.java (Page Object Class)**

```
public class LoginPage {
    WebDriver driver;

    @FindBy(id="username") WebElement username;
    @FindBy(id="password") WebElement password;
    @FindBy(id="login") WebElement loginButton;

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

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


■ Test Case (LoginTest.java)

```
LoginPage login = new LoginPage(driver);  
login.login("admin", "admin123");
```

20. How Do You Capture Screenshots in Selenium?

✓ Capture Full Page Screenshot

```
File src = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);  
FileUtils.copyFile(src, new File("screenshot.png"));
```

✓ Capture Element-Specific Screenshot

```
WebElement element = driver.findElement(By.id("logo"));  
File src = element.getScreenshotAs(OutputType.FILE);  
FileUtils.copyFile(src, new File("element.png"));
```

21. How to Handle Multiple Windows in Selenium?

✓ Switch to a New Window

```
String parentWindow = driver.getWindowHandle();  
for (String windowHandle : driver.getWindowHandles()) {  
    if (!windowHandle.equals(parentWindow)) {  
        driver.switchTo().window(windowHandle);  
    }  
}
```

✓ Switch Back to Parent Window

```
driver.switchTo().window(parentWindow);
```

22. TestNG – What Is the Difference Between @BeforeMethod and @BeforeClass?

Annotation	When It Runs?	Scope
@BeforeMethod	Before each test method	Runs before every @Test
@BeforeClass	Once before all test methods in a class	Runs once before all @Test methods

★ Example:

```
@BeforeMethod  
public void beforeMethod() {  
    System.out.println("Runs before each test");  
}
```

```
@BeforeClass
```

```
public void beforeClass() {  
    System.out.println("Runs once before all tests in the class");  
}
```

23. How Do You Run Cucumber Tests in Parallel?

✓ Using `testng.xml` for Parallel Execution

```
<suite name="ParallelSuite" parallel="methods" thread-count="3">  
    <test name="CucumberTest">  
        <classes>  
            <class name="runner.TestRunner"/>  
        </classes>  
    </test>  
</suite>
```

✓ Enable Parallel Execution in Cucumber

```
@DataProvider(parallel = true)  
public Object[][] scenarios() {  
    return super.scenarios();  
}
```

24. How to Handle Web Tables in Selenium?

✓ Get All Rows of a Table

```
List<WebElement> rows = driver.findElements(By.xpath("//table/tbody/tr"));  
System.out.println("Total rows: " + rows.size());
```

✓ Read Specific Column Data

```
List<WebElement> columns =  
driver.findElements(By.xpath("//table/tbody/tr/td[2]"));  
for (WebElement column : columns) {  
    System.out.println(column.getText());  
}
```

✓ Click on a Specific Cell in a Dynamic Table

```
driver.findElement(By.xpath("//table/tbody/tr[3]/td[2]")).click();
```

Maven

Here are the detailed answers to your **Maven, CI/CD, and Jenkins** questions:

Maven

1. Lifecycle of Maven

Maven has **three** major lifecycles:

- 1 **Clean Lifecycle**: Cleans the project
- 2 **Default (Build) Lifecycle**: Compiles, tests, and packages the project
- 3 **Site Lifecycle**: Generates project documentation

✓ **Phases of Default Lifecycle:**

Phase	Description
validate	Checks if project is correct
compile	Compiles the source code
test	Runs unit tests
package	Creates a JAR/WAR file
verify	Runs additional checks
install	Adds the package to the local repository
deploy	Copies the package to a remote repository

★ **Command to Execute Maven Lifecycle:**

```
mvn clean compile test package install deploy
```

2. Use of Maven Surefire Plugin

The **Surefire Plugin** is used for running **unit and integration tests**.

- ✓ **Where?** Inside the `pom.xml` under `<build>`
- ✓ **Why?** Runs tests, generates reports, and re-runs failed tests.

★ **Configuration in `pom.xml`**

```
<build>
```

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M5</version>
    <configuration>
      <testFailureIgnore>true</testFailureIgnore>
    </configuration>
  </plugin>
</plugins>
</build>
```

✓ Run Test Cases Using Surefire Plugin

```
mvn test
```

3. What Is the Use of pom.xml?

pom.xml (Project Object Model) manages the project's dependencies, build process, and configurations.

✓ Key Elements in pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>myproject</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.5.0</version>
    </dependency>
  </dependencies>
</project>
```

4. How to Handle Dependencies at Runtime in Maven?

You can manage dependencies dynamically by using:

✓ Using Profiles in pom.xml

```
<profiles>
  <profile>
    <id>qa</id>
    <dependencies>
      <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>4.5.0</version>
      </dependency>
    </dependencies>
  </profile>
```

```
</profiles>
```

★ Run tests for a specific profile

```
mvn test -Pqa
```

✓ Using System Properties

```
mvn test -Denv=qa
```

5. What If Dependencies Are Deleted from `pom.xml`?

🔗 Solution: Use `mvn install` to Restore Dependencies

If dependencies are deleted, you won't be able to run tests **until you restore them**. If you had previously executed:

```
mvn install
```

Then, Maven has cached dependencies in `.m2` directory, and they can be reused. Otherwise, **you need to add them back** in `pom.xml`.

6. How to Pass Different Data for Different Environments?

✓ Using Profiles in `pom.xml`

```
<profiles>
  <profile>
    <id>qa</id>
    <properties>
      <env.url>https://qa.example.com</env.url>
    </properties>
  </profile>
  <profile>
    <id>prod</id>
    <properties>
      <env.url>https://prod.example.com</env.url>
    </properties>
  </profile>
</profiles>
```

★ Run for QA Environment

```
mvn test -Pqa
```

7. Basic Maven Commands

Command	Purpose
<code>mvn clean</code>	Cleans the project (removes target directory)
<code>mvn compile</code>	Compiles the source code
<code>mvn test</code>	Runs the test cases
<code>mvn package</code>	Creates a JAR/WAR file
<code>mvn install</code>	Installs the package in local repository
<code>mvn deploy</code>	Deploys package to remote repository
<code>mvn dependency:tree</code>	Displays dependency hierarchy

CI/CD and Jenkins

8. What Is Jenkins?


Jenkins is an open-source CI/CD tool that helps in automating **builds, tests, and deployments**.

✓ Why Use Jenkins?

- Automatically triggers test execution on **code commits**.
 - Integrates with **Git, Maven, Selenium, and TestNG**.
 - Enables **parallel execution** and scheduling of test runs.
-

9. How to Configure a Jenkins Job?

1  **Install Jenkins and Open UI** (<http://localhost:8080>)

2  **Create a New Job** → Select "Freestyle Project"

3  **Configure Git Repository**

- Add repository URL (e.g., <https://github.com/user/repo.git>)

4  **Configure Build Step**

- Add command:

```
mvn clean test
```

5  **Schedule Automated Builds**

- Use CRON syntax:

```
H 12 * * 1-5 # Run at 12 PM (Monday-Friday)
```

10. How Do You Schedule Deployments in Jenkins?

- 1 ☐ Use **Post-Build Actions** → Add **"Deploy to Server"**
- 2 ☐ Use **Jenkins Pipeline** to Deploy on Build Trigger

★ Example of a Jenkinsfile for Deployment

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Deploy') {
      steps {
        deploy adapters: [tomcat8(credentialsId: 'tomcat-creds',
path: '', url: 'http://server:8080')],
        contextPath: '/', war: '**/target/*.war'
      }
    }
  }
}
```

11. What Are Two Components Jenkins Is Integrated With?

- 1 ☐ **Version Control System (Git, GitHub, GitLab, Bitbucket)**
 - 2 ☐ **Build Automation Tools (Maven, Gradle, Ant)**
-

12. What Is Version Control?

Version control is **a system that tracks code changes over time.**

✓ Popular Tools:

- **Git** (Most widely used)
 - **SVN** (Older version control)
 - **Mercurial**
-

13. Git Commands You Have Used

Command	Purpose
<code>git init</code>	Initialize a Git repo

Command	Purpose
<code>git clone <repo-url></code>	Clone a repository
<code>git status</code>	Check file changes
<code>git add .</code>	Stage all changes
<code>git commit -m "message"</code>	Commit changes
<code>git push origin main</code>	Push code to remote
<code>git pull</code>	Fetch latest changes

14. Difference Between groupId and artifactId in Maven

Concept	groupId	artifactId
Definition	Defines project group	Defines the specific module
Example	<code>org.seleniumhq</code>	<code>selenium-java</code>

★ Example in pom.xml

```
<groupId>com.mycompany</groupId>
<artifactId>myproject</artifactId>
```

15. How and When Is Jenkins Used in Your Automation?

- Runs test scripts on every code commit.
- Triggers nightly test execution.
- Generates reports and sends email alerts for failures.

API Testing (REST, Postman, Authorization, OAuth, Status Codes)

1. Difference Between REST and SOAPUI

Feature	REST (Representational State Transfer)	SOAP (Simple Object Access Protocol)
Protocol	Uses HTTP	Uses XML-based protocol
Data Format	JSON, XML, Plain Text	Only XML
Performance	Lightweight and fast	Slower due to XML processing

Feature	REST (Representational State Transfer)	SOAP (Simple Object Access Protocol)
Security	Uses OAuth, JWT, or basic auth	WS-Security, SSL
Usage	Preferred for Web APIs	Used in enterprise applications

★ Example of REST API Call

GET https://api.example.com/users

★ Example of SOAP Request (XML format)

```
<soapenv:Envelope>
  <soapenv:Body>
    <GetUserDetails>
      <UserId>123</UserId>
    </GetUserDetails>
  </soapenv:Body>
</soapenv:Envelope>
```

2. Methods in REST API

Method	Purpose
GET	Retrieve data
POST	Create a new resource
PUT	Update a resource (Replaces entire resource)
PATCH	Partial update of a resource
DELETE	Remove a resource

3. Difference Between PUT and PATCH

Method	Purpose	Example
PUT	Updates entire resource	If a user has {name, email, age} and PUT updates only email, then other fields get removed.
PATCH	Updates only specific fields	If a user has {name, email, age}, and PATCH updates email, other fields remain unchanged.

★ PUT Example

```
PUT /users/123
{
  "name": "John",
  "email": "john@example.com",
  "age": 30
}
```

★ PATCH Example

```
PATCH /users/123
{
  "email": "john@example.com"
}
```

4. How to Integrate Postman into a Project?

✓ Approaches:

1 Use Newman (Postman CLI) to Run Tests

```
newman run collection.json -e environment.json
```

2 Integrate Postman Tests with CI/CD (Jenkins)

- Add a **Post-Build Step**
- Run:

```
newman run collection.json -r junit --reporter-junit-export results.xml
```

- Publish results in **Jenkins Reports**
-

5. How to Handle Dynamic Payloads in API Testing?

✓ Approach: Use Variables and Data-Driven Testing

★ Dynamic Payload with Java and RestAssured

```
JSONObject request = new JSONObject();
request.put("name", "User" + System.currentTimeMillis());
request.put("email", "user" + Math.random() + "@example.com");
```

★ Using Postman Pre-Request Script for Dynamic Data

```
pm.globals.set("dynamicEmail", "user" + Math.random() + "@example.com");
```

6. How to Capture API Response and Pass It to Another Request?

✓ Postman Approach:

1 Capture Response Value

```
var jsonData = pm.response.json();
pm.globals.set("userId", jsonData.id);
```

2 Use in Another API Call

```
GET https://api.example.com/users/{{userId}}
```

✓ RestAssured Approach:

```
Response response = given().when().get("/users/123");
String userId = response.jsonPath().getString("id");

given().pathParam("id", userId).when().get("/users/{id}");
```

7. Challenges in API Testing

✓ Common Challenges:

- **Handling Dynamic Responses** (IDs, timestamps)
 - **Authentication Issues** (OAuth, JWT expiration)
 - **Testing Asynchronous APIs** (WebSockets, polling)
 - **Rate Limits** (API throttling, retries)
 - **Handling Different Environments** (QA, Staging, Prod)
-

8. Difference Between Authentication and Authorization

Concept	Authentication	Authorization
Definition	Confirms who you are	Defines what you can access
Process	Uses credentials (username/password, OAuth)	Uses roles & permissions
Example	Logging in with username & password	Admins can delete users, but normal users cannot

★ Example of Authentication Token (JWT)

```
{
  "access_token": "eyJhbGciOiJIUzI1...",
  "expires_in": 3600
}
```

9. Common API Status Codes

Status Code	Meaning
200 OK	Success
201 Created	Resource created
400 Bad Request	Invalid request
401 Unauthorized	Authentication failed
403 Forbidden	Not allowed
404 Not Found	Resource does not exist

Status Code	Meaning
500 Internal Server Error	Server failure

10. Difference Between OAuth 1.0 and OAuth 2.0

Feature	OAuth 1.0	OAuth 2.0
Security	Complex signing process	Uses Bearer Tokens
Performance	Slower	Faster
Use Case	Legacy APIs	Modern Web & Mobile Apps

★ OAuth 2.0 Authorization Example in Java

```
given()
    .auth()
    .oauth2("your_access_token")
    .when()
    .get("/api/resource")
    .then()
    .statusCode(200);
```

11. How to Pass API Response from One Call to Another?

✓ Using Postman

1. Extract Token from Response

```
var jsonData = pm.response.json();
pm.globals.set("access_token", jsonData.access_token);
```

2. Use in Next Request

```
GET https://api.example.com/data
Authorization: Bearer {{access_token}}
```

✓ Using RestAssured

```
Response response = given().when().post("/login");
String token = response.jsonPath().getString("access_token");

given().header("Authorization", "Bearer " + token).when().get("/users");
```

Java

Java Core Concepts – Interview Questions & Answers

1. Why is String Immutable in Java?

✓ Reasons:

- 1 ☐ **Security** – Used in class loading, encryption, etc.
- 2 ☐ **Thread-Safety** – No need for synchronization.
- 3 ☐ **Caching & Performance** – String Pool reuses objects.
- 4 ☐ **HashCode Consistency** – Prevents changes in HashMap keys.

★ Example:

```
String s1 = "Hello";
String s2 = s1.concat(" World"); // Creates a new object, doesn't modify s1
System.out.println(s1); // Output: Hello
System.out.println(s2); // Output: Hello World
```

2. What is static in Java?

✓ **static** is a keyword used for memory management.

Where Used?	Purpose
Static Variable	Shared among all objects.
Static Method	Can be called without creating an object.
Static Block	Runs before the <code>main()</code> method.

★ Example:

```
class Example {
    static int count = 0;

    static void display() {
        System.out.println("Static Method Called");
    }

    static { // Static Block
        System.out.println("Static Block Executed");
    }
}
```

3. What is `final` in Java?

✓ Used for restriction (No Modification Allowed)

Where Used?	Purpose
final Variable	Value cannot be changed.
final Method	Cannot be overridden.
final Class	Cannot be inherited.

★ **Example:**

```
final class Parent { }  
class Child extends Parent { } // ✗ Error: Cannot inherit from final class
```

4. What is `this` in Java?

✓ Refers to the current object instance.

★ **Use Cases:**

1 ☐ **Distinguishing Instance & Local Variables:**

```
class Example {  
    int x;  
    Example(int x) {  
        this.x = x; // `this` refers to the instance variable  
    }  
}
```

2 ☐ **Calling Constructor Inside Another Constructor (`this()`)**

```
class Example {  
    Example() {  
        this(10); // Calls another constructor  
    }  
    Example(int x) {  
        System.out.println("Value: " + x);  
    }  
}
```

5. What is `finally` and where do we use it?

✓ Used in Exception Handling. Executes Code Regardless of Exception Occurrence.

★ **Example:**

```
try {  
    int x = 10 / 0;  
} catch (ArithmeticException e) {
```

```
        System.out.println("Exception caught");
    } finally {
        System.out.println("Finally block executed");
    }
}
```

★ Output:

```
Exception caught
Finally block executed
```

6. What is Autoboxing and Unboxing?

✓ **Autoboxing:** Converting Primitive → Wrapper Class

✓ **Unboxing:** Converting Wrapper Class → Primitive

★ Example:

```
// Autoboxing
Integer num = 10; // int → Integer

// Unboxing
int x = num; // Integer → int
```

7. What is Serialization and Deserialization?

✓ **Serialization** → Convert Object → Byte Stream

✓ **Deserialization** → Convert Byte Stream → Object

★ Example Using ObjectOutputStream & ObjectInputStream

```
import java.io.*;

class Student implements Serializable {
    int id;
    String name;
    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

public class SerializationExample {
    public static void main(String[] args) throws Exception {
        // Serialization
        ObjectOutputStream out = new ObjectOutputStream(new
        FileOutputStream("student.ser"));
        out.writeObject(new Student(1, "John"));
        out.close();

        // Deserialization
        ObjectInputStream in = new ObjectInputStream(new
        FileInputStream("student.ser"));
        Student s = (Student) in.readObject();
    }
}
```

```
        in.close();
        System.out.println(s.id + " " + s.name); // Output: 1 John
    }
}
```

8. What is an Abstract Modifier?

✓ Used to define abstract classes and methods.

- **Abstract Class:** Cannot be instantiated.
- **Abstract Method:** Must be implemented in subclass.

★ Example:

```
abstract class Animal {
    abstract void makeSound(); // No implementation
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Bark");
    }
}
```

9. What is Call by Value and Call by Reference?

✓ **Call by Value:** Copy of Variable is Passed

✓ **Call by Reference:** Reference (Memory Address) is Passed

★ Example (Call by Value in Java - Primitive Types)

```
class Example {
    void change(int x) {
        x = 50;
    }
    public static void main(String[] args) {
        Example obj = new Example();
        int num = 10;
        obj.change(num);
        System.out.println(num); // Output: 10 (Original value unchanged)
    }
}
```

★ Example (Call by Reference - Objects are Passed by Reference)

```
class Example {
    int num = 10;
    void change(Example obj) {
        obj.num = 50;
    }
    public static void main(String[] args) {
        Example obj = new Example();
    }
}
```



```

        obj.change(obj);
        System.out.println(obj.num); // Output: 50 (Value changed)
    }
}

```

10. Primitives vs Non-Primitives in Java

Primitive Data Types	Non-Primitive Data Types
byte, short, int, long, float, double, char, boolean	String, Array, Class, Interface
Stored in Stack Memory	Stored in Heap Memory
Faster	Slower
No additional methods available	Has built-in methods

★ Example:

```

int x = 10; // Primitive
String str = "Hello"; // Non-Primitive

```

11. What is Method Overloading?

✓ **Method Overloading** → Defining multiple methods with the **same name but different parameters** (type, number, or order).

★ Example:

```

class Example {
    void show(int a) {
        System.out.println("Integer: " + a);
    }

    void show(double a) {
        System.out.println("Double: " + a);
    }

    void show(int a, int b) {
        System.out.println("Sum: " + (a + b));
    }

    public static void main(String[] args) {
        Example obj = new Example();
        obj.show(10);
        obj.show(5.5);
        obj.show(10, 20);
    }
}

```

★ Output:

```

Integer: 10
Double: 5.5
Sum: 30

```

12. Why Override `hashCode()` When Overriding `equals()`?

✓ **Reason:** In collections (like `HashMap`, `HashSet`), objects are stored based on **hashCode**. If `equals()` is overridden but `hashCode()` is not, objects that are **logically equal** may not be **treated as equal** in collections.

★ Example:

```
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Student student = (Student) obj;
        return id == student.id && name.equals(student.name);
    }

    @Override
    public int hashCode() { // Ensures objects with same data get the same
hash        return id;
    }
}
```

13. Checked vs. Unchecked Exceptions

✓ **Checked Exceptions** → Checked at compile time (e.g., `IOException`, `SQLException`).

✓ **Unchecked Exceptions** → Checked at runtime (e.g., `NullPointerException`, `ArrayIndexOutOfBoundsException`).

★ Example:

```
// Checked Exception
try {
    File file = new File("test.txt");
    FileReader fr = new FileReader(file);
} catch (IOException e) {
    e.printStackTrace();
}

// Unchecked Exception
int a = 5 / 0; // ArithmeticException
```

14. Difference between `final`, `finally`, `finalize`

Keyword	Purpose
<code>final</code>	Used for constants, prevents method overriding, prevents class inheritance.
<code>finally</code>	Block used in exception handling, always executes.
<code>finalize()</code>	Method used for garbage collection before object is destroyed.

★ Example:

```
final class Test { } // Cannot be inherited

try {
    int x = 10 / 0;
} catch (Exception e) {
    System.out.println("Exception caught");
} finally {
    System.out.println("Finally block executed");
}
```

15. Abstract Class vs. Interface

Feature	Abstract Class	Interface
Methods	Can have abstract & concrete methods	Only abstract methods (until Java 8)
Variables	Can have instance variables	Only <code>static final</code> constants
Inheritance	Supports single inheritance	Supports multiple inheritance

★ Example:

```
abstract class Animal {
    abstract void sound();
}

interface Pet {
    void play();
}
```

16. `StringBuilder` vs. `StringBuffer`

✓ Both are mutable, but `StringBuffer` is synchronized (thread-safe) while `StringBuilder` is not.

★ Example:

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World"); // Faster
```

```
StringBuffer sbf = new StringBuffer("Hello");
sbf.append(" World"); // Thread-Safe
```

17. Array vs. ArrayList

Feature	Array	ArrayList
Size	Fixed	Dynamic
Performance	Fast	Slower (due to resizing)
Data Type	Can store primitives	Stores objects only

★ Example:

```
int[] arr = new int[5]; // Array

ArrayList<Integer> list = new ArrayList<>(); // ArrayList
```

18. Can We Create an Object of an Abstract Class?

✗ No, but we can create an instance using an **anonymous inner class**.

```
abstract class Example {
    abstract void display();
}

public class Test {
    public static void main(String[] args) {
        Example obj = new Example() { // Anonymous class
            void display() {
                System.out.println("Hello");
            }
        };
        obj.display();
    }
}
```

9. Can the Constructor Be Overloaded?

✓ Yes, by changing parameters.

★ Example:

```
class Example {
    Example() {
```

```
        System.out.println("Default Constructor");
    }

    Example(int x) {
        System.out.println("Parameterized Constructor: " + x);
    }
}
```

20. Can We Override `main()`?

✗ No, because `main()` is static, and static methods cannot be overridden.

★ Example:

```
class Parent {
    static void main(String[] args) { } // Not overridden
}
```

21. Can We Overload `main()`?

✓ Yes, but JVM only calls `main(String[] args)`.

★ Example:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Main method");
    }

    public static void main(int a) {
        System.out.println("Overloaded main method: " + a);
    }
}
```

22. Can We Overload Static Methods?

✓ Yes, but method signature must be different.

```
class Example {
    static void show() { System.out.println("Static Method 1"); }
    static void show(int a) { System.out.println("Static Method 2: " + a); }
}
```

23. HashMap vs. HashSet

Feature	HashMap	HashSet
Stores	Key-Value Pairs	Only Unique Elements
Allows Duplicates?	Keys ✗, Values ✓	✗
Performance	Faster (Uses Hashing)	Slower (No Key Lookup)

✦ Example:

```
HashMap<Integer, String> map = new HashMap<>();  
map.put(1, "One");
```

```
HashSet<Integer> set = new HashSet<>();  
set.add(1);
```

24. How to Check if an Array is Empty or Null?

✦ Example:

```
int[] arr = null;  
if (arr == null || arr.length == 0) {  
    System.out.println("Array is empty or null");  
}
```

25. Can We Execute Java Without `main()` ?

✓ Before Java 7 (Using Static Block), Not Possible in Java 8+.

```
static {  
    System.out.println("Executed without main");  
    System.exit(0);  
}
```

26. Can We Call a Non-Static Variable in a Static Method?

✗ No, because static methods belong to the class and don't have access to instance variables.

✦ Example:

```
class Example {  
    int x = 10;  
  
    static void display() {  
        System.out.println(x); // ✗ Compilation Error  
    }  
}
```

```
}  
}
```

✓ **Solution:** Create an object inside the static method.

```
class Example {  
    int x = 10;  
  
    static void display() {  
        Example obj = new Example();  
        System.out.println(obj.x); // ✓ Works fine  
    }  
}
```

27. Can I Execute Multiple Catch Blocks Without try?

✗ No, a try block is required for catch blocks to execute.

★ **Example:**

```
// ✗ Compilation Error  
catch (Exception e) {  
    System.out.println("Catch block");  
}
```

✓ **Correct Way:**

```
try {  
    int a = 10 / 0;  
} catch (ArithmeticException e) {  
    System.out.println("Arithmetic Exception");  
} catch (Exception e) {  
    System.out.println("General Exception");  
}
```

28. How to Achieve Serialization and Deserialization?

✓ **Serialization:** Converting an object into a byte stream (for file storage or transmission).

✓ **Deserialization:** Converting byte stream back into an object.

★ **Example:**

```
import java.io.*;  
  
class Student implements Serializable {  
    int id;  
    String name;  
  
    Student(int id, String name) {  
        this.id = id;  
        this.name = name;  
    }  
}
```

```

}

public class SerializationDemo {
    public static void main(String[] args) throws IOException,
    ClassNotFoundException {
        // Serialization
        Student s1 = new Student(1, "John");
        FileOutputStream fos = new FileOutputStream("student.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(s1);
        oos.close();

        // Deserialization
        FileInputStream fis = new FileInputStream("student.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Student s2 = (Student) ois.readObject();
        ois.close();

        System.out.println(s2.id + " " + s2.name);
    }
}

```

29. What Happens If `main()` is Declared as Private?

✗ The program compiles, but it fails to run because the JVM cannot find the `main()` method.

★ **Example:**

```

class Test {
    private static void main(String[] args) {
        System.out.println("Hello");
    }
}

```

★ **Error:**

Main method not found in class Test

30. What are the Classes Available in a List Interface?

✓ The `List` interface in Java has multiple implementations, including:

Class	Description
<code>ArrayList</code>	Fast retrieval, dynamic array
<code>LinkedList</code>	Fast insertions, uses nodes
<code>Vector</code>	Thread-safe, legacy

Class	Description
Stack	LIFO data structure

31. What is the Use of Constructors in Java?

✓ **Constructors initialize an object when it's created.**

★ Example:

```
class Example {
    int x;

    Example(int value) {
        x = value;
        System.out.println("Constructor executed: " + x);
    }

    public static void main(String[] args) {
        Example obj = new Example(10);
    }
}
```

32. What is HashMap? Can We Store Objects in HashMap?

✓ **HashMap** stores key-value pairs. We **can** store objects as keys or values.

★ Example:

```
import java.util.*;

class Employee {
    int id;
    String name;

    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

public class HashMapDemo {
    public static void main(String[] args) {
        HashMap<Integer, Employee> map = new HashMap<>();
        map.put(1, new Employee(101, "Alice"));
        map.put(2, new Employee(102, "Bob"));

        Employee e = map.get(1);
        System.out.println(e.name); // Output: Alice
    }
}
```

33. Difference Between `HashMap` and `HashSet`

Feature	HashMap	HashSet
Stores	Key-Value Pairs	Unique Elements
Allows Duplicates?	Keys ✗, Values ✓	✗
Performance	Faster (Uses Hashing)	Slower

★ Example:

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "One");

HashSet<Integer> set = new HashSet<>();
set.add(1);
```

34. How to Check if an Array is Empty or Null?

★ Example:

```
int[] arr = null;
if (arr == null || arr.length == 0) {
    System.out.println("Array is empty or null");
}
```

35. Can We Execute Java Without `main()`?

✓ Before Java 7 (Using Static Block), Not Possible in Java 8+.

```
static {
    System.out.println("Executed without main");
    System.exit(0);
}
```

36. Where Did You Use `HashMap` in Your Project?

✓ Example Usage in Automation Framework:

- **Scenario:** Storing **test data** for different test cases.
- **Use Case:** Instead of using Excel or a JSON file, I used a `HashMap` to store key-value pairs dynamically.

★ Example:

```
HashMap<String, String> testData = new HashMap<>();
```

```
testData.put("username", "admin");
testData.put("password", "Admin123");

// Using the HashMap in Selenium
driver.findElement(By.id("username")).sendKeys(testData.get("username"));
driver.findElement(By.id("password")).sendKeys(testData.get("password"));
```

✓ This makes data retrieval **faster** and **efficient** during test execution.

37. Where Did You Use OOPs Concepts in Your Automation Framework?

1. Encapsulation

- Used in **Page Object Model (POM)** to keep web elements and methods private.
- Example:

```
public class LoginPage {
    private WebDriver driver;
    private By username = By.id("username");

    public void enterUsername(String user) {
        driver.findElement(username).sendKeys(user);
    }
}
```

2. Inheritance

- Used in **Base Class**, where all common functionalities (browser setup, teardown) are inherited by test classes.

```
public class BaseTest {
    WebDriver driver;
    @BeforeMethod
    public void setup() {
        driver = new ChromeDriver();
    }
}

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

3. Polymorphism

- **Method Overloading:** Used in **utility classes**.

```
public class Utility {
    public void waitForElement(WebElement element) { }
    public void waitForElement(By locator) { }
}
```
- **Method Overriding:** Used in Selenium **WebDriver** (ChromeDriver, FirefoxDriver).

4. Abstraction

- Implemented using **interfaces**, such as WebDriver, RemoteWebDriver.
-

38. Access Modifiers in Java and Their Scope

Modifier	Scope
private	Only within the same class
default	Within the same package
protected	Same package + subclasses
public	Accessible from anywhere

★ Example:

```
class Test {  
    private int a = 10; // Only inside this class  
    protected int b = 20; // Accessible in subclasses  
    public int c = 30; // Accessible everywhere  
}
```

39. What is Meant by Thread?

- ✓ A **Thread** is a lightweight subprocess used for parallel execution.
- ✓ In Java, threads can be created using:

- **Extending Thread class**
- **Implementing Runnable interface**

★ Example:

```
class MyThread extends Thread {  
    public void run() {  
        System.out.println("Thread is running...");  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        MyThread t1 = new MyThread();  
        t1.start();  
    }  
}
```

40. What is a Singleton Class in Java?

- ✓ A **Singleton class** ensures **only one instance** is created.

★ Example:

```
class Singleton {  
    private static Singleton instance;  
  
    private Singleton() {} // Private Constructor
```

```

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}

```

✓ Used in Selenium for **WebDriver instance management**.

41. Difference Between Static Binding and Dynamic Binding?

Feature	Static Binding (Early Binding)	Dynamic Binding (Late Binding)
Binding Time	Compile-time	Runtime
Method Type	Static methods & private methods	Overridden methods
Example	Method Overloading	Method Overriding

★ Example:

```

class Parent {
    void display() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    void display() {
        System.out.println("Child");
    }
}

public class Test {
    public static void main(String[] args) {
        Parent obj = new Child(); // Dynamic Binding
        obj.display(); // Calls Child's display method
    }
}

```

42. Is `HashMap` Thread Safe?

✗ No, `HashMap` is not thread-safe.

✓ Use `ConcurrentHashMap` for **thread safety**.

★ Example:

```

Map<String, String> map = Collections.synchronizedMap(new HashMap<>());

```

43. What is `static`? How to Set Value of Static Variable?

✓ `static` means **belongs to the class, not objects**.

★ Example:

```
class Test {
    static int count = 0;

    Test() {
        count++;
    }

    public static void main(String[] args) {
        new Test();
        new Test();
        System.out.println(Test.count); // Output: 2
    }
}
```

44. Can We Overload Private Methods?

✓ **Yes, private methods can be overloaded** within the same class.

★ Example:

```
class Test {
    private void display(int a) {
        System.out.println(a);
    }

    private void display(String b) {
        System.out.println(b);
    }
}
```

45. Can We Extend a Final Class?

✗ **No, a `final` class cannot be extended.**

★ Example:

```
final class Test {}
class SubTest extends Test {} // Compilation Error
```

46. Can We Override Static Methods?

✗ **No, static methods cannot be overridden.**

✓ They can be **hidden** by defining another static method.

★ Example:

```
class Parent {
    static void show() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    static void show() {
        System.out.println("Child");
    }
}
```

47. Can We Overload `main()` Method?

✓ Yes, but only the standard `main(String[] args)` is used by JVM.

★ Example:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Standard Main Method");
    }

    public static void main(int a) {
        System.out.println("Overloaded Main Method");
    }
}
```

48. Can We Initialize a Variable in an Interface?

✓ Yes, but only with `public static final` values.

★ Example:

```
interface Test {
    int a = 10; // Implicitly public, static, final
}
```

49. What Would Happen If Java Allowed Multiple Inheritance?

✗ It would cause the "Diamond Problem".

★ Example:

```
class A {
    void show() { System.out.println("A"); }
}
```

```

class B extends A {
    void show() { System.out.println("B"); }
}

class C extends A, B {} // Compilation Error

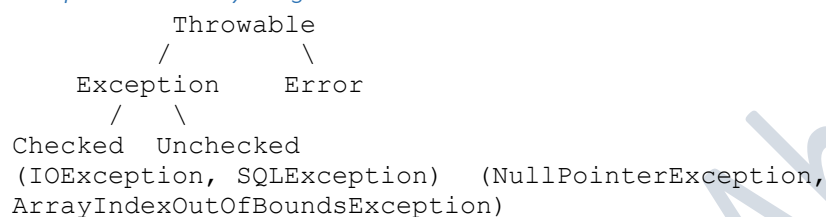
```

✓ Java solves this by **using interfaces instead of multiple inheritance**.

50. Exception Hierarchy in Java

Java exceptions are divided into **Checked Exceptions, Unchecked Exceptions, and Errors**.

Exception Hierarchy Diagram:



✓ **Checked Exceptions** (Compile-time exceptions)

- Must be handled using try-catch or throws.
- Examples: IOException, SQLException.

✓ **Unchecked Exceptions** (Runtime exceptions)

- Occur due to programming mistakes.
- Examples: NullPointerException, ArithmeticException.

✓ **Errors** (Serious system issues)

- Examples: OutOfMemoryError, StackOverflowError.

★ **Example of Handling Exception:**

```

try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero.");
}

```

51. Set vs. Map in Java

Feature	Set	Map
Duplicates	✗ No duplicates	✓ Key-Value pairs
Order	✗ Unordered (HashSet), Ordered (LinkedHashSet)	✗ Unordered (HashMap), Ordered (LinkedHashMap)
Implementation	HashSet, TreeSet, LinkedHashSet	HashMap, TreeMap, LinkedHashMap

★ Example of Set:

```
Set<String> set = new HashSet<>();
set.add("A");
set.add("B");
set.add("A"); // Duplicate ignored
System.out.println(set); // Output: [A, B]
```

★ Example of Map:

```
Map<Integer, String> map = new HashMap<>();
map.put(1, "A");
map.put(2, "B");
System.out.println(map.get(1)); // Output: A
```

51. What is Inheritance?

✓ Inheritance allows one class to acquire properties of another class (extends keyword).

★ Example:

```
class Parent {
    void display() {
        System.out.println("Parent class");
    }
}

class Child extends Parent {
    void show() {
        System.out.println("Child class");
    }
}

public class Test {
    public static void main(String[] args) {
        Child c = new Child();
        c.display(); // Inherited method
        c.show();
    }
}
```

53. Overloading vs. Overriding

Feature	Overloading	Overriding
Definition	Multiple methods with the same name but different parameters	Redefining a method from the parent class in the child class
Return Type	Can change	Must be the same or covariant
Scope	Same class	Parent-child relationship

★ Overloading Example:

```
class Test {  
    void add(int a, int b) {}  
    void add(double a, double b) {}  
}
```

★ Overriding Example:

```
class Parent {  
    void display() {}  
}  
  
class Child extends Parent {  
    @Override  
    void display() {} // Overriding parent method  
}
```

54. Encapsulation vs. Abstraction

Feature	Encapsulation	Abstraction
Definition	Hides data using private access modifiers	Hides implementation using abstract classes & interfaces
Example	Getters and Setters	<code>abstract</code> keyword

★ Encapsulation Example:

```
class BankAccount {  
    private int balance;  
    public void setBalance(int balance) { this.balance = balance; }  
    public int getBalance() { return balance; }  
}
```

★ Abstraction Example:

```

abstract class Vehicle {
    abstract void start();
}

class Car extends Vehicle {
    void start() { System.out.println("Car starts"); }
}

```

55. Throw vs. Throws

Feature	throw	throws
Purpose	Used to explicitly throw an exception	Declares exceptions that might be thrown
Placement	Inside method	In method signature

★ Example of throw:

```
throw new ArithmeticException("Cannot divide by zero");
```

★ Example of throws:

```

void test() throws IOException {
    throw new IOException("File not found");
}

```

56. What is Polymorphism?

✓ Ability to take **multiple forms** (Method Overloading + Method Overriding).

★ Example:

```

class Animal {
    void sound() { System.out.println("Animal makes sound"); }
}

class Dog extends Animal {
    void sound() { System.out.println("Dog barks"); }
}

```

57. How and When to Use Interfaces?

✓ When to Use?

- When multiple classes share the same behavior.
- When **multiple inheritance** is required.

★ Example:

```
interface Animal {  
    void makeSound();  
}  
  
class Dog implements Animal {  
    public void makeSound() {  
        System.out.println("Bark");  
    }  
}
```

58. Can We Instantiate an Interface?

✗ No, interfaces cannot be instantiated.

★ Example:

```
Animal a = new Animal(); // Compilation Error
```

59. Can We Overload `main()` Method in Java?

✓ Yes, but JVM calls only `main(String[] args)`.

★ Example:

```
public class Test {  
    public static void main(String[] args) { System.out.println("Main  
method"); }  
    public static void main(int a) { System.out.println("Overloaded main");  
}  
}
```

60. Can We Override a Constructor?

✗ No, constructors cannot be overridden.

61. Where Do You Use Polymorphism in Java?

✓ In Selenium Automation:

```
WebDriver driver = new ChromeDriver(); // Dynamic Polymorphism  
driver = new FirefoxDriver();
```

62. What is `System.out.println()` and Its Use?

✓ `System.out.println()` is used for console output.

- `System`: Predefined final class.
- `out`: `PrintStream` object.
- `println()`: Method to print data.

★ Example:

```
System.out.println("Hello World");
```

63. Difference Between `final` and `finally`

Feature	<code>final</code>	<code>finally</code>
Purpose	Used for constants and preventing inheritance	Used in exception handling
Example	<code>final int a = 10;</code>	<code>finally { cleanup code }</code>

★ Example of `finally`:

```
try {  
    int a = 10 / 0;  
} catch (Exception e) {  
    System.out.println("Error");  
} finally {  
    System.out.println("Cleanup Code");  
}
```

64. Can We Use Multiple `catch` Blocks?

✓ Yes, multiple `catch` blocks can be used.

★ Example:

```
try {  
    int a = 10 / 0;  
} catch (ArithmeticException e) {  
    System.out.println("Arithmetic Error");  
} catch (Exception e) {  
    System.out.println("General Error");  
}
```

65. Difference Between Apache POI and JXL

Feature	Apache POI	JXL
File Format	<code>.xls</code> and <code>.xlsx</code>	<code>.xls</code> only

Feature	Apache POI	JXL
Latest Updates	Actively maintained	Deprecated
Features	Read/Write both	Read-only support

★ Example Using Apache POI:

```
FileInputStream file = new FileInputStream("test.xlsx");
XSSFWorkbook workbook = new XSSFWorkbook(file);
```

Java Core Concepts

1. How to prevent a method from being overridden in Java?

- Use the `final` keyword before the method declaration.

```
2. class Parent {
3.     final void display() {
4.         System.out.println("This method cannot be overridden");
5.     }
6. }
7. class Child extends Parent {
8.     // Cannot override the display() method due to final keyword
9. }
```

10. Why is the main method static?

- The `main` method is `static` because it is the entry point of the Java application and needs to be called without creating an object of the class.

11. What is the use of static variables?

- Static variables are shared among all instances of a class. They store class-level data instead of instance-specific data.

12. Difference between List and Set?

Feature	List	Set
Duplicates	Allows	Doesn't allow
Order	Maintains insertion order	No guaranteed order
Implementations	ArrayList, LinkedList	HashSet, TreeSet

13. How will you access default and protected members of a class?

- Default** members can be accessed within the same package.
- Protected** members can be accessed within the same package and by subclasses in different packages.

14. Why is object creation not possible for Abstract classes?

- Abstract classes may have incomplete methods (abstract methods), so they cannot be instantiated directly.

15. Design Patterns in Java?

- Singleton, Factory, Observer, Strategy, Prototype, etc.

16. What do all classes in the Java Collection Framework have?

- They implement `Collection`, `List`, `Set`, `Queue`, or `Map` interfaces.

17. When to use Abstraction vs. Interface?

- **Abstraction:** Used when classes have some common implementation.
- **Interface:** Used when multiple classes should follow a common behavior but have different implementations.

18. Does Java provide a default constructor?

- Yes, if no constructor is explicitly defined.

11. Difference between ArrayList and LinkedList and when to use them?

- ArrayList is preferred for fast random access, whereas LinkedList is better for frequent insertions and deletions.

12. Difference between these two declarations?

```
List<String> list = new ArrayList<String>(); // Recommended for flexibility
ArrayList<String> list = new ArrayList<String>(); // More specific, less flexible
```

- The first allows switching implementations easily.

13. Difference between HashMap and MultiMap?

- HashMap<K, V> allows only one value per key.
- MultiMap<K, V> (from Apache Commons or Guava) allows multiple values per key.

14. When should a method be static vs. non-static?

- **Static:** When the method doesn't depend on instance variables.
- **Non-static:** When the method needs instance-specific data.

15. How does HashMap work internally?

- It uses an array of buckets and stores key-value pairs using hashing.

16. What executes first: constructor or main method?

- The main() method runs first. If it creates an object, then the constructor executes.

17. What is a POJO and why is it used?

- A Plain Old Java Object (POJO) is a simple class used to encapsulate data.

18. How do you call a method from a parent class when child class overrides it?

- Use super.methodName();

Java Programming

1. Swap Two Strings Without Using a Third Variable

```
class SwapStrings {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";

        str1 = str1 + str2; // HelloWorld
        str2 = str1.substring(0, str1.length() - str2.length());
        str1 = str1.substring(str2.length());

        System.out.println("After Swapping: str1 = " + str1 + ", str2 = " +
str2);
    }
}
```

2. Find Duplicates in a String

```
import java.util.HashMap;

class DuplicateCharacters {
    public static void main(String[] args) {
        String str = "programming";
        HashMap<Character, Integer> charCount = new HashMap<>();

        for (char c : str.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        for (char c : charCount.keySet()) {
            if (charCount.get(c) > 1) {
                System.out.println(c + " appears " + charCount.get(c) + "
times");
            }
        }
    }
}
```

3. Find String Length Without Using length()

```
class StringLength {
    public static void main(String[] args) {
        String str = "HelloWorld";
        int count = 0;

        for (char c : str.toCharArray()) {
            count++;
        }

        System.out.println("Length of String: " + count);
    }
}
```

4. Find the Largest Number in an Array

```
class LargestNumber {
    public static void main(String[] args) {
        int[] arr = {10, 45, 67, 89, 23};
        int max = arr[0];

        for (int num : arr) {
            if (num > max) {
                max = num;
            }
        }

        System.out.println("Largest Number: " + max);
    }
}
```

5. Reverse a String Without Using `reverse()`

```
class ReverseString {
    public static void main(String[] args) {
        String str = "Hello";
        String rev = "";

        for (int i = str.length() - 1; i >= 0; i--) {
            rev += str.charAt(i);
        }

        System.out.println("Reversed String: " + rev);
    }
}
```

6. Fibonacci Series

```
class FibonacciSeries {
    public static void main(String[] args) {
        int n = 10, first = 0, second = 1;

        System.out.print("Fibonacci Series: " + first + " " + second);

        for (int i = 2; i < n; i++) {
            int next = first + second;
            System.out.print(" " + next);
            first = second;
            second = next;
        }
    }
}
```

7. Print Even Numbers from an Array

```
class EvenNumbers {
    public static void main(String[] args) {
        int[] arr = {12, 3, 7, 8, 10, 21, 30};

        System.out.print("Even Numbers: ");
        for (int num : arr) {
            if (num % 2 == 0) {
                System.out.print(num + " ");
            }
        }
    }
}
```

8. Find Special Characters, Numbers, Capital & Small Letters in a String

```
class CharacterCount {
    public static void main(String[] args) {
        String str = "Hello@123World!";
        int letters = 0, digits = 0, special = 0, upper = 0, lower = 0;

        for (char c : str.toCharArray()) {
            if (Character.isLetter(c)) {
                letters++;
                if (Character.isUpperCase(c)) upper++;
                else lower++;
            } else if (Character.isDigit(c)) {
                digits++;
            } else {
                special++;
            }
        }

        System.out.println("Letters: " + letters + ", Uppercase: " + upper
+ ", Lowercase: " + lower);
        System.out.println("Digits: " + digits);
        System.out.println("Special Characters: " + special);
    }
}
```

9. Check if a Number is a Palindrome

```
class PalindromeNumber {
    public static void main(String[] args) {
        int num = 121, reversed = 0, temp = num;

        while (temp > 0) {
            int digit = temp % 10;
            reversed = reversed * 10 + digit;
            temp /= 10;
        }

        if (num == reversed) {
            System.out.println(num + " is a Palindrome");
        } else {
        }
```

```
        System.out.println(num + " is not a Palindrome");
    }
}
}
```

10. Reverse an Integer Without Using Built-in Method

```
class ReverseInteger {
    public static void main(String[] args) {
        int num = 12345, reversed = 0;

        while (num > 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }

        System.out.println("Reversed Number: " + reversed);
    }
}
```

11. Check if Two Strings Are Anagrams

```
import java.util.Arrays;

class AnagramCheck {
    public static boolean isAnagram(String str1, String str2) {
        char[] arr1 = str1.toCharArray();
        char[] arr2 = str2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }

    public static void main(String[] args) {
        String str1 = "listen";
        String str2 = "silent";

        if (isAnagram(str1, str2)) {
            System.out.println("The strings are anagrams.");
        } else {
            System.out.println("The strings are not anagrams.");
        }
    }
}
```

12. Find the Highest Number in an Array

```
class HighestNumber {
    public static void main(String[] args) {
        int[] arr = {34, 78, 12, 90, 55};
        int max = arr[0];

        for (int num : arr) {
            if (num > max) {

```

```

        max = num;
    }
}

System.out.println("Highest Number: " + max);
}
}

```

Manual Testing

1. What is a Test Plan?

A **Test Plan** is a document that outlines the scope, objectives, approach, and resources required for testing a software product. It includes details about:

- Features to be tested
- Testing strategies
- Test environment
- Entry & exit criteria
- Roles and responsibilities
- Risks & mitigation plans

2. Bug Life Cycle (Defect Life Cycle)

The **Bug Life Cycle** consists of different stages a defect goes through from identification to closure:

1. **New** – Bug is reported.
2. **Assigned** – Assigned to a developer.
3. **Open** – Developer starts working on it.
4. **Fixed** – Developer fixes the bug.
5. **Retest** – QA retests the fix.
6. **Verified** – If fixed, marked as verified.
7. **Reopened** – If the issue persists.
8. **Closed** – If the issue is resolved.

3. Difference between Smoke and Sanity Testing

Feature	Smoke Testing	Sanity Testing
Purpose	Ensures basic functionalities work	Ensures specific functionalities work
Scope	Broad	Narrow

Feature	Smoke Testing	Sanity Testing
Performed	Initial testing after build deployment	After minor changes or bug fixes
Automation	Often automated	Usually manual

4. Difference between Regression and Retesting

Feature	Regression Testing	Retesting
Purpose	Ensures new changes don't break existing functionality	Tests failed test cases again
Scope	Wide	Narrow
Dependency	Independent of bug fixes	Dependent on specific defect fixes

5. Difference between Functional and Regression Testing

Feature	Functional Testing	Regression Testing
Focus	Validates new functionality	Ensures no impact on existing features
Scope	Specific features	All previously tested features

6. Difference between Severity and Priority

Feature	Severity	Priority
Meaning	Impact of defect on the system	Urgency of fixing the defect
Set By	Tester	Developer or Product Manager
Example	App crashes (High Severity)	Spelling mistake in UI (Low Severity)

7. Difference between Test Plan and Test Strategy

Feature	Test Plan	Test Strategy
Purpose	Project-specific document	Organization-level document
Defines	Testing approach for a project	General testing guidelines

8. Difference between Boundary Value Analysis (BVA) and Equivalence Partitioning (EP)

Feature	BVA	EP
Purpose	Tests boundary limits	Tests input groups
Example	If input range is 1-100, test 0,1,100,101	Divide inputs as valid (1-100) and invalid (negative, 101+)

9. Difference between White Box and Black Box Testing

Feature	White Box Testing	Black Box Testing
Who performs	Developers	Testers
Knowledge required	Code knowledge required	No coding knowledge needed

10. How to select test cases for automation in a 1000-test case suite?

- Automate repetitive and high-priority tests.
- Focus on regression test cases.
- Avoid one-time test cases.

11. Can a method return multiple values in Java?

Yes, using:

1. **Array**
2. **Collection (List, Map)**
3. **Object Wrapping**
4. **Pair/Tuple (JavaFX)**

```
class MultipleValues {  
    public static int[] getNumbers() {  
        return new int[]{10, 20};  
    }  
}
```

12. What is Bug Triage?

Bug triage is the process of prioritizing and categorizing defects based on their impact and urgency.

13. What is Exploratory Testing?

Exploratory testing is an **ad-hoc testing** approach where testers explore the application without predefined test cases.

14. What is Ad-hoc Testing?

Ad-hoc testing is an **informal testing** method performed without test cases or documentation.

15. What is Build Acceptance Testing (BAT)?

BAT ensures a newly deployed build is stable enough for further testing.

16. Difference between Validation and Verification

Feature	Validation	Verification
Purpose	Ensures product meets user needs	Ensures product is built correctly
Example	User feedback, UAT	Code reviews, unit testing

17. Severity-Priority Example

Scenario	Severity	Priority
Crash in payment system	High	High
Typo in privacy policy	Low	Low
Logo is incorrect	Low	High

18. Test Case Priority Execution Order (-1, 0, 1, 2)

Execution order: -1 → 0 → 1 → 2
(Lower priority numbers are executed first.)

19. Difference between Inner Join and Outer Join

Feature	Inner Join	Outer Join
Output	Common records	All records with NULLs for missing values

Example:

```
SELECT * FROM Customers INNER JOIN Orders ON Customers.ID =  
Orders.CustomerID;
```

20. Difference between DELETE, DROP, and TRUNCATE

Feature	DELETE	DROP	TRUNCATE
Removes	Rows	Table	Data only
Rollback?	Yes	No	No

21. Test Design Techniques

- BVA (Boundary Value Analysis)
 - EP (Equivalence Partitioning)
 - Decision Table Testing
 - State Transition Testing
-

22. What is a Deferred Bug?

A defect that is **not fixed immediately** due to lower priority.

23. How to Decide What Tests to Automate?

- **Repetitive** and **time-consuming** tests.
 - **High-priority** test cases.
-

24. When to Stop Testing?

- When **exit criteria** are met.
 - When test execution is **stable**.
 - **Deadlines** reached.
-

25. How to Reduce Test Execution Time?

- **Parallel Execution**
 - **Optimized Locators**
 - **Data-driven Testing**
-

Here are the answers to your questions:

26. How many test cases are in your regression test suite? How long does it take to execute?

- The number of test cases in a **regression test suite** varies depending on the project. Typically, it contains **300-500 test cases**.
 - **Execution time** depends on:
 - The number of automated vs. manual test cases.
 - The test execution environment (local vs. CI/CD).
 - Parallel execution setup.
 - **Example:** If running **automated tests in parallel**, execution may take **1-2 hours**, whereas **manual execution may take 2-3 days**.
-

27. How to cover character keyboard operations using user-defined keywords?

- If using **Selenium with Java**, the **Actions class** can be used for keyboard operations.
- Example:

```
Actions action = new Actions(driver);  
action.sendKeys(Keys.ARROW_DOWN).perform();
```
- In **Cucumber with Selenium**, you can create a step definition for character input:

```
@When("User presses {string} key")  
public void userPressesKey(String key) {  
    Actions action = new Actions(driver);  
    action.sendKeys(Keys.valueOf(key.toUpperCase())).perform();  
}
```

28. What are the most important things to define in a bug?

A good bug report should include:

1. **Bug ID** – Unique identifier.
2. **Title** – A short description of the issue.
3. **Severity & Priority** – Impact level.
4. **Environment** – OS, browser, device.
5. **Steps to Reproduce** – Exact steps to recreate the bug.
6. **Expected vs. Actual Behavior**.
7. **Attachments** – Screenshots, logs.
8. **Status** – New, Assigned, Open, Fixed, Verified, Closed.

29. Can you share any achievements in automation?

- **Reduced execution time** by automating **manual test cases**, improving efficiency by **50%**.
- **Integrated Cucumber with Jenkins**, enabling CI/CD pipeline execution.
- **Developed a reusable framework** that reduced script maintenance by **30%**.
- **Identified critical defects** early using automated regression suites.

30. Difference between BDD and TDD?

Feature	BDD (Behavior-Driven Development)	TDD (Test-Driven Development)
Focus	User behavior & business requirements	Code functionality
Syntax	Uses Gherkin (Given-When-Then)	Uses JUnit/TestNG
Who Writes	QA, BA, Devs	Developers

Feature	BDD (Behavior-Driven Development)	TDD (Test-Driven Development)
Example	Given user is logged in, When they click logout, Then they should be logged out	<code>assertEquals(loginStatus, false);</code>

31. What is a Test Plan? Steps to create one?

A **Test Plan** outlines the testing approach. Steps to create it:

1. **Scope & Objectives**
2. **Test Strategy**
3. **Test Environment**
4. **Test Deliverables**
5. **Test Schedule**
6. **Roles & Responsibilities**
7. **Risk Management**

32. What are inbound and outbound testing?

- **Inbound Testing** – Validates **data coming into a system** (e.g., API requests, file uploads).
- **Outbound Testing** – Validates **data going out** (e.g., reports, email notifications).

33. Smoke, Regression, and Sanity Testing?

Type	Purpose	When Performed
Smoke	Checks if the application is stable enough for testing	After a new build
Sanity	Verifies specific fixes & functionalities	After minor changes
Regression	Ensures new changes don't break existing features	After new feature additions

34. What is the next step if a developer rejects an open defect?

1. **Reproduce the issue** and gather evidence.
2. **Check logs/screenshots** to validate the issue.
3. **Discuss with the developer** and justify the defect.
4. **Escalate** if necessary.

35. Explain Test Metrics

Test metrics help measure testing effectiveness. Examples:

1. **Test Execution Metrics** – % of test cases executed.
 2. **Defect Density** – No. of defects per module.
 3. **Defect Leakage** – Bugs found after release.
-

36. How would you prioritize tests?

- Critical business functionalities first.
 - High-risk areas.
 - Frequent defect-prone areas.
 - Time-sensitive tests.
-

37. How do you perform an Automation Code Review?

1. Follow coding standards.
 2. Check for reusability (avoid hardcoding).
 3. Optimize locators.
 4. Use Page Object Model (POM).
-

38. How to segregate 250 manual test cases into Smoke, Sanity, Regression?

- **Smoke:** ~10-15% (Critical workflows, login, checkout).
 - **Sanity:** ~20-25% (Specific features, bug fixes).
 - **Regression:** ~60-70% (All functional & integration cases).
-

39. What makes a test case a good candidate for automation?

- Stable functionality.
 - Frequent execution.
 - Time-consuming manual execution.
-

40. When are Regression, Sanity, and Smoke test scripts executed?

Type	When
Smoke	Before full testing starts
Sanity	After minor changes
Regression	After major changes

Agile Methodology & Scrum Questions

1. What is Agile methodology?

Agile is a **software development approach** that focuses on **iterative, incremental delivery** of software. It emphasizes **flexibility, collaboration, customer feedback, and continuous improvement**.

- Agile follows frameworks like **Scrum, Kanban, SAFe**.
- Work is divided into **small iterations** (Sprints).
- Frequent releases ensure quick adaptation to changes.

2. What is Scrum, and who is your Scrum Master?

Scrum is an Agile framework that focuses on iterative development using **Sprints** (typically **2-4 weeks** long).

- **Scrum Master**: Facilitates Agile processes, removes blockers, and ensures adherence to Scrum practices.
- **Roles in Scrum**:
 - **Product Owner** – Defines requirements.
 - **Scrum Master** – Facilitates Scrum practices.
 - **Development Team** – Executes the work.

3. What Agile ceremonies do you follow?

Agile Scrum includes the following ceremonies:

1. **Sprint Planning** – Defines sprint goals and backlog items.
2. **Daily Stand-up (Daily Scrum)** – Short meeting to track progress.
3. **Sprint Review** – Demonstration of completed work.
4. **Sprint Retrospective** – Discussion to improve the next sprint.

4. What is a Retrospective Meeting?

A **Sprint Retrospective** is conducted at the **end of a sprint** to reflect on:

- What went **well**?
- What went **wrong**?
- What can be **improved**?

Example:

- **Actionable improvement:** Reduce unnecessary meetings to improve developer focus.
-

5. Describe Scrum Ceremony

Ceremony	Purpose
Sprint Planning	Decide sprint backlog, define scope
Daily Stand-up	Track progress, discuss blockers
Sprint Review	Demonstrate completed work
Sprint Retrospective	Discuss improvements for the next sprint

6. When do you automate – in the current sprint or the next sprint?

- **Depends on project requirements.**
 - Ideally, automation is done **in the same sprint** if time allows.
 - If the feature is complex, automation is done in the **next sprint** after manual testing.
-

7. Explain Velocity in Sprint.

- **Velocity = Total Story Points Completed / Number of Sprints**
 - It helps in **estimating the team's capacity** for future sprints.
 - Example: If a team completes **40 story points** in Sprint 1, **35 in Sprint 2**, then the **average velocity is 37.5**.
-

8. In a tight sprint schedule, if a new requirement is added, how will you handle it?

- **Assess the impact** of the new requirement.
 - Discuss with the **Scrum Master & Product Owner**.
 - **Prioritize** the backlog items.
 - **Negotiate scope changes** or shift tasks to the next sprint.
-

9. What is a Backlog in Scrum?

A **Backlog** is a prioritized list of features, tasks, or bugs maintained by the **Product Owner**.

- **Product Backlog** – High-level features for the project.
 - **Sprint Backlog** – Tasks selected for a sprint.
-

10. You have 30+ sprints in your release. How will you design your test scripts and run them?

1. **Identify reusable test cases** for automation.
 2. **Prioritize critical functionalities** for regression testing.
 3. **Execute automated regression** after every sprint.
 4. **Use parallel execution** (Selenium Grid, Jenkins).
 5. **Optimize test execution time** by selecting relevant test cases for each sprint.
-

Managerial Round Questions & Answers

1. How will you be an asset to the team?

I bring **3.5 years of experience** in **manual and automation testing**, ensuring software quality. My expertise in **Selenium, Java, Cucumber, and TestNG** helps in creating **robust automation frameworks**. I also excel in **collaborating with developers, identifying defects early, and improving test efficiency**.

Example: In my current role, I optimized the **automation suite execution time from 3 hours to 1.5 hours**, improving CI/CD efficiency.

2. Why are you looking for a change?

I am looking for **career growth, new challenges, and opportunities to enhance my skills**. I want to work on **more complex automation projects** and contribute to a **dynamic team** where I can **learn new technologies** and make a greater impact.

3. How soon can you join?

I am currently serving a **[mention notice period]** notice period, but I am open to negotiation based on the urgency of the requirement.

4. Suppose we assign you only manual testing for six months to a year. What will you do?

While my expertise is in **automation**, I understand the **importance of manual testing** in identifying **edge cases** and ensuring **better test coverage**. I would:

- Optimize test case execution using **better test design techniques**.
 - Work on **test case reusability**.
 - Explore **automation opportunities** to enhance efficiency.
-

5. How interested are you in learning new technologies?

I am always eager to learn. **Technology evolves constantly**, and I keep myself updated with the latest trends in **test automation, API testing, CI/CD tools, and cloud-based testing**.

Example: Recently, I started exploring **Playwright for UI automation** and **Postman for API testing** to expand my skill set.

6. Failures in your work life.

Failures are part of learning.

Example: In an earlier project, I **missed testing a critical API integration**, which led to an issue in production. After this, I introduced **API test automation** in our framework, which prevented such gaps in the future.

7. As a Lead, how do you define the quality of a product before release?

- Ensuring **high test coverage** through **manual & automation testing**.
 - Conducting **regression testing** to prevent functionality breakage.
 - Checking for **critical defects** and ensuring they are **resolved** before release.
 - Performing **performance and security testing** (if required).
 - Validating the product against **business requirements**.
-

8. A new team member joins when a deadline is near. How do you onboard them efficiently?

1. **Quick knowledge transfer** – Explain high-priority tasks.
 2. **Pair testing** – Assign a buddy to help them.
 3. **Assign simpler tasks** – Involve them in **test execution** rather than complex scripting.
 4. **Utilize documentation** – Share existing test cases, automation scripts, and previous defect reports.
-

9. As a QA, where do you see yourself after 3 years?

I see myself as a **Senior QA Automation Engineer or QA Lead**, leading automation initiatives, mentoring juniors, and improving testing strategies to enhance product quality.

10. What are your strengths and weaknesses?

✓ Strengths:

- Strong **analytical skills** to identify test scenarios.
- Expertise in **automation frameworks**.
- **Good communication & collaboration** skills.

✗ Weaknesses:

- I tend to **focus on perfection**, which sometimes affects timelines.
 - I am **working on improving my knowledge of performance testing**.
-

11. Best practices you learned & the impact they made?

Before: We used **Excel for test case management**, which was time-consuming.

After: Implemented **TestRail/JIRA**, reducing test documentation time by **40%**.

Before: Regression execution **took 3-4 hours** manually.

After: Introduced **parallel execution in Selenium Grid**, reducing execution time to **1.5 hours**.

12. After running regression tests, if new regression bugs are found, how will you prioritize them?

1. **Bugs breaking core functionality** (high severity & high priority) → **Fix immediately.**
 2. **Bugs affecting secondary features** → **Fix in the next sprint.**
 3. **Minor UI bugs** → Logged but fixed in later sprints based on priority.
-

***** Best Of Luck *****