Interview Questions & Answers for Experienced Automation Testers

---

Section 1: Selenium WebDriver (20 Questions)

Q1. What are different types of waits in Selenium?

Selenium provides three types of waits to handle synchronization issues:

- **Implicit Wait**: Sets a default waiting time for all elements throughout the session.

  ```
  driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
  ```

- **Explicit Wait**: Waits for specific conditions on particular elements before performing actions.

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

- **Fluent Wait**: Provides more control with custom polling frequency and exception handling.

  ```
  Wait<WebDriver> wait = new FluentWait<>(driver)
      .withTimeout(Duration.ofSeconds(10))
      .pollingEvery(Duration.ofMillis(500))
      .ignoring(NoSuchElementException.class);
  wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("element
  ")));
  ```

Q2. Explain the difference between findElement() and findElements() methods.

- **findElement()**: Returns a single WebElement; throws NoSuchElementException if not found. `java  WebElement element = driver.findElement(By.id("username"));`

- **findElements()**: Returns a List of WebElements; returns empty list if no elements are found (no exception thrown).

  ```
  List<WebElement> elements = driver.findElements(By.className("item"));
  ```

Use `findElement()` when you expect exactly one element; use `findElements()` when checking if elements exist or iterating through multiple elements.

Q3. What are XPath and CSS Selector? When would you use each?

Both are locator strategies in Selenium:

- **XPath**: A query language for navigating XML/HTML documents. More powerful and flexible.

```
By.xpath("//button[@id='submit' and @class='primary']")
By.xpath("//input[@placeholder='Enter username']")
By.xpath("//div[contains(@class, 'error-message')]")
```

- **CSS Selector**: Uses CSS styling rules. Generally faster than XPath.

```
By.cssSelector("button#submit.primary")
By.cssSelector("input[placeholder='Enter username']")
By.cssSelector("div.error-message")
```

When to use: Use CSS Selector for simple, stable locators (faster performance). Use XPath for complex scenarios like traversing parent elements or multiple conditions.

Q4. How would you handle dynamic web elements or elements with frequently changing IDs?

Strategies to handle dynamic elements:

1. **Use Partial Attribute Matching**:

```
By.xpath("//button[starts-with(@id, 'btn_')]")
By.cssSelector("button[id*='dynamic']")
```

2. **Use Relative XPath Based on Text Content**:

```
By.xpath("//button[contains(text(), 'Submit')]")
By.xpath("//label[text()='Username']/following::input")
```

3. **Use WebDriverWait with Custom Expected Condition**:

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath("//butt
on[contains(@class, 'dynamic')]")));
```

4. **Use Explicit Waits Before Interaction**:

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[@
onclick*='submit']")));
element.click();
```

Q5. Explain the Page Object Model (POM) pattern and its advantages.

Page Object Model is a design pattern that represents each web page as a Java class.

Structure:

```
public class LoginPage {
    private WebDriver driver;

    // Locators
    private By usernameField = By.id("username");
    private By passwordField = By.id("password");
    private By loginButton = By.id("loginBtn");

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

    // Page actions
    public void enterUsername(String username) {
        driver.findElement(usernameField).sendKeys(username);
    }

    public void enterPassword(String password) {
        driver.findElement(passwordField).sendKeys(password);
    }

    public DashboardPage clickLogin() {
        driver.findElement(loginButton).click();
        return new DashboardPage(driver);
    }
}
```

Advantages: - Improved maintainability (locators centralized in one place) - Reduced code duplication - Easy to update locators without touching test logic - Better readability and reusability - Separation of concerns (test logic vs. UI interactions)

Q6. What is the difference between Actions class and direct element methods?

- **Direct Methods**: Perform actions directly on an element. `java element.click(); element.sendKeys("text"); element.submit();`

- **Actions Class**: Simulates complex user interactions using mouse and keyboard.

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

```
actions.dragAndDrop(source, target).perform();
actions.keyDown(Keys.SHIFT).click(element).keyUp(Keys.SHIFT).perform();
```

Use Actions class for: hovering over elements, drag-and-drop, right-click, key combinations, or simulating complex user gestures.

Q7. How do you handle StaleElementReferenceException?

StaleElementReferenceException occurs when a WebElement reference becomes stale (DOM has been refreshed).

Solutions:

1. **Re-locate the element after action**:

```
WebElement element = driver.findElement(By.id("dynamicElement"));
element.click();
// DOM refreshes
element = driver.findElement(By.id("dynamicElement")); // Re-Locate
element.sendKeys("text");
```

2. **Use Explicit Wait with WebDriverWait**:

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

3. **Wrap in Try-Catch and Retry**:

```
int attempts = 0;
while (attempts < 3) {
    try {
        driver.findElement(By.id("element")).click();
        break;
    } catch (StaleElementReferenceException e) {
        attempts++;
    }
}
```

Q8. Explain different locator strategies and their reliability order.

Locator strategies ranked by reliability:

1. **ID** (Most reliable) - Unique, direct, and unchanged

```
By.id("userId")
```

2. **Name** - Usually stable and unique

   ```
   By.name("username")
   ```

3. **CSS Selector** - Fast, stable, and maintainable

   ```
   By.cssSelector("input.form-control[name='email']")
   ```

4. **XPath** - Flexible but can be fragile with DOM changes

   ```
   By.xpath("//input[@type='email']")
   ```

5. **Link Text / Partial Link Text** - Good for links but limited

   ```
   By.linkText("Click Here")
   By.partialLinkText("Click")
   ```

6. **Class Name** - Can be fragile if styling changes

   ```
   By.className("button-primary")
   ```

7. **Tag Name** (Least reliable) - Not unique, too generic

   ```
   By.tagName("button")
   ```

Best Practice: Use ID or CSS selectors; avoid overly complex XPath or tag names.

Q9. What are the differences between driver.navigate(), driver.get(), and driver.switchTo()?

- **driver.get()**: Loads a URL and waits for page load. Blocking operation. `java driver.get("https://example.com");`

- **driver.navigate().to()**: Similar to get() but allows navigation history.

  ```
  driver.navigate().to("https://example.com");
  driver.navigate().back();    // Go to previous page
  driver.navigate().forward(); // Go to next page
  driver.navigate().refresh(); // Refresh current page
  ```

- **driver.switchTo()**: Switches context between frames, windows, alerts, etc.

  ```
  driver.switchTo().frame("iframeId");
  driver.switchTo().parentFrame();
  driver.switchTo().defaultContent();
  driver.switchTo().alert().accept();
  driver.switchTo().window(windowHandle);
  ```

Q10. How do you handle pop-ups, alerts, and browser dialogs?

- **JavaScript Alerts**: `java Alert alert = driver.switchTo().alert(); String alertText = alert.getText(); alert.accept(); // Click OK alert.dismiss(); // Click Cancel alert.sendKeys("text"); // Type in prompt`

- **Window Handles**:

```
String mainWindow = driver.getWindowHandle();
Set<String> allWindows = driver.getWindowHandles();
for (String window : allWindows) {
    driver.switchTo().window(window);
}
driver.switchTo().window(mainWindow); // Back to main
```

- **Frames/iFrames**:

```
driver.switchTo().frame("frameName");    // By name
driver.switchTo().frame(0);              // By index
driver.switchTo().frame(frameElement);   // By element
driver.switchTo().parentFrame();         // Back to parent
driver.switchTo().defaultContent();      // Back to main page
```

Q11. What is implicit wait vs. explicit wait vs. fluent wait in practical scenarios?

| Scenario | Wait Type | Implementation | |———-|———-|———————–| | Page load & all elements should load within time | Implicit | `driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));` | | Wait for specific element visibility before action | Explicit | `new WebDriverWait(driver, Duration.ofSeconds(10)).until(ExpectedConditions.visibilityOfElementLocated(By.id("btn")));` | | Complex condition with custom polling & exceptions | Fluent | `new FluentWait<>(driver).withTimeout(Duration.ofSeconds(10)).pollingEvery(Duration.ofMillis(500)).ignoring(NoSuchElementException.class).until(...);` |

Best Practice: Avoid implicit waits; use explicit waits for better control and faster test execution.

Q12. How would you take screenshots and generate visual reports in Selenium?

```
// Take full page screenshot
File screenshot = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(screenshot, new File("./screenshots/test_" +
System.currentTimeMillis() + ".png"));
```

```java
// Take element screenshot (WebDriver 4.x)
File elementScreenshot = element.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(elementScreenshot, new File("./screenshots/element.png"));

// Screenshot on failure
@AfterMethod
public void tearDown(ITestResult result) {
    if (ITestResult.FAILURE == result.getStatus()) {
        File screenshot = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(screenshot, new File("./screenshots/failure_" +
result.getName() + ".png"));
    }
}
```

For visual reports, integrate with ExtentReports:

```java
extent.attachFile(screenshotPath);
extent.attachBase64String(((TakesScreenshot)
driver).getScreenshotAs(OutputType.BASE64));
```

Q13. Explain how to handle SSL certificate errors in Selenium.

```java
// For Chrome
ChromeOptions options = new ChromeOptions();
options.setAcceptInsecureCerts(true);
WebDriver driver = new ChromeDriver(options);

// For Firefox
FirefoxOptions options = new FirefoxOptions();
options.setAcceptInsecureCerts(true);
WebDriver driver = new FirefoxDriver(options);

// For Edge
EdgeOptions options = new EdgeOptions();
options.setAcceptInsecureCerts(true);
WebDriver driver = new EdgeDriver(options);

// For Internet Explorer
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability(CapabilityType.ACCEPT_INSECURE_CERTS, true);
WebDriver driver = new InternetExplorerDriver(caps);
```

Q14. How do you perform keyboard and mouse interactions using Selenium?

```
// Keyboard interactions
Actions actions = new Actions(driver);
actions.sendKeys(Keys.ENTER).perform();
actions.sendKeys(Keys.TAB).perform();
actions.sendKeys(Keys.ESCAPE).perform();
actions.keyDown(Keys.CONTROL).sendKeys("A").keyUp(Keys.CONTROL).perform(); //
Ctrl+A

// Mouse interactions
actions.click(element).perform();
actions.doubleClick(element).perform();
actions.rightClick(element).perform();
actions.moveToElement(element).perform();
actions.dragAndDrop(source, target).perform();

// Key combinations
actions.keyDown(Keys.SHIFT).click(element1).click(element2).keyUp(Keys.SHIFT)
.perform(); // Multi-select
```

Q15. What is the difference between switchTo() methods and how do you switch between frames and windows?

```
// Switch to frame by index
driver.switchTo().frame(0);

// Switch to frame by name/ID
driver.switchTo().frame("frameName");

// Switch to frame by WebElement
WebElement frameElement = driver.findElement(By.id("frame"));
driver.switchTo().frame(frameElement);

// Switch back to parent frame
driver.switchTo().parentFrame();

// Switch to main page (out of all frames)
driver.switchTo().defaultContent();

// Switch to window
String mainWindow = driver.getWindowHandle();
Set<String> handles = driver.getWindowHandles();
for (String handle : handles) {
```

```
    driver.switchTo().window(handle);
    if (driver.getTitle().equals("Expected Title")) {
        break;
    }
}

// Switch back to main window
driver.switchTo().window(mainWindow);
```

Q16. How do you handle file uploads and downloads in Selenium?

```
// File Upload
WebElement uploadElement = driver.findElement(By.id("fileUpload"));
uploadElement.sendKeys("C:\\path\\to\\file.txt");

// Using Robot class for OS-level file dialogs
Robot robot = new Robot();
robot.keyPress(KeyEvent.VK_CONTROL);
robot.keyPress(KeyEvent.VK_V);
robot.keyRelease(KeyEvent.VK_V);
robot.keyRelease(KeyEvent.VK_CONTROL);

// File Download (configure browser to auto-download)
ChromeOptions options = new ChromeOptions();
HashMap<String, Object> chromePrefs = new HashMap<>();
chromePrefs.put("download.default_directory", "/download/path");
chromePrefs.put("download.prompt_for_download", false);
options.setExperimentalOption("prefs", chromePrefs);
WebDriver driver = new ChromeDriver(options);

// Verify downloaded file
File downloadDir = new File("/download/path");
File[] files = downloadDir.listFiles();
assertTrue(files != null && files.length > 0);
```

Q17. What are Expected Conditions in Selenium WebDriverWait and give 5 examples?

Expected Conditions are predefined conditions to wait for elements:

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

// 1. Presence - element exists in DOM
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("element")));

// 2. Visibility - element visible and rendered on page
```

```
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("element")));

// 3. Clickability - element is visible and enabled
wait.until(ExpectedConditions.elementToBeClickable(By.id("button")));

// 4. Text to be present in element
wait.until(ExpectedConditions.textToBePresentInElementLocated(By.id("label"),
"Expected Text"));

// 5. Staleness - element is no longer attached to DOM
WebElement element = driver.findElement(By.id("temp"));
wait.until(ExpectedConditions.stalenessOf(element));

// 6. Invisibility
wait.until(ExpectedConditions.invisibilityOfElementLocated(By.id("loader")));

// 7. Number of elements
wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.className("item
"), 5));

// 8. URL contains
wait.until(ExpectedConditions.urlContains("login"));

// 9. Alert present
wait.until(ExpectedConditions.alertIsPresent());
```

Q18. How do you execute JavaScript in Selenium?

```
// Execute JavaScript
JavascriptExecutor js = (JavascriptExecutor) driver;

// Get element attribute
String value = (String) js.executeScript("return
document.getElementById('element').value;");

// Click element (bypass clicks intercepted by overlays)
js.executeScript("arguments[0].click();", element);

// Scroll to element
js.executeScript("arguments[0].scrollIntoView(true);", element);

// Scroll to top/bottom
js.executeScript("window.scrollTo(0, 0);");       // Top
js.executeScript("window.scrollTo(0, document.body.scrollHeight);"); //
Bottom
```

```
// Remove element from DOM
js.executeScript("arguments[0].remove();", element);

// Set attribute
js.executeScript("arguments[0].setAttribute('value', 'newValue');", element);

// Highlight element
js.executeScript("arguments[0].style.border='3px solid red';", element);

// Get text content
String text = (String) js.executeScript("return arguments[0].textContent;",
element);

// Wait for AJAX completion (jQuery)
js.executeScript("return jQuery.active == 0");
js.executeScript("return document.readyState").equals("complete");
```

Q19. Explain the headless browser testing and its advantages/disadvantages.

Headless Mode: Browser runs without GUI; faster and uses fewer resources.

Advantages: - Faster execution (no rendering overhead) - Lower resource consumption (RAM & CPU) - CI/CD friendly (no display needed) - Parallel execution-friendly - Better for server environments

Disadvantages: - Cannot see what's happening visually - Some JS/CSS behaviors might differ - Debugging is harder

Implementation:

```
// Chrome Headless
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless=new");
WebDriver driver = new ChromeDriver(options);

// Firefox Headless
FirefoxOptions options = new FirefoxOptions();
options.addArguments("--headless");
WebDriver driver = new FirefoxDriver(options);

// Edge Headless
EdgeOptions options = new EdgeOptions();
options.addArguments("--headless");
WebDriver driver = new EdgeDriver(options);
```

Q20. How do you manage test data and environment configuration in Selenium automation?

```
// Using Properties file
public class ConfigReader {
    private static Properties properties = new Properties();

    static {
        try {
            FileInputStream file = new
FileInputStream("./src/test/resources/config.properties");
            properties.load(file);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String getProperty(String key) {
        return properties.getProperty(key);
    }
}

// Usage
String url = ConfigReader.getProperty("base.url");
String username = ConfigReader.getProperty("test.username");
String password = ConfigReader.getProperty("test.password");

// Using JSON for test data
public class TestDataReader {
    public static Map<String, Object> readTestData(String testDataFile)
throws IOException {
        String content = new
String(Files.readAllBytes(Paths.get(testDataFile)));
        ObjectMapper mapper = new ObjectMapper();
        return mapper.readValue(content, Map.class);
    }
}

// Properties file example (config.properties)
/*
base.url=https://example.com
browser=chrome
wait.timeout=10
test.username=user@example.com
test.password=password123
environment=staging
*/
```

Section 2: Core Java (20 Questions)

Q21. Explain OOP concepts: Encapsulation, Inheritance, Polymorphism, and Abstraction.

- **Encapsulation**: Bundling data (variables) and methods in a class; controlling access using access modifiers. ```java public class Person { private String name; // Private variable

```
  public String getName() {
      return name;
  }

  public void setName(String name) {
      this.name = name;
  }
```

}```

- **Inheritance**: Deriving a new class from an existing class to reuse code and establish relationships.

  ```java
  public class Animal {
      public void eat() { System.out.println("Eating..."); }
  }

  public class Dog extends Animal {
      public void bark() { System.out.println("Barking..."); }
  }
  ```

- **Polymorphism**: Ability of objects to take multiple forms; achieved through method overloading and overriding.

  ```java
  // Overloading
  public void click() { }
  public void click(WebElement element) { }

  // Overriding
  public class Animal {
      public void sound() { System.out.println("Some sound"); }
  }
  public class Dog extends Animal {
      @Override
      public void sound() { System.out.println("Bark"); }
  }
  ```

- **Abstraction**: Hiding complex details and showing only essential features; achieved through abstract classes and interfaces.

  ```java
  public abstract class Browser {
      public abstract void launchBrowser();
      public abstract void closeBrowser();
  ```

```
    }

    public class Chrome extends Browser {
        @Override
        public void launchBrowser() { System.out.println("Launching
    Chrome"); }
        @Override
        public void closeBrowser() { System.out.println("Closing Chrome");
    }
    }
```

Q22. What are the differences between static and non-static members?

| Feature | Static | Non-Static | |———|———–|———–| | Memory | Allocated once at class load time | Allocated per object instance | | Access | Through class name (ClassName.member) | Through object reference (obj.member) | | Modification | Changes affect all instances | Changes affect only that instance | | Inheritance | Cannot be overridden (only hidden) | Can be overridden |

Example:

```
public class Counter {
    public static int staticCount = 0;  // Shared across all instances
    public int nonStaticCount = 0;      // Unique per instance

    public static void incrementStatic() {
        staticCount++;
    }

    public void incrementNonStatic() {
        nonStaticCount++;
    }
}

Counter c1 = new Counter();
Counter c2 = new Counter();

c1.incrementStatic();      // staticCount = 1
c2.incrementStatic();      // staticCount = 2 (both instances see same value)

c1.incrementNonStatic();   // c1.nonStaticCount = 1
c2.incrementNonStatic();   // c2.nonStaticCount = 1 (independent values)
```

Q23. Explain the Exception Hierarchy in Java and how to handle exceptions.

Exception Hierarchy:

```
Throwable
├── Error (System errors, not recoverable)
│   ├── OutOfMemoryError
│   ├── StackOverflowError
├── Exception (Recoverable)
│   ├── Checked Exception (must be caught/declared)
│   │   ├── IOException
│   │   ├── SQLException
│   │   ├── FileNotFoundException
│   ├── Unchecked Exception (inherits RuntimeException)
│       ├── NullPointerException
│       ├── ArrayIndexOutOfBoundsException
│       ├── NoSuchElementException
```

Exception Handling:

```java
// Try-Catch
try {
    WebElement element = driver.findElement(By.id("nonexistent"));
} catch (NoSuchElementException e) {
    System.out.println("Element not found: " + e.getMessage());
}

// Try-Catch-Finally
try {
    int[] arr = {1, 2, 3};
    System.out.println(arr[5]);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Index out of bounds");
} finally {
    System.out.println("This always executes");
}

// Multiple Catch Blocks
try {
    // code
} catch (NoSuchElementException e) {
    System.out.println("Element not found");
} catch (TimeoutException e) {
    System.out.println("Wait timeout");
} catch (Exception e) {
    System.out.println("Generic exception");
}

// Try-Catch-Resources (Auto-closes resources)
try (FileInputStream file = new FileInputStream("test.txt")) {
    // Use file
} catch (IOException e) {
    e.printStackTrace();
}
```

```java
// Throw
public void validateInput(String input) throws IllegalArgumentException {
    if (input == null || input.isEmpty()) {
        throw new IllegalArgumentException("Input cannot be empty");
    }
}
```

Q24. What is the difference between checked and unchecked exceptions?

| Checked | Unchecked | |———|———–| | Extends Exception (not RuntimeException) | Extends RuntimeException | | Must be caught or declared in method signature | Not required to be caught | | Compiler enforces exception handling | Compiler does not enforce | | Examples: IOException, SQLException | Examples: NullPointerException, ArrayIndexOutOfBoundsException |

Example:

```java
// Checked Exception
public void readFile(String path) throws IOException {
    FileInputStream file = new FileInputStream(path);
    // Must handle or throw
}

// Unchecked Exception
public void printArray(int[] arr, int index) {
    System.out.println(arr[index]); // Can throw
ArrayIndexOutOfBoundsException
    // No requirement to handle
}

// Handling checked exception
try {
    readFile("nonexistent.txt");
} catch (IOException e) {
    e.printStackTrace();
}
```

Q25. Explain the difference between String, StringBuilder, and StringBuffer.

| Feature | String | StringBuilder | StringBuffer | |———|——–|————|————–| | Immutable | Yes | No | No | | Thread-Safe | Yes | No | Yes (synchronized) | | Performance | Slow (creates new objects) | Fast | Slower (due to synchronization) | | Use Case | Fixed strings | Single-threaded dynamic strings | Multi-threaded dynamic strings |

Example:

```java
// String - Immutable
String str = "Hello";
str = str + " World";  // Creates new String object

// StringBuilder - Mutable, not thread-safe
StringBuilder sb = new StringBuilder();
sb.append("Hello");
sb.append(" World");
System.out.println(sb.toString());  // "Hello World"

// StringBuffer - Mutable, thread-safe
StringBuffer sbf = new StringBuffer();
sbf.append("Hello");
sbf.append(" World");
System.out.println(sbf.toString());  // "Hello World"

// Comparison - Performance
long start = System.currentTimeMillis();
String s = "";
for (int i = 0; i < 10000; i++) {
    s += i;  // Creates new String each iteration
}
System.out.println("String: " + (System.currentTimeMillis() - start)); //
Slower

start = System.currentTimeMillis();
StringBuilder sb2 = new StringBuilder();
for (int i = 0; i < 10000; i++) {
    sb2.append(i);  // Appends to same object
}
System.out.println("StringBuilder: " + (System.currentTimeMillis() - start));
// Faster
```

Q26. What are the access modifiers in Java and their scope?

| Modifier | Same Class | Same Package | Different Package (Subclass) | Different Package (Other) | |———-|———-|———-|———————————|———————————| | public | ✓ | ✓ | ✓ | ✓ | | protected | ✓ | ✓ | ✓ | ✗ | | default (no modifier) | ✓ | ✓ | ✗ | ✗ | | private | ✓ | ✗ | ✗ | ✗ |

Example:

```java
public class Parent {
    public void publicMethod() { }      // Accessible everywhere
    protected void protectedMethod() { } // Accessible in same package and
subclasses
    void defaultMethod() { }             // Accessible in same package only
    private void privateMethod() { }     // Accessible only in this class
```

```
}

public class Child extends Parent {
    public void test() {
        publicMethod();           // ✓ Accessible
        protectedMethod();        // ✓ Accessible (subclass)
        // defaultMethod();       // ✗ Not accessible (different package)
        // privateMethod();       // ✗ Not accessible
    }
}
```

Q27. Explain interfaces and abstract classes. When to use each?

- **Interfaces**: Contract defining methods that implementing classes must follow. Can contain constants and default methods. ```java public interface WebBrowser { void launchBrowser(); void closeBrowser(); void navigate(String url); }

public class Chrome implements WebBrowser { @Override public void launchBrowser() { System.out.println("Launching Chrome"); } @Override public void closeBrowser() { System.out.println("Closing Chrome"); } @Override public void navigate(String url) { System.out.println("Navigate to" + url); } } ```

- **Abstract Classes**: Partial implementation; can have abstract and concrete methods, constructors, and state.

```
public abstract class Browser {
    public abstract void launchBrowser();

    public void closeBrowser() {
        System.out.println("Closing browser");
    }
}

public class Firefox extends Browser {
    @Override
    public void launchBrowser() { System.out.println("Launching
Firefox"); }
}
```

When to use: - **Interface**: Define behavior/contract; allow unrelated classes to implement same functionality. - **Abstract Class**: Share code among related classes; define common state and behavior.

Q28. What is method overloading and method overriding? Provide examples.

- **Method Overloading**: Multiple methods with same name but different parameters in same class. Resolved at compile-time (static binding). ```java public class Calculator { // Same method name, different parameters public int add(int a, int b) { return a + b; } public double add(double a, double b) { return a + b; } public int add(int a, int b, int c) { return a + b + c; } public String add(String a, String b) { return a.concat(b); } }

Calculator calc = new Calculator(); System.out.println(calc.add(5, 10)); // Calls int version System.out.println(calc.add(5.5, 10.5)); // Calls double version System.out.println(calc.add(5, 10, 15)); // Calls int version with 3 params System.out.println(calc.add("Hello", " World")); // Calls String version ```

- **Method Overriding**: Subclass provides specific implementation of parent class method. Resolved at runtime (dynamic binding).

```java
public class Animal {
    public void sound() { System.out.println("Generic animal sound"); }
}

public class Dog extends Animal {
    @Override
    public void sound() { System.out.println("Bark"); }
}

public class Cat extends Animal {
    @Override
    public void sound() { System.out.println("Meow"); }
}

Animal animal = new Dog();  // Reference of Animal, object of Dog
animal.sound();             // Output: Bark (runtime polymorphism)
```

Q29. What is the "final" keyword and its uses?

The **final** keyword restricts modification:

- **Final Class**: Cannot be extended/subclassed.

```java
public final class ImmutableClass {
    // Cannot be subclassed
}

// This will cause compile error
// public class SubClass extends ImmutableClass { }
```

- **Final Method**: Cannot be overridden.

```
public class Parent {
    public final void criticalMethod() {
        System.out.println("Cannot override");
    }
}

public class Child extends Parent {
    // This will cause compile error
    // public void criticalMethod() { }
}
```

- **Final Variable**: Cannot be reassigned (immutable).

```
public class Configuration {
    public static final String DATABASE_URL =
"jdbc:mysql://localhost:3306/db";
    public static final int CONNECTION_TIMEOUT = 30;

    public final String username = "admin";  // Instance final variable
}

// Cannot reassign
// DATABASE_URL = "new_url";  // Compile error
```

Q30. Explain the concept of pass-by-value and pass-by-reference in Java.

Java is always pass-by-value, but behavior differs for primitives and objects:

- **Primitives**: Value is copied; changes don't affect original.

```
public void modifyPrimitive(int num) {
    num = 100;  // Modifies local copy only
}

int value = 5;
modifyPrimitive(value);
System.out.println(value);  // Still 5 (not affected)
```

- **Objects**: Reference is copied; changes to object state affect original, but reassigning reference doesn't.

```
public class User {
    public String name;
}

public void modifyObject(User user) {
    user.name = "Modified";  // Affects original object
    user = new User();       // Reassignment doesn't affect original
```

```
    reference
    }

    User user = new User();
    user.name = "Original";
    modifyObject(user);
    System.out.println(user.name);  // "Modified" (object state changed)
```

Q31. What are constructors? Explain default, parameterized, and copy constructors.

- **Default Constructor**: No parameters; initializes objects with default values. ```java public class WebDriver { private String browserName;

```
  // Default constructor
  public WebDriver() {
      this.browserName = "Chrome";
  }
```

}

WebDriver driver = new WebDriver(); // Uses default constructor ```

- **Parameterized Constructor**: Accepts parameters; allows custom initialization.

```
public class WebDriver {
    private String browserName;

    // Parameterized constructor
    public WebDriver(String browserName) {
        this.browserName = browserName;
    }
}

WebDriver chromeDriver = new WebDriver("Chrome");
WebDriver firefoxDriver = new WebDriver("Firefox");
```

- **Copy Constructor**: Creates a copy of existing object.

```
public class TestData {
    private String username;
    private String password;

    // Copy constructor
    public TestData(TestData other) {
        this.username = other.username;
        this.password = other.password;
    }
}
```

```
TestData original = new TestData();
original.username = "user";
TestData copy = new TestData(original);  // Copy created
```

Q32. Explain the “this” and “super” keywords.

- **this**: References current object instance; used to access instance variables and methods.
```java public class TestStep { private String stepName; private int stepNumber;

```
  public TestStep(String stepName, int stepNumber) {
      this.stepName = stepName;    // Distinguishes instance variable from
parameter
      this.stepNumber = stepNumber;
  }

  public void execute() {
      System.out.println("Executing: " + this.stepName);  // Reference
current object
  }

  public TestStep cloneStep() {
      return new TestStep(this.stepName, this.stepNumber);  // Pass current
object state
  }
```

}```

- **super**: References parent class; used to call parent methods and constructors.

```
public class BaseTest {
    public void setup() {
        System.out.println("Base setup");
    }
}

public class DerivedTest extends BaseTest {
    @Override
    public void setup() {
        super.setup();  // Call parent method
        System.out.println("Derived setup");
    }

    public DerivedTest() {
        super();  // Call parent constructor
    }
}
```

```

Q33. What is the “instanceof” operator and how is it used?

The **instanceof** operator checks if an object is an instance of a class or interface.

```
public class BrowserFactory {
    public void performBrowserAction(WebDriver driver) {
        if (driver instanceof ChromeDriver) {
            System.out.println("Using Chrome specific features");
            // Chrome-specific code
        } else if (driver instanceof FirefoxDriver) {
            System.out.println("Using Firefox specific features");
            // Firefox-specific code
        } else if (driver instanceof WebDriver) {
            System.out.println("Generic WebDriver action");
        }
    }
}

// Practical example
WebDriver driver = new ChromeDriver();
if (driver instanceof ChromeDriver) {
    ChromeDriver chromeDriver = (ChromeDriver) driver;  // Type casting
    chromeDriver.executeScript("console.log('Chrome specific');");
}

// Check against interfaces
if (driver instanceof TakesScreenshot) {
    TakesScreenshot screenshot = (TakesScreenshot) driver;
    // Take screenshot
}
```

Q34. Explain memory management, garbage collection, and memory leaks in Java.

- **Memory Management**: Java manages memory through Heap (objects) and Stack (references/primitives).

- **Garbage Collection (GC)**: Automatic process that removes unreferenced objects from memory.

  ```
  public class MemoryExample {
      public static void main(String[] args) {
          WebDriver driver = new ChromeDriver();  // Object allocated in
  Heap
          driver = null;                          // Reference removed
          // GC can now collect the object (not immediately, but
  eligible)
      }
  }
  ```

- **Memory Leaks**: Objects that should be garbage collected but aren't.

```java
// Bad - Memory Leak
public class BrowserManager {
    private static List<WebDriver> drivers = new ArrayList<>();

    public static void createDriver() {
        WebDriver driver = new ChromeDriver();
        drivers.add(driver);  // Driver added to static list
    }

    // Drivers never removed from static list - MEMORY LEAK
}

// Good - Prevent memory leak
public class BrowserManager {
    private List<WebDriver> drivers = new ArrayList<>();

    public void closeAllDrivers() {
        for (WebDriver driver : drivers) {
            driver.quit();
        }
        drivers.clear();  // Clear references
    }
}
```

Best Practices: - Set unused references to null - Close resources in finally blocks or try-with-resources - Avoid static collections that grow indefinitely - Use WeakReference for caching if needed

---

Q35. What are wrapper classes and auto-boxing/auto-unboxing?

Wrapper Classes convert primitives to objects.

```java
// Wrapper classes
Integer intObj = 10;          // Boxing (automatic)
int intValue = intObj;        // Unboxing (automatic)

Double doubleObj = 10.5;      // Boxing
double doubleValue = doubleObj; // Unboxing

Boolean boolObj = true;       // Boxing
boolean boolValue = boolObj;  // Unboxing

// Before Java 5 (Manual boxing/unboxing)
Integer num = new Integer(5);
int value = num.intValue();
```

```java
// Auto-boxing in Collections
List<Integer> numbers = new ArrayList<>();
numbers.add(10);               // Auto-boxing: 10 -> new Integer(10)
int firstNum = numbers.get(0); // Auto-unboxing: Integer -> int

// Useful wrapper methods
String num = "123";
int value = Integer.parseInt(num);        // String to int
int value2 = Integer.valueOf(num);        // String to Integer

Double value3 = Double.parseDouble("45.6");
Boolean value4 = Boolean.parseBoolean("true");

// Convert primitives to String
String str1 = String.valueOf(100);
String str2 = Integer.toString(100);
```

Q36. What is the difference between == and .equals() in Java?

- **== operator**: Compares references (memory addresses) for objects; values for primitives. ```java String str1 = new String(“Hello”); String str2 = new String(“Hello”); System.out.println(str1 == str2); // false (different objects) System.out.println(str1.equals(str2)); // true (same content)

int a = 5; int b = 5; System.out.println(a == b); // true (same value) ```

- **.equals() method**: Compares content/values. Can be overridden in classes.

```java
public class TestData {
    private String username;

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        TestData data = (TestData) obj;
        return Objects.equals(username, data.username);
    }
}

TestData data1 = new TestData("user1");
TestData data2 = new TestData("user1");
System.out.println(data1.equals(data2));  // true (if equals() properly
overridden)
```

Q37. Explain the concept of immutability and how to create immutable classes.

Immutable Class: Object state cannot be changed after creation.

```
public final class ImmutableTestData {
    private final String username;
    private final String password;
    private final List<String> tags;

    // Constructor
    public ImmutableTestData(String username, String password, List<String>
tags) {
        this.username = username;
        this.password = password;
        // Defensive copy for mutable fields
        this.tags = new ArrayList<>(tags);
    }

    // Getters only (no setters)
    public String getUsername() { return username; }
    public String getPassword() { return password; }

    // Return copy for mutable fields to prevent external modification
    public List<String> getTags() {
        return new ArrayList<>(tags);
    }

    @Override
    public String toString() {
        return "ImmutableTestData{" +
               "username='" + username + '\'' +
               ", password='" + password + '\'' +
               ", tags=" + tags +
               '}';
    }
}

// Usage
List<String> tags = new ArrayList<>();
tags.add("smoke");
tags.add("regression");

ImmutableTestData testData = new ImmutableTestData("user", "pass", tags);
// Cannot modify testData after creation
```

Key Rules for Immutability: 1. Make class final (prevent inheritance) 2. Make all fields private and final 3. No setters 4. Return copies of mutable fields in getters 5. Initialize mutable fields defensively

---

Q38. What are generics in Java? How do they ensure type safety?

Generics enable type-safe collections and methods by specifying type at compile-time.

```
// Without generics (old way) - Type unsafe
List list = new ArrayList();
list.add("String");
list.add(123);
String value = (String) list.get(1);  // ClassCastException at runtime

// With generics - Type safe
List<String> stringList = new ArrayList<String>();
stringList.add("Hello");
// stringList.add(123);  // Compile error - type mismatch
String value = stringList.get(0);  // No casting needed

// Generic class
public class TestDataContainer<T> {
    private T data;

    public void setData(T data) { this.data = data; }
    public T getData() { return data; }
}

TestDataContainer<String> container1 = new TestDataContainer<>();
container1.setData("test data");
String data1 = container1.getData();  // No casting

TestDataContainer<Integer> container2 = new TestDataContainer<>();
container2.setData(123);
Integer data2 = container2.getData();  // Type safe

// Generic method
public static <T> void printArray(T[] array) {
    for (T element : array) {
        System.out.println(element);
    }
}

String[] strings = {"a", "b", "c"};
Integer[] integers = {1, 2, 3};
printArray(strings);   // Works with String array
printArray(integers);  // Works with Integer array

// Bounded generics
public static <T extends Number> void printNumber(T number) {
    System.out.println(number.doubleValue());
}
```

```
printNumber(123);      // Integer extends Number - OK
printNumber(45.6);     // Double extends Number - OK
// printNumber("test");  // String doesn't extend Number - Compile error
```

Type Safety Benefits: - Compile-time error detection instead of runtime exceptions - Eliminates need for explicit casting - Enables code reusability with type safety - Improves code readability and maintainability

---

Q39. What is the difference between HashMap and Hashtable?

| Feature | HashMap | Hashtable | |———|———|———–| | Thread-Safe | No | Yes (synchronized) | | Performance | Fast | Slower (due to synchronization) | | Null Keys/Values | Allows one null key & multiple null values | Does not allow null keys or values | | Iteration | Fail-fast iterator | Enumeration and Iterator | | Legacy | Modern (introduced in Java 1.2) | Legacy (Java 1.0) |

Example:

```
// HashMap - Not thread-safe
Map<String, String> map = new HashMap<>();
map.put("browser", "Chrome");
map.put("url", "https://example.com");
map.put(null, "nullValue");  // Allowed
map.put("key", null);          // Allowed

// Hashtable - Thread-safe but outdated
Map<String, String> table = new Hashtable<>();
table.put("browser", "Chrome");
// table.put(null, "value");  // NullPointerException

// Thread-safe HashMap alternative
Map<String, String> syncMap = Collections.synchronizedMap(new HashMap<>());
```

Q40. Explain lambda expressions and functional interfaces.

Lambda Expressions: Short syntax for implementing functional interfaces (interfaces with single abstract method).

```
// Functional interface
@FunctionalInterface
public interface TestAction {
    void execute(String parameter);
}

// Before Lambda (Anonymous class)
TestAction action = new TestAction() {
    @Override
```

```
    public void execute(String parameter) {
        System.out.println("Executing: " + parameter);
    }
};

// With Lambda
TestAction action = (parameter) -> System.out.println("Executing: " +
parameter);
action.execute("test");  // Output: Executing: test

// Built-in Functional Interfaces
// 1. Predicate<T> - Tests a condition
Predicate<Integer> isEven = (num) -> num % 2 == 0;
System.out.println(isEven.test(4));  // true

// 2. Function<T, R> - Transforms input to output
Function<String, Integer> stringLength = (str) -> str.length();
System.out.println(stringLength.apply("Hello"));  // 5

// 3. Consumer<T> - Performs action without return
Consumer<String> printUpperCase = (str) ->
System.out.println(str.toUpperCase());
printUpperCase.accept("hello");  // HELLO

// 4. Supplier<T> - Provides value
Supplier<WebDriver> driverSupplier = () -> new ChromeDriver();
WebDriver driver = driverSupplier.get();

// Lambda with Streams
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
numbers.stream()
    .filter(n -> n % 2 == 0)
    .map(n -> n * 2)
    .forEach(n -> System.out.println(n));  // 4, 8

// Method references (shorthand for lambdas)
List<String> browsers = Arrays.asList("Chrome", "Firefox", "Safari");
browsers.forEach(System.out::println);  // Method reference
```

Section 3: Java Collections Framework (15 Questions)

Q41. Explain the Collection hierarchy and the difference between Collection and Collections.

- **Collection**: Interface at top of hierarchy; represents group of objects. Collection (interface) ├── List (ordered, allows duplicates) │ ├── ArrayList │ ├── LinkedList │ └── Vector ├── Set (unique, unordered) │ ├──

```
HashSet    │    ├── TreeSet    │    └── LinkedHashSet    └── Queue (FIFO)
├── PriorityQueue        └── LinkedList
```

- **Collections**: Utility class with static methods for manipulating collections.

```java
import java.util.Collections;

List<Integer> list = new ArrayList<>(Arrays.asList(3, 1, 4, 1, 5));
Collections.sort(list);                    // Sort
Collections.reverse(list);                 // Reverse
Collections.shuffle(list);                 // Shuffle
int maxValue = Collections.max(list);      // Maximum
int minValue = Collections.min(list);      // Minimum

List<String> syncList = Collections.synchronizedList(new
ArrayList<>()); // Thread-safe
Map<String, String> syncMap = Collections.synchronizedMap(new
HashMap<>()); // Thread-safe

List<Integer> immutableList = Collections.unmodifiableList(list); //
Immutable
```

Q42. Explain ArrayList, LinkedList, and Vector. When to use each?

| Feature | ArrayList | LinkedList | Vector | |———|———-|———-|——-| | Internal Structure | Dynamic array | Doubly linked list | Dynamic array (legacy) | | Access Time | O(1) | O(n) | O(1) | | Insertion/Deletion | O(n) | O(1) | O(n) | | Thread-Safe | No | No | Yes (synchronized) | | Use Case | Random access | Frequent insertions/deletions | Legacy (avoid) |

Examples:

```java
// ArrayList - Best for frequent access
List<String> browsers = new ArrayList<>();
browsers.add("Chrome");
browsers.add("Firefox");
browsers.get(0);  // Fast - O(1)
browsers.remove(1);  // Slower - O(n)

// LinkedList - Best for insertions/deletions at beginning or middle
LinkedList<String> queue = new LinkedList<>();
queue.add("first");
queue.add("second");
queue.addFirst("zeroth");  // Fast - O(1)
queue.removeFirst();  // Fast - O(1)
queue.get(0);  // Slower - O(n)

// Vector - Legacy, avoid in new code
```

```
Vector<String> vector = new Vector<>();  // Synchronized but outdated
// Use Collections.synchronizedList(new ArrayList<>()) instead
```

Q43. Explain HashSet, TreeSet, and LinkedHashSet with differences.

| Feature | HashSet | TreeSet | LinkedHashSet | |———|———|———|————-| | Order | No guaranteed order | Sorted order | Insertion order | | Performance | O(1) average | O(log n) | O(1) average | | Thread-Safe | No | No | No | | Null Values | Allows | Does not allow | Allows | | Comparable | No | Yes (elements must be comparable) | No |

Examples:

```
// HashSet - No order guarantee
Set<String> hashSet = new HashSet<>();
hashSet.add("Chrome");
hashSet.add("Firefox");
hashSet.add("Safari");
System.out.println(hashSet);  // Order may vary: [Firefox, Chrome, Safari]

// TreeSet - Sorted order
Set<String> treeSet = new TreeSet<>();
treeSet.add("Chrome");
treeSet.add("Firefox");
treeSet.add("Safari");
System.out.println(treeSet);  // Output: [Chrome, Firefox, Safari] (sorted)

// LinkedHashSet - Insertion order
Set<String> linkedSet = new LinkedHashSet<>();
linkedSet.add("Chrome");
linkedSet.add("Firefox");
linkedSet.add("Safari");
System.out.println(linkedSet);  // Output: [Chrome, Firefox, Safari]
(insertion order)

// TreeSet with custom objects
class TestCase implements Comparable<TestCase> {
    String name;
    int priority;

    @Override
    public int compareTo(TestCase other) {
        return this.priority - other.priority;  // Sort by priority
    }
}

Set<TestCase> testCases = new TreeSet<>();
testCases.add(new TestCase("Login", 1));
```

```
testCases.add(new TestCase("Dashboard", 2));
// Sorted by priority
```

Q44. Explain HashMap, TreeMap, and LinkedHashMap with practical examples.

| Feature | HashMap | TreeMap | LinkedHashMap | |———|———|———|————-| | Order | No guaranteed order | Sorted by key | Insertion order | | Performance | O(1) average | O(log n) | O(1) average | | Thread-Safe | No | No | No | | Null Keys/Values | Allows null key and values | Does not allow null key | Allows null key and values |

Examples:

```
// HashMap - No order guarantee
Map<String, Integer> hashMap = new HashMap<>();
hashMap.put("Chrome", 1);
hashMap.put("Firefox", 2);
hashMap.put("Safari", 3);
System.out.println(hashMap);  // Order may vary

// TreeMap - Sorted by key
Map<String, Integer> treeMap = new TreeMap<>();
treeMap.put("Chrome", 1);
treeMap.put("Firefox", 2);
treeMap.put("Safari", 3);
System.out.println(treeMap);  // Output: {Chrome=1, Firefox=2, Safari=3}
(sorted)

// LinkedHashMap - Insertion order
Map<String, Integer> linkedMap = new LinkedHashMap<>();
linkedMap.put("Chrome", 1);
linkedMap.put("Firefox", 2);
linkedMap.put("Safari", 3);
System.out.println(linkedMap);  // Output: {Chrome=1, Firefox=2, Safari=3}
(insertion order)

// Practical - Test execution order preservation
Map<String, String> testSteps = new LinkedHashMap<>();
testSteps.put("1", "Launch browser");
testSteps.put("2", "Navigate to URL");
testSteps.put("3", "Enter credentials");
testSteps.put("4", "Click login");

for (Map.Entry<String, String> entry : testSteps.entrySet()) {
    System.out.println(entry.getKey() + ": " + entry.getValue());
}  // Prints in insertion order
```

Q45. Explain Comparable and Comparator interfaces with examples.

- **Comparable**: Interface for natural sorting; object sorts itself. ```java public class TestCase implements Comparable { private String name; private int priority;

```
  public TestCase(String name, int priority) {
      this.name = name;
      this.priority = priority;
  }

  @Override
  public int compareTo(TestCase other) {
      // Sort by priority ascending
      return this.priority - other.priority;
  }
```

}

List cases = new ArrayList<>(); cases.add(new TestCase("Login", 3)); cases.add(new TestCase("Dashboard", 1)); cases.add(new TestCase("Logout", 2)); Collections.sort(cases); // Sorted by priority: Dashboard(1), Logout(2), Login(3) ```

- **Comparator**: External comparator for custom sorting; can sort same object in different ways.

```
public class TestCase {
    private String name;
    private int priority;

    // Getters
    public String getName() { return name; }
    public int getPriority() { return priority; }
}

// Sort by priority
Comparator<TestCase> byPriority = (t1, t2) -> t1.getPriority() -
t2.getPriority();

// Sort by name
Comparator<TestCase> byName = (t1, t2) ->
t1.getName().compareTo(t2.getName());

// Sort by priority descending
Comparator<TestCase> byPriorityDesc = (t1, t2) -> t2.getPriority() -
t1.getPriority();

List<TestCase> cases = new ArrayList<>();
cases.add(new TestCase("Login", 3));
cases.add(new TestCase("Dashboard", 1));
```

```
        Collections.sort(cases, byPriority);     // Sort by priority
        Collections.sort(cases, byName);         // Sort by name
        Collections.sort(cases, byPriorityDesc); // Sort by priority descending
```

Key Difference: - **Comparable**: One natural sorting (part of class definition) - **Comparator**: Multiple sorting strategies (external, flexible)

Q46. Explain Iterator and how to remove elements while iterating.

```
// Iterator interface
List<String> browsers = new ArrayList<>();
browsers.add("Chrome");
browsers.add("Firefox");
browsers.add("Safari");

// Traditional iteration
Iterator<String> iterator = browsers.iterator();
while (iterator.hasNext()) {
    String browser = iterator.next();
    System.out.println(browser);
}

// WRONG - Concurrent modification exception
for (String browser : browsers) {
    if (browser.equals("Firefox")) {
        browsers.remove(browser);  // ConcurrentModificationException
    }
}

// CORRECT - Use iterator to remove
Iterator<String> iter = browsers.iterator();
while (iter.hasNext()) {
    String browser = iter.next();
    if (browser.equals("Firefox")) {
        iter.remove();  // Safe removal
    }
}

// Alternative - removeIf (Java 8+)
browsers.removeIf(browser -> browser.equals("Firefox"));

// Alternative - Collect non-matching elements
List<String> result = browsers.stream()
    .filter(browser -> !browser.equals("Firefox"))
    .collect(Collectors.toList());
```

Q47. Explain the fail-fast and fail-safe iterators.

- **Fail-Fast**: Iterator throws ConcurrentModificationException if collection modified during iteration. ```java List list = new ArrayList<>(); list.add("A"); list.add("B"); list.add("C");

Iterator iterator = list.iterator(); while (iterator.hasNext()) { String element = iterator.next(); if (element.equals("B")) { list.remove("B"); // ConcurrentModificationException thrown } } ```

- **Fail-Safe**: Iterator doesn't throw exception if collection modified; works on snapshot/copy.

```
Map<String, String> map = new ConcurrentHashMap<>();
map.put("1", "One");
map.put("2", "Two");
map.put("3", "Three");

Iterator<String> iterator = map.keySet().iterator();
while (iterator.hasNext()) {
    String key = iterator.next();
    map.put("4", "Four");  // No exception - ConcurrentHashMap is fail-
safe
}
```

Collections with these characteristics: - **Fail-Fast**: ArrayList, HashMap, HashSet, LinkedList - **Fail-Safe**: ConcurrentHashMap, CopyOnWriteArrayList

---

Q48. What is the difference between Collection.stream() and Collection.parallelStream()?

- **stream()**: Sequential stream; processes elements one by one. `java List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5); numbers.stream() .filter(n -> n % 2 == 0) .map(n -> n * 2) .forEach(System.out::println); // Process sequentially`

- **parallelStream()**: Parallel stream; processes elements using multiple threads.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
numbers.parallelStream()
    .filter(n -> n % 2 == 0)
    .map(n -> n * 2)
    .forEach(System.out::println);  // Process in parallel
```

Comparison:

```
// Stream (sequential) - Order guaranteed
List<String> testCases = Arrays.asList("Test1", "Test2", "Test3", "Test4",
"Test5");
```

```
testCases.stream().forEach(System.out::println);
// Output: Test1, Test2, Test3, Test4, Test5 (always same order)

// ParallelStream - Order not guaranteed
testCases.parallelStream().forEach(System.out::println);
// Output: Test3, Test1, Test4, Test2, Test5 (order may vary)

// Use forEachOrdered() to maintain order in parallel stream
testCases.parallelStream().forEachOrdered(System.out::println);
// Output: Test1, Test2, Test3, Test4, Test5 (order maintained)

// Performance comparison
List<Integer> largeList = IntStream.range(0,
1000000).boxed().collect(Collectors.toList());

long start = System.currentTimeMillis();
largeList.stream().filter(n -> n % 2 == 0).count();
System.out.println("Stream: " + (System.currentTimeMillis() - start) + "ms");

start = System.currentTimeMillis();
largeList.parallelStream().filter(n -> n % 2 == 0).count();
System.out.println("ParallelStream: " + (System.currentTimeMillis() - start)
+ "ms");
```

When to use: - **stream()**: Small to medium collections, order matters - **parallelStream()**: Large collections, order doesn't matter, CPU-intensive operations

---

Q49. Explain Queue, Deque, and PriorityQueue with examples.

- **Queue**: FIFO (First-In-First-Out) data structure. ```java Queue queue = new LinkedList<>(); queue.add("Step1"); // Add to end queue.add("Step2"); queue.add("Step3");

String first = queue.poll(); // Remove from front: "Step1" String peek = queue.peek(); // View front without removing: "Step2" ```

- **Deque**: Double-ended queue; supports insertion/removal from both ends.

```
Deque<String> deque = new LinkedList<>();
deque.addFirst("First");      // Add to front
deque.addLast("Last");        // Add to end

String front = deque.removeFirst();  // Remove from front
String back = deque.removeLast();    // Remove from end

String peekFirst = deque.getFirst(); // View front
String peekLast = deque.getLast();   // View end
```

- **PriorityQueue**: Elements ordered by priority (natural order or comparator).

```
PriorityQueue<Integer> pQueue = new PriorityQueue<>();
pQueue.add(5);
pQueue.add(3);
pQueue.add(7);
pQueue.add(1);

while (!pQueue.isEmpty()) {
    System.out.println(pQueue.poll());  // Output: 1, 3, 5, 7 (sorted)
}

// PriorityQueue with custom comparator (descending)
PriorityQueue<Integer> descQueue = new PriorityQueue<>((a, b) -> b -
a);
descQueue.add(5);
descQueue.add(3);
descQueue.add(7);
while (!descQueue.isEmpty()) {
    System.out.println(descQueue.poll());  // Output: 7, 5, 3 (reverse
sorted)
}
```

Q50. Explain Map.Entry and how to iterate over Map with different methods.

Map.Entry: Interface representing key-value pair in Map.

```
Map<String, String> testData = new LinkedHashMap<>();
testData.put("username", "testuser");
testData.put("password", "pass123");
testData.put("email", "test@example.com");

// Method 1: entrySet() - Most efficient
for (Map.Entry<String, String> entry : testData.entrySet()) {
    String key = entry.getKey();
    String value = entry.getValue();
    System.out.println(key + " : " + value);
}

// Method 2: keySet()
for (String key : testData.keySet()) {
    String value = testData.get(key);
    System.out.println(key + " : " + value);
}

// Method 3: values()
for (String value : testData.values()) {
    System.out.println(value);
```

```
}

// Method 4: Iterator with entrySet()
Iterator<Map.Entry<String, String>> iterator =
testData.entrySet().iterator();
while (iterator.hasNext()) {
    Map.Entry<String, String> entry = iterator.next();
    System.out.println(entry.getKey() + " : " + entry.getValue());
}

// Method 5: forEach() with lambda
testData.forEach((key, value) -> System.out.println(key + " : " + value));

// Method 6: Stream API
testData.entrySet().stream()
    .forEach(entry -> System.out.println(entry.getKey() + " : " +
entry.getValue()));

// Modify values during iteration
testData.entrySet().stream()
    .filter(entry -> entry.getKey().equals("password"))
    .forEach(entry -> entry.setValue("newpass123"));
```

Q51. What are the differences between fail-fast and fail-safe collections?

Explained in Q47, but expanding for completeness:

```
// Fail-Fast Collections: ArrayList, HashMap, HashSet
List<String> failFastList = new ArrayList<>();
failFastList.add("A");
failFastList.add("B");
failFastList.add("C");

// This will throw ConcurrentModificationException
try {
    for (String element : failFastList) {
        if (element.equals("B")) {
            failFastList.remove(element);  // Modification during iteration
        }
    }
} catch (ConcurrentModificationException e) {
    System.out.println("Caught ConcurrentModificationException");
}

// Fail-Safe Collections: CopyOnWriteArrayList, ConcurrentHashMap
CopyOnWriteArrayList<String> failSafeList = new CopyOnWriteArrayList<>();
failSafeList.add("A");
failSafeList.add("B");
```

```
failSafeList.add("C");

// No exception thrown
for (String element : failSafeList) {
    if (element.equals("B")) {
        failSafeList.remove(element);  // Safe modification during iteration
    }
}

// But note: Fail-safe may use snapshot approach
// Iterator sees snapshot from creation time
ConcurrentHashMap<String, Integer> concMap = new ConcurrentHashMap<>();
concMap.put("1", 100);
concMap.put("2", 200);
concMap.put("3", 300);

Iterator<Integer> iter = concMap.values().iterator();
concMap.put("4", 400);  // Add after iterator creation
while (iter.hasNext()) {
    System.out.println(iter.next());  // May or may not include "4" depending
on implementation
}
```

Q52. Explain Stream API operations: filter, map, flatMap, reduce, collect.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);

// filter() - Keep elements matching condition
List<Integer> evenNumbers = numbers.stream()
    .filter(n -> n % 2 == 0)
    .collect(Collectors.toList());  // [2, 4, 6]

// map() - Transform each element
List<Integer> squared = numbers.stream()
    .map(n -> n * n)
    .collect(Collectors.toList());  // [1, 4, 9, 16, 25, 36]

// flatMap() - Transform and flatten nested collections
List<List<String>> nested = Arrays.asList(
    Arrays.asList("a", "b"),
    Arrays.asList("c", "d"),
    Arrays.asList("e", "f")
);
List<String> flattened = nested.stream()
    .flatMap(List::stream)
    .collect(Collectors.toList());  // [a, b, c, d, e, f]
```

```java
// reduce() - Combine elements into single value
int sum = numbers.stream().reduce(0, (a, b) -> a + b);  // 21
int product = numbers.stream().reduce(1, (a, b) -> a * b);  // 720

// collect() - Gather into collection
List<String> words = Arrays.asList("apple", "banana", "cherry", "apricot");

// Collect to List
List<String> aWords = words.stream()
    .filter(w -> w.startsWith("a"))
    .collect(Collectors.toList());  // [apple, apricot]

// Collect to Set
Set<String> uniqueLengths = words.stream()
    .map(String::length)
    .map(String::valueOf)
    .collect(Collectors.toSet());

// Collect to Map
Map<String, Integer> wordLength = words.stream()
    .collect(Collectors.toMap(
        word -> word,
        String::length
    ));  // {apple=5, banana=6, cherry=6, apricot=7}

// Collect with grouping
Map<Integer, List<String>> groupedByLength = words.stream()
    .collect(Collectors.groupingBy(String::length));
    // {5=[apple], 6=[banana, cherry], 7=[apricot]}

// Chaining operations
List<Integer> result = numbers.stream()
    .filter(n -> n > 2)           // [3, 4, 5, 6]
    .map(n -> n * 10)             // [30, 40, 50, 60]
    .filter(n -> n % 40 != 0)     // [30, 50, 60]
    .collect(Collectors.toList());
```

Q53. What are the differences between array and ArrayList?

| Feature | Array | ArrayList | |———|——-|———–| | Size | Fixed size | Dynamic size | | Type | Can store primitives and objects | Only stores objects | | Performance | Slightly faster | Slightly slower | | Memory | Fixed memory | Grows as needed | | Methods | Limited methods | Rich API (add, remove, etc.) |

Examples:

```java
// Array - Fixed size
String[] browsers = new String[3];
browsers[0] = "Chrome";
browsers[1] = "Firefox";
// browsers[3] = "Safari";  // ArrayIndexOutOfBoundsException

// Cannot dynamically add elements
int length = browsers.length;

// ArrayList - Dynamic size
List<String> browserList = new ArrayList<>();
browserList.add("Chrome");
browserList.add("Firefox");
browserList.add("Safari");  // Dynamically grows
browserList.add("Edge");    // No error

// Rich API
browserList.remove("Firefox");
browserList.contains("Chrome");  // true
browserList.size();              // 3

// Array with primitives
int[] numbers = {1, 2, 3};  // Can store int directly

// ArrayList cannot store primitives directly - uses wrapper classes
List<Integer> numberList = new ArrayList<>();  // Integer, not int
numberList.add(1);   // Auto-boxing
int firstNum = numberList.get(0);  // Auto-unboxing

// Convert array to ArrayList
String[] arr = {"Chrome", "Firefox", "Safari"};
List<String> list = new ArrayList<>(Arrays.asList(arr));
list.add("Edge");  // Now can add

// Convert ArrayList to array
String[] newArr = list.toArray(new String[0]);
```

Q54. Explain the difference between List, Set, and Map.

| Feature | List | Set | Map | |———|——|—–|—–| | Duplicates | Allows | Does not allow | Keys unique, values can duplicate | | Order | Maintains insertion/index order | Unordered (except TreeSet, LinkedHashSet) | Unordered (except TreeMap, LinkedHashMap) | | Access | By index (get(i)) | No index-based access | By key | | Iteration | Ordered iteration | Unordered iteration | Iterate over entries |

Examples:

```java
// List - Ordered collection with duplicates
List<String> list = new ArrayList<>();
list.add("Chrome");
list.add("Firefox");
list.add("Chrome");  // Duplicate allowed
System.out.println(list.get(0));  // Access by index
// Iteration maintains order

// Set - Unique elements only
Set<String> set = new HashSet<>();
set.add("Chrome");
set.add("Firefox");
set.add("Chrome");  // Duplicate ignored
System.out.println(set.size());  // 2 (not 3)
// Cannot access by index

// Map - Key-value pairs
Map<String, String> map = new HashMap<>();
map.put("browser1", "Chrome");
map.put("browser2", "Firefox");
map.put("browser1", "Edge");  // Overwrites previous value
System.out.println(map.get("browser1"));  // "Edge"
System.out.println(map.size());  // 2

// Iteration differences
// List
for (int i = 0; i < list.size(); i++) {
    System.out.println(list.get(i));
}

// Set
for (String element : set) {
    System.out.println(element);
}

// Map
for (Map.Entry<String, String> entry : map.entrySet()) {
    System.out.println(entry.getKey() + " : " + entry.getValue());
}
```

Q55. Explain Collectors class and its common methods.

Collectors: Utility class for collecting stream results into collections.

```java
List<String> browsers = Arrays.asList("Chrome", "Firefox", "Safari", "Edge",
"Chrome");

// toList() - Collect to List
```

```java
List<String> list = browsers.stream().collect(Collectors.toList());

// toSet() - Collect to Set
Set<String> set = browsers.stream().collect(Collectors.toSet());

// toMap() - Collect to Map
Map<String, Integer> lengthMap = browsers.stream()
    .collect(Collectors.toMap(
        b -> b,              // Key
        String::length       // Value
    ));

// groupingBy() - Group by criteria
Map<Integer, List<String>> groupedByLength = browsers.stream()
    .collect(Collectors.groupingBy(String::length));

// partitioningBy() - Partition into true/false groups
Map<Boolean, List<String>> partitioned = browsers.stream()
    .collect(Collectors.partitioningBy(b -> b.length() > 5));

// joining() - Join elements into String
String joined = browsers.stream()
    .collect(Collectors.joining(", "));  // "Chrome, Firefox, Safari, Edge,
Chrome"

// counting() - Count elements
long count = browsers.stream()
    .collect(Collectors.counting());  // 5

// averagingInt() - Average
double avgLength = browsers.stream()
    .collect(Collectors.averagingInt(String::length));

// maxBy() / minBy() - Find max/min
Optional<String> longest = browsers.stream()
    .collect(Collectors.maxBy(Comparator.comparingInt(String::length)));

// summarizingInt() - Get statistics
IntSummaryStatistics stats = browsers.stream()
    .collect(Collectors.summarizingInt(String::length));
System.out.println("Sum: " + stats.getSum());
System.out.println("Average: " + stats.getAverage());
System.out.println("Max: " + stats.getMax());
System.out.println("Min: " + stats.getMin());
System.out.println("Count: " + stats.getCount());
```

Section 4: TestNG Framework (16 Questions)

Q56. What is TestNG and how does it differ from JUnit?

TestNG is a testing framework inspired by JUnit but with more advanced features.

Key Differences: | Feature | TestNG | JUnit | |———|———–|———-| | Annotations | @Test, @BeforeClass, @AfterClass | @Test, @Before, @After | | Test Groups | Supported | Not directly supported | | Parallel Execution | Built-in | Requires plugin | | Parametrization | DataProvider | Parameterized | | Dependency | @Test(dependsOnMethods) | Not supported | | Configuration Methods | Class/Group/Suite level | Limited |

Example:

```
// TestNG test
public class LoginTest {
    @BeforeClass
    public void setUp() {
        System.out.println("Setup before all tests");
    }

    @Test
    public void testLogin() {
        System.out.println("Testing login");
    }

    @AfterClass
    public void tearDown() {
        System.out.println("Cleanup after all tests");
    }
}

// JUnit test
public class LoginTestJUnit {
    @BeforeClass
    public static void setUpClass() {
        System.out.println("Setup before all tests");
    }

    @Test
    public void testLogin() {
        System.out.println("Testing login");
    }

    @AfterClass
    public static void tearDownClass() {
        System.out.println("Cleanup after all tests");
    }
}
```

Q57. Explain TestNG annotations: @BeforeClass, @BeforeMethod, @AfterClass, @AfterMethod.

| Annotation | Executes | Frequency | Use Case | |———–|———-|———–|———-| | @BeforeClass | Once before all methods in class | Once | Setup test environment (driver, database) | | @BeforeMethod | Before each @Test method | For each test | Reset state for each test | | @AfterClass | Once after all methods in class | Once | Cleanup test environment | | @AfterMethod | After each @Test method | For each test | Screenshot on failure, close popups |

Example:

```
public class BrowserTest {
    private WebDriver driver;

    @BeforeClass
    public void setUpBrowser() {
        System.out.println("@BeforeClass - Initializing WebDriver");
        driver = new ChromeDriver();
    }

    @BeforeMethod
    public void navigateHome() {
        System.out.println("@BeforeMethod - Navigating to home");
        driver.get("https://example.com");
    }

    @Test
    public void testLoginSuccess() {
        System.out.println("Test 1 - Login Success");
    }

    @Test
    public void testLoginFailure() {
        System.out.println("Test 2 - Login Failure");
    }

    @AfterMethod
    public void captureScreenshot() {
        System.out.println("@AfterMethod - Capturing screenshot");
    }

    @AfterClass
    public void closeBrowser() {
        System.out.println("@AfterClass - Closing WebDriver");
        driver.quit();
    }
```

```
}

/* Execution Order:
@BeforeClass - Initializing WebDriver
@BeforeMethod - Navigating to home
Test 1 - Login Success
@AfterMethod - Capturing screenshot
@BeforeMethod - Navigating to home
Test 2 - Login Failure
@AfterMethod - Capturing screenshot
@AfterClass - Closing WebDriver
*/
```

Q58. What is @Test annotation and its attributes? Explain invocationCount, timeOut, expectedExceptions.

@Test marks method as test; can include various attributes for behavior control.

```
public class AdvancedTestNGTest {

    // Basic test
    @Test
    public void testBasic() {
        assertTrue(true);
    }

    // invocationCount - Run test multiple times
    @Test(invocationCount = 3)
    public void testMultipleInvocations() {
        System.out.println("Running 3 times");
    }

    // timeOut - Test fails if exceeds time (milliseconds)
    @Test(timeOut = 3000)
    public void testWithTimeout() throws InterruptedException {
        Thread.sleep(2000);  // Will pass
    }

    // expectedExceptions - Test passes if specified exception thrown
    @Test(expectedExceptions = NullPointerException.class)
    public void testExpectedException() {
        String str = null;
        str.length();  // Throws NullPointerException - TEST PASSES
    }

    // description - Test description
    @Test(description = "Verify login functionality")
    public void testLogin() {
```

```
        System.out.println("Testing login");
    }

    // enabled - Disable test without removing it
    @Test(enabled = false)
    public void testDisabled() {
        System.out.println("This test is disabled");
    }

    // alwaysRun - Run even if dependency fails
    @Test
    public void dependencyTest() {
        throw new RuntimeException("Dependency failed");
    }

    @Test(dependsOnMethods = "dependencyTest", alwaysRun = true)
    public void testAlwaysRun() {
        System.out.println("Runs even if dependency failed");
    }

    // priority - Control test execution order
    @Test(priority = 2)
    public void testSecond() {
        System.out.println("Second");
    }

    @Test(priority = 1)
    public void testFirst() {
        System.out.println("First");
    }
}
```

Q59. Explain TestNG assertions and how they differ from standard assertions.

TestNG Assertions provide more detailed failure messages and better reporting.

```
import org.testng.Assert;

public class AssertionTest {

    // assertEquals - Check equality
    @Test
    public void testAssertEquals() {
        String actual = "Chrome";
        String expected = "Chrome";
        Assert.assertEquals(actual, expected, "Browser should be Chrome");
    }
```

```
    // assertTrue / assertFalse
    @Test
    public void testBooleanAssertions() {
        boolean loginSuccessful = true;
        Assert.assertTrue(loginSuccessful, "Login should be successful");

        boolean errorMessage = false;
        Assert.assertFalse(errorMessage, "Error message should not appear");
    }

    // assertNull / assertNotNull
    @Test
    public void testNullAssertions() {
        String result = null;
        Assert.assertNull(result, "Result should be null");

        String actualResult = "Success";
        Assert.assertNotNull(actualResult, "Result should not be null");
    }

    // assertSame / assertNotSame - Reference equality
    @Test
    public void testReferenceEquality() {
        WebDriver driver1 = new ChromeDriver();
        WebDriver driver2 = driver1;
        Assert.assertSame(driver1, driver2, "Should be same object");

        WebDriver driver3 = new ChromeDriver();
        Assert.assertNotSame(driver1, driver3, "Should be different
objects");
    }

    // fail - Explicitly fail test
    @Test
    public void testFail() {
        boolean result = performAction();
        if (!result) {
            Assert.fail("Action should have succeeded");
        }
    }

    // Soft assertions - Don't stop on first failure
    @Test
    public void testSoftAssertions() {
        SoftAssert softAssert = new SoftAssert();

        softAssert.assertEquals("Chrome", "Firefox", "Browser mismatch");  //
Fails but continues
        softAssert.assertTrue(false, "Condition failed");
```

```
// Fails but continues
        softAssert.assertNotNull("value", "Value is null");
// Passes

        softAssert.assertAll();  // Report all failures together
    }

    private boolean performAction() {
        return true;
    }
}
```

Q60. What is DataProvider in TestNG and how to use it for parameterization?

DataProvider passes multiple sets of test data to same test method.

```
public class DataProviderTest {

    // Simple DataProvider
    @DataProvider(name = "loginData")
    public Object[][] getLoginData() {
        return new Object[][] {
            {"user1", "pass1", true},
            {"user2", "pass2", true},
            {"", "", false},
            {"user3", "wrong", false}
        };
    }

    @Test(dataProvider = "loginData")
    public void testLogin(String username, String password, boolean expected)
{
        boolean result = performLogin(username, password);
        Assert.assertEquals(result, expected, "Login result should match");
    }

    // DataProvider with Map
    @DataProvider(name = "browserData")
    public Object[][] getBrowserData() {
        return new Object[][] {
            {"Chrome", "Google Chrome"},
            {"Firefox", "Mozilla Firefox"},
            {"Safari", "Apple Safari"}
        };
    }

    @Test(dataProvider = "browserData")
    public void testBrowserCompatibility(String browserName, String
```

```
browserFullName) {
        System.out.println("Testing: " + browserName + " (" + browserFullName
+ ")");
    }

    // DataProvider from external source (CSV, JSON, Database)
    @DataProvider(name = "testDataFromCSV")
    public Object[][] getTestDataFromCSV() {
        // Read from CSV file
        List<String[]> data = readCSVFile("testdata.csv");
        return data.toArray(new Object[0][]);
    }

    @Test(dataProvider = "testDataFromCSV")
    public void testWithExternalData(String... testData) {
        System.out.println("Test dat " + Arrays.toString(testData));
    }

    // DataProvider with Iterator (Memory efficient for large data)
    @DataProvider(name = "largeDataSet")
    public Iterator<Object[]> getLargeDataSet() {
        List<Object[]> data = new ArrayList<>();
        for (int i = 0; i < 1000; i++) {
            data.add(new Object[]{"User" + i, "Pass" + i, true});
        }
        return data.iterator();
    }

    @Test(dataProvider = "largeDataSet")
    public void testWithLargeDataSet(String user, String pass, boolean
expected) {
        // Test executed 1000 times with different data
    }

    private boolean performLogin(String username, String password) {
        return !username.isEmpty() && !password.isEmpty();
    }

    private List<String[]> readCSVFile(String fileName) {
        // Implementation to read CSV file
        return new ArrayList<>();
    }
}
```

Q61. What is TestNG XML configuration file and how to run tests?

testng.xml controls test execution, grouping, and configuration.

testng.xml Structure:

```xml
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Automation Test Suite" parallel="tests" thread-count="2">

    <!-- Parameters -->
    <parameter name="browser" value="Chrome"/>
    <parameter name="baseUrl" value="https://example.com"/>

    <!-- Test Group 1 -->
    <test name="Smoke Tests">
        <groups>
            <run>
                <include name="smoke"/>
            </run>
        </groups>
        <classes>
            <class name="com.automation.tests.LoginTest"/>
            <class name="com.automation.tests.DashboardTest"/>
        </classes>
    </test>

    <!-- Test Group 2 -->
    <test name="Regression Tests">
        <groups>
            <run>
                <include name="regression"/>
                <exclude name="skip"/>
            </run>
        </groups>
        <classes>
            <class name="com.automation.tests.AdvancedTest"/>
        </classes>
    </test>

    <!-- Run specific methods -->
    <test name="Specific Methods">
        <classes>
            <class name="com.automation.tests.CustomTest">
                <methods>
                    <include name="testMethod1"/>
                    <include name="testMethod2"/>
                    <exclude name="testMethod3"/>
                </methods>
            </class>
        </classes>
    </test>

</suite>
```

Running Tests:

```
# Run from command line
testng testng.xml

# Run from Maven
mvn clean test

# Run specific suite
mvn clean test -Dsuite=testng.xml

# Run with parameters
mvn clean test -Dbrowser=Firefox -DbaseUrl=https://staging.example.com
```

Java Code:

```
// Run testng.xml programmatically
TestNG testng = new TestNG();
testng.setTestSuites(Arrays.asList("testng.xml"));
testng.run();

// Run with listeners
TestNG testng = new TestNG();
testng.addListener(new TestNGListener());
testng.setTestSuites(Arrays.asList("testng.xml"));
testng.run();
```

Q62. Explain TestNG Listeners and how to implement custom listeners.

Listeners intercept TestNG events (test start, pass, fail, etc.).

```
// Common Listeners
import org.testng.ITestListener;
import org.testng.ITestResult;

public class TestNGListener implements ITestListener {

    @Override
    public void onStart(ITestContext context) {
        System.out.println("Test Suite Started: " + context.getName());
    }

    @Override
    public void onFinish(ITestContext context) {
        System.out.println("Test Suite Finished: " + context.getName());
        System.out.println("Total Tests: " +
context.getAllTestMethods().length);
    }
```

```
    @Override
    public void onTestStart(ITestResult result) {
        System.out.println("Test Started: " + result.getName());
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        System.out.println("✓ Test Passed: " + result.getName());
    }

    @Override
    public void onTestFailure(ITestResult result) {
        System.out.println("✗ Test Failed: " + result.getName());
        System.out.println("Error: " + result.getThrowable().getMessage());

        // Take screenshot on failure
        takeScreenshot(result.getMethod().getMethodName());
    }

    @Override
    public void onTestSkipped(ITestResult result) {
        System.out.println("⊘ Test Skipped: " + result.getName());
    }

    private void takeScreenshot(String testName) {
        // Screenshot implementation
        System.out.println("Screenshot saved for: " + testName);
    }
}

// Another useful listener
import org.testng.ISuite;
import org.testng.ISuiteListener;

public class SuiteListener implements ISuiteListener {

    @Override
    public void onStart(ISuite suite) {
        System.out.println("Suite Execution Started: " + suite.getName());
    }

    @Override
    public void onFinish(ISuite suite) {
        System.out.println("Suite Execution Finished: " + suite.getName());
        Map<String, ISuiteResult> results = suite.getResults();
        for (ISuiteResult result : results.values()) {
            System.out.println("Tests run: " +
result.getTestContext().getAllTestMethods().length);
```

```
        }
    }
}

// Register listener in testng.xml
/*
<suite>
    <listeners>
        <listener class-name="com.automation.listeners.TestNGListener"/>
        <listener class-name="com.automation.listeners.SuiteListener"/>
    </listeners>
</suite>
*/

// Or register programmatically
TestNG testng = new TestNG();
testng.addListener(new TestNGListener());
testng.addListener(new SuiteListener());
```

Q63. Explain test grouping and how to run specific test groups.

Groups organize tests by category; allows selective test execution.

```
public class GroupedTests {

    @Test(groups = "smoke")
    public void testLoginSmoke() {
        System.out.println("Smoke: Login Test");
    }

    @Test(groups = "smoke")
    public void testDashboardSmoke() {
        System.out.println("Smoke: Dashboard Test");
    }

    @Test(groups = "regression")
    public void testAdvancedSearch() {
        System.out.println("Regression: Advanced Search");
    }

    @Test(groups = {"regression", "important"})
    public void testDataValidation() {
        System.out.println("Regression + Important: Data Validation");
    }

    @Test(groups = "regression")
    public void testErrorHandling() {
        System.out.println("Regression: Error Handling");
```

```
    }

    @Test(groups = {"sanity", "important"})
    public void testCriticalFlow() {
        System.out.println("Sanity + Important: Critical Flow");
    }
}

// testng.xml - Run specific groups
/*
<suite name="Group Test Suite">
    <test name="Smoke Tests">
        <groups>
            <run>
                <include name="smoke"/>
            </run>
        </groups>
        <classes>
            <class name="com.automation.tests.GroupedTests"/>
        </classes>
    </test>

    <test name="Regression Tests">
        <groups>
            <run>
                <include name="regression"/>
            </run>
        </groups>
        <classes>
            <class name="com.automation.tests.GroupedTests"/>
        </classes>
    </test>

    <test name="Important Tests">
        <groups>
            <run>
                <include name="important"/>
                <exclude name="skip"/>
            </run>
        </groups>
        <classes>
            <class name="com.automation.tests.GroupedTests"/>
        </classes>
    </test>

    <test name="Smoke OR Regression">
        <groups>
            <run>
                <include name="smoke"/>
```

```
                <include name="regression"/>
            </run>
        </groups>
        <classes>
            <class name="com.automation.tests.GroupedTests"/>
        </classes>
    </test>
</suite>
*/

// Run from command line
// mvn clean test -Dgroups=smoke
// mvn clean test -Dgroups=smoke,regression
```

Q64. Explain parallel execution in TestNG and configure it.

Parallel Execution runs tests simultaneously on multiple threads.

```
// testng.xml - Parallel execution configuration
/*
<suite name="Parallel Suite" parallel="tests" thread-count="3">
    <!-- Parallel execution at test level -->
    <test name="Test 1" parallel="classes" thread-count="2">
        <classes>
            <class name="com.automation.tests.LoginTest"/>
            <class name="com.automation.tests.DashboardTest"/>
        </classes>
    </test>

    <test name="Test 2">
        <classes>
            <class name="com.automation.tests.SearchTest"/>
        </classes>
    </test>
</suite>

Parallel options:
- parallel="tests" - Tests run in parallel
- parallel="classes" - Classes within test run in parallel
- parallel="methods" - Methods run in parallel
- parallel="instances" - Test instances run in parallel
- thread-count - Number of threads
*/

// Java configuration
public class ParallelTest {

    // Mark test method as thread-safe for parallel execution
```

```
    @Test(threadPoolSize = 3, invocationCount = 9)
    public void testParallelExecution() {
        System.out.println("Test running on thread: " +
            Thread.currentThread().getId());
    }

    // Another approach
    @Test(threadPoolSize = 5, invocationCount = 20)
    public void testMultiThreaded() {
        System.out.println("Running in parallel - Thread: " +
            Thread.currentThread().getName());
    }
}

// Maven POM configuration
/*
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.0.0-M5</version>
            <configuration>
                <suiteXmlFiles>
                    <suiteXmlFile>testng.xml</suiteXmlFile>
                </suiteXmlFiles>
                <parallel>methods</parallel>
                <threadCount>3</threadCount>
            </configuration>
        </plugin>
    </plugins>
</build>
*/

// Execute tests
// mvn clean test -DparallelThreadCount=4
```

Q65. Explain test dependency in TestNG: @Test(dependsOnMethods).

Test Dependency controls execution order; test depends on another's success/failure.

```
public class DependencyTest {

    @Test
    public void testStep1() {
        System.out.println("Step 1: Setup");
        Assert.assertTrue(true);
    }
```

```java
    @Test(dependsOnMethods = "testStep1")
    public void testStep2() {
        System.out.println("Step 2: Execute (depends on Step1)");
        // Only runs if testStep1 passes
    }

    @Test(dependsOnMethods = "testStep2")
    public void testStep3() {
        System.out.println("Step 3: Verify (depends on Step2)");
        // Only runs if testStep2 passes
    }

    // Multiple dependencies
    @Test
    public void testPrerequisite1() {
        System.out.println("Prerequisite 1");
    }

    @Test
    public void testPrerequisite2() {
        System.out.println("Prerequisite 2");
    }

    @Test(dependsOnMethods = {"testPrerequisite1", "testPrerequisite2"})
    public void testWithMultipleDependencies() {
        System.out.println("Depends on multiple tests");
        // Runs only if both prerequisites pass
    }

    // Dependency with alwaysRun
    @Test
    public void testFailingDependency() {
        System.out.println("This will fail");
        Assert.assertTrue(false);
    }

    @Test(dependsOnMethods = "testFailingDependency")
    public void testNormal() {
        System.out.println("Skipped because dependency failed");
    }

    @Test(dependsOnMethods = "testFailingDependency", alwaysRun = true)
    public void testAlwaysRun() {
        System.out.println("Runs even though dependency failed");
    }
}

// Execution flow
```

```
/*
@BeforeClass
testStep1()                             // Executes, passes
testStep2()                             // Executes, passes (depends on Step1)
testStep3()                             // Executes, passes (depends on Step2)
testPrerequisite1()                     // Executes
testPrerequisite2()                     // Executes
testWithMultipleDependencies()          // Executes (all dependencies met)
testFailingDependency()                 // Executes, FAILS
testNormal()                            // SKIPPED (dependency failed)
testAlwaysRun()                         // Executes (alwaysRun=true)
@AfterClass
*/
```

Q66. How to generate TestNG reports and integrate with reporting tools?

```
// Basic TestNG reporting
public class ReportingTest {
    @Test
    public void testWithReporting() {
        System.out.println("Test with built-in reporting");
    }
}

// testng.xml with listeners for reporting
/*
<suite>
    <listeners>
        <listener class-name="org.testng.reporters.FailedReporter"/>
        <listener class-name="org.testng.reporters.EmailableReporter"/>
    </listeners>
</suite>
*/

// ExtentReports integration (Popular reporting tool)
public class ExtentReportListener implements ITestListener {
    private ExtentReports extentReports;
    private ExtentTest extentTest;

    public ExtentReportListener() {
        extentReports = new ExtentReports();
        extentReports.attachReporter(
            new ExtentSparkReporter("target/ExtentReport.html")
        );
    }
```

```java
    @Override
    public void onStart(ITestContext context) {
        System.out.println("Report started");
    }

    @Override
    public void onTestStart(ITestResult result) {
        extentTest = extentReports.createTest(result.getName());
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        extentTest.pass("Test passed");
    }

    @Override
    public void onTestFailure(ITestResult result) {
        extentTest.fail(result.getThrowable());
        // Attach screenshot
        extentTest.addScreenCaptureFromPath("screenshot_path");
    }

    @Override
    public void onTestSkipped(ITestResult result) {
        extentTest.skip("Test skipped");
    }

    @Override
    public void onFinish(ITestContext context) {
        extentReports.flush();
    }
}

// Allure reporting integration
public class AllureReportTest {
    @Test
    @Description("Test login functionality")
    @Severity(SeverityLevel.CRITICAL)
    public void testLoginWithAllure() {
        Step("Login with valid credentials", () -> {
            System.out.println("Logging in");
        });

        Step("Verify dashboard", () -> {
            System.out.println("Dashboard verified");
        });
    }
}
```

```
// POM configuration for reporting
/*
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <suiteXmlFiles>
            <suiteXmlFile>testng.xml</suiteXmlFile>
        </suiteXmlFiles>
        <reportFormat>html</reportFormat>
    </configuration>
</plugin>
*/

// Generate and view reports
// mvn clean test
// Reports generated in: target/surefire-reports/
// Open HTML report: target/surefire-reports/index.html
```

Q67. Explain soft assertions and hard assertions in TestNG.

- **Hard Assertion**: Test stops at first failure; throws exception immediately. `java @Test public void testHardAssertions() { String actual = "Chrome"; Assert.assertEquals(actual, "Firefox"); // Test fails and stops Assert.assertTrue(true); // This line never executes }`

- **Soft Assertion**: Test continues even after assertion failure; collects all failures.

```
@Test
public void testSoftAssertions() {
    SoftAssert softAssert = new SoftAssert();

    softAssert.assertEquals("Chrome", "Firefox");  // Fails but
continues
    softAssert.assertTrue(false);                  // Fails but
continues
    softAssert.assertNotNull("value");             // Passes
    softAssert.assertEquals(5, 5);                 // Passes

    // Report all failures at end
    softAssert.assertAll();  // If any assertion failed, test fails
here
}

// Output if soft assertions fail:
// 1. expected [Firefox] but found [Chrome]
// 2. condition evaluated to false
// Then test fails with all failures combined
```

Practical Example:

```
public class SoftAssertionExample {

    @Test
    public void testUserProfile() {
        SoftAssert softAssert = new SoftAssert();

        // Verify user details without stopping on first failure
        String actualName = getUserName();
        String actualEmail = getUserEmail();
        int actualAge = getUserAge();

        softAssert.assertEquals(actualName, "John Doe", "Name mismatch");
        softAssert.assertEquals(actualEmail, "john@example.com", "Email
mismatch");
        softAssert.assertEquals(actualAge, 30, "Age mismatch");

        // All comparisons execute, failures are collected
        // Test status determined at this point
        softAssert.assertAll();
    }
}
```

Q68. What are the attributes of @DataProvider annotation?

```
public class DataProviderAttributesTest {

    // Basic DataProvider
    @DataProvider(name = "simpleData")
    public Object[][] getSimpleData() {
        return new Object[][]{
            {"user1", "pass1"},
            {"user2", "pass2"}
        };
    }

    @Test(dataProvider = "simpleData")
    public void testWithSimpleData(String user, String pass) {
        System.out.println("User: " + user + ", Pass: " + pass);
    }

    // DataProvider with parallel execution
    @DataProvider(name = "parallelData", parallel = true)
    public Object[][] getParallelData() {
        return new Object[][]{
```

```
            {1},
            {2},
            {3},
            {4}
        };
    }

    @Test(dataProvider = "parallelData", threadPoolSize = 2)
    public void testWithParallelData(int value) {
        System.out.println("Value: " + value + ", Thread: " +
            Thread.currentThread().getId());
    }

    // DataProvider with ITestContext (provides test context)
    @DataProvider(name = "contextData")
    public Object[][] getContextData(ITestContext context) {
        String param = context.getCurrentXmlTest()
            .getParameter("testParam");
        return new Object[][]{
            {param + "_1"},
            {param + "_2"}
        };
    }

    @Test(dataProvider = "contextData")
    public void testWithContext(String data) {
        System.out.println("Context dat " + data);
    }

    // DataProvider with ITestMethod (provides method info)
    @DataProvider(name = "methodData")
    public Object[][] getMethodData(ITestMethod method) {
        System.out.println("Method: " + method.getMethodName());
        return new Object[][]{
            {"data1"},
            {"data2"}
        };
    }

    @Test(dataProvider = "methodData")
    public void testWithMethodInfo(String data) {
        System.out.println("Dat " + data);
    }

    // Attributes summary:
    // name - Unique name for DataProvider
    // parallel - true/false for parallel data provision
    // ITestContext - Get test context information
```

```
    // ITestMethod - Get method information
}
```

Q69. How to skip tests in TestNG? Explain @Test(enabled = false) and SkipException.

```
public class SkipTestExample {

    // Method 1: enabled = false - Test is skipped during execution
    @Test(enabled = false)
    public void testDisabledByDefault() {
        System.out.println("This test is disabled");
    }

    // Method 2: Conditional skip based on environment
    @Test
    public void testConditionalSkip() {
        String environment = System.getProperty("env", "dev");

        if ("dev".equals(environment)) {
            throw new SkipException("Skipping in dev environment");
        }

        System.out.println("Test executed in production");
    }

    // Method 3: Skip in @BeforeMethod
    @BeforeMethod
    public void checkPreconditions(Method method) {
        if (method.getName().equals("testRequiringDatabase")) {
            if (!isDatabaseAvailable()) {
                throw new SkipException("Database not available");
            }
        }
    }

    @Test
    public void testRequiringDatabase() {
        System.out.println("Database test executed");
    }

    // Method 4: Conditional based on OS
    @Test
    public void testWindowsOnly() {
        String osName = System.getProperty("os.name").toLowerCase();

        if (!osName.contains("win")) {
```

```
            throw new SkipException("Windows only test - skipped on " +
osName);
        }

        System.out.println("Windows specific test");
    }

    // Method 5: Skip with ITestResult
    @Test
    public void testWithDynamicSkip(ITestResult result) {
        boolean shouldSkip = checkFeatureFlag("newFeature");

        if (shouldSkip) {
            result.setStatus(ITestResult.SKIP);
            throw new SkipException("Feature not enabled");
        }

        System.out.println("Feature enabled, test runs");
    }

    private boolean isDatabaseAvailable() {
        return false;  // Simulate database unavailable
    }

    private boolean checkFeatureFlag(String feature) {
        return false;  // Simulate feature disabled
    }
}

// testng.xml - Skip tests
/*
<suite>
    <test name="Selective Tests">
        <classes>
            <class name="com.automation.tests.SkipTestExample">
                <methods>
                    <exclude name="testDisabledByDefault"/>
                    <exclude name="testWindowsOnly"/>
                </methods>
            </class>
        </classes>
    </test>
</suite>
*/

// Execution output:
/*
testDisabledByDefault ..... SKIPPED (enabled = false)
testConditionalSkip ..... PASSED (no skip exception)
```

```
testRequiringDatabase ..... SKIPPED (skip exception)
testWindowsOnly ..... SKIPPED (skip exception)
testWithDynamicSkip ..... SKIPPED (skip exception)
*/
```

Q70. Explain @BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest annotations.

| Annotation | Scope | Executes | |———–|——-|———-| | @BeforeSuite | Entire Suite | Once before all tests in suite | | @AfterSuite | Entire Suite | Once after all tests in suite | | @BeforeTest | Test tag in XML | Before each tag | | @AfterTest | Test tag in XML | After each tag |

Example:

```java
public class LifecycleTest {

    @BeforeSuite
    public void setupSuite() {
        System.out.println("@BeforeSuite - Suite initialization");
        // Initialize database connection
        // Load configuration
        // Setup test environment
    }

    @AfterSuite
    public void teardownSuite() {
        System.out.println("@AfterSuite - Suite cleanup");
        // Close database connection
        // Generate reports
        // Send email notifications
    }

    @BeforeTest
    public void setupTest() {
        System.out.println("@BeforeTest - Test initialization");
        // Initialize test-specific resources
    }

    @AfterTest
    public void teardownTest() {
        System.out.println("@AfterTest - Test cleanup");
        // Clean test-specific resources
    }

    @BeforeClass
    public void setupClass() {
        System.out.println("@BeforeClass - Class initialization");
    }
```

```
    @AfterClass
    public void teardownClass() {
        System.out.println("@AfterClass - Class cleanup");
    }

    @BeforeMethod
    public void setupMethod() {
        System.out.println("@BeforeMethod - Method initialization");
    }

    @AfterMethod
    public void teardownMethod() {
        System.out.println("@AfterMethod - Method cleanup");
    }

    @Test
    public void testMethod1() {
        System.out.println("TestMethod 1");
    }

    @Test
    public void testMethod2() {
        System.out.println("TestMethod 2");
    }
}

// testng.xml
/*
<suite name="Complete Lifecycle Suite">
    <test name="Test 1">
        <classes>
            <class name="com.automation.tests.LifecycleTest"/>
        </classes>
    </test>

    <test name="Test 2">
        <classes>
            <class name="com.automation.tests.LifecycleTest"/>
        </classes>
    </test>
</suite>
*/

// Execution order:
/*
@BeforeSuite

    @BeforeTest (Test 1)
```

```
        @BeforeClass
            @BeforeMethod
            TestMethod 1
            @AfterMethod

            @BeforeMethod
            TestMethod 2
            @AfterMethod
        @AfterClass
    @AfterTest (Test 1)

    @BeforeTest (Test 2)
        @BeforeClass
            @BeforeMethod
            TestMethod 1
            @AfterMethod

            @BeforeMethod
            TestMethod 2
            @AfterMethod
        @AfterClass
    @AfterTest (Test 2)

@AfterSuite
*/
```

Q71. What are attributes of @Test annotation? Explain priority, groups, and alwaysRun.

```java
public class TestAnnotationAttributesTest {

    // priority - Control execution order
    @Test(priority = 3)
    public void testThird() {
        System.out.println("Executes third");
    }

    @Test(priority = 1)
    public void testFirst() {
        System.out.println("Executes first");
    }

    @Test(priority = 2)
    public void testSecond() {
        System.out.println("Executes second");
    }
```

```java
@Test(priority = 0)  // Default if not specified
public void testDefaultPriority() {
    System.out.println("Default priority");
}

// groups - Categorize tests
@Test(groups = "smoke")
public void testLoginSmoke() {
    System.out.println("Smoke: Login");
}

@Test(groups = "regression")
public void testAdvancedFeature() {
    System.out.println("Regression: Advanced");
}

@Test(groups = {"smoke", "critical"})
public void testMultipleGroups() {
    System.out.println("Part of multiple groups");
}

// alwaysRun - Execute even if dependency fails
@Test
public void testDependency() {
    Assert.assertTrue(false);  // Fails
}

@Test(dependsOnMethods = "testDependency")
public void testNormalDependent() {
    // SKIPPED - dependency failed
}

@Test(dependsOnMethods = "testDependency", alwaysRun = true)
public void testAlwaysRunDependent() {
    // EXECUTED - even though dependency failed
}

// Combining attributes
@Test(
    priority = 1,
    groups = {"smoke", "critical"},
    timeOut = 5000,
    dependsOnMethods = "testSetup",
    alwaysRun = true
)
public void testComplexAttributes() {
    System.out.println("Complex test with multiple attributes");
}
```

```java
    @Test(description = "Setup for dependency test")
    public void testSetup() {
        System.out.println("Setup test");
    }
}

// Execution order (by priority, then by declaration):
/*
testDefaultPriority
testFirst
testSecond
testThird
testDependency (fails)
testNormalDependent (SKIPPED)
testAlwaysRunDependent (EXECUTED)
*/
```

Section 5: Cucumber BDD (16 Questions)

Q72. What is Cucumber and BDD? Explain Gherkin syntax.

Cucumber is a BDD (Behavior-Driven Development) tool that uses plain English to write test scenarios.

Gherkin Syntax:

```
Feature: User Authentication
  Description of the feature

  Background:
    Given the user is on login page

  Scenario: Successful login with valid credentials
    Given user enters username "testuser"
    And user enters password "password123"
    When user clicks login button
    Then user should see dashboard
    And user should see welcome message

  Scenario: Failed login with invalid credentials
    Given user enters username "invalid"
    And user enters password "wrong"
    When user clicks login button
    Then user should see error message
    And error message should contain "Invalid credentials"

  Scenario Outline: Login with multiple users
    Given user enters username "<username>"
```

```
    And user enters password "<password>"
    When user clicks login button
    Then user should see "<result>"

    Examples:
      | username | password | result |
      | user1 | pass1 | dashboard |
      | user2 | pass2 | dashboard |
      | invalid | wrong | error |
```

Gherkin Keywords: - **Feature**: Describes functionality to test - **Scenario**: Single test case - **Given**: Precondition (initial state) - **When**: Action or event - **Then**: Expected outcome - **And/But**: Additional steps - **Background**: Common steps for all scenarios - **Scenario Outline**: Template for multiple data sets - **Examples**: Data for Scenario Outline

Advantages: - Non-technical stakeholders understand tests - Living documentation - Focuses on behavior, not implementation - Facilitates communication

---

Q73. Explain step definitions and how to create them.

Step Definitions are Java methods that map Gherkin steps to code.

```
// Feature file: login.feature
/*
Feature: Login Functionality
  Scenario: User login with valid credentials
    Given user is on login page
    When user enters username "testuser"
    And user enters password "password123"
    Then user should see dashboard
*/

// Step definitions
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;

public class LoginStepDefinitions {
    private WebDriver driver;
    private LoginPage loginPage;
    private DashboardPage dashboardPage;

    public LoginStepDefinitions() {
        this.driver = DriverManager.getDriver();
        this.loginPage = new LoginPage(driver);
    }

    // @Given - Setup/precondition step
```

```java
    @Given("user is on login page")
    public void userIsOnLoginPage() {
        driver.get("https://example.com/login");
        loginPage.verifyLoginPageLoaded();
    }

    // @When - Action step
    @When("user enters username {string}")
    public void userEntersUsername(String username) {
        loginPage.enterUsername(username);
    }

    @When("user enters password {string}")
    public void userEntersPassword(String password) {
        loginPage.enterPassword(password);
    }

    // @Then - Verification/assertion step
    @Then("user should see dashboard")
    public void userShouldSeeDashboard() {
        dashboardPage = loginPage.clickLoginButton();
        dashboardPage.verifyDashboardLoaded();
    }
}

// Using parameters
public class ParameterStepDefinitions {

    // String parameter
    @Given("user has {string} items")
    public void userHasItems(String quantity) {
        int count = Integer.parseInt(quantity);
        System.out.println("Items count: " + count);
    }

    // Integer parameter
    @When("user clicks button {int}")
    public void userClicksButton(int buttonNumber) {
        System.out.println("Clicking button: " + buttonNumber);
    }

    // Regular expression
    @When("user enters {word} in {word} field")
    public void userEntersData(String data, String field) {
        System.out.println("Entering " + data + " in " + field);
    }

    // Float parameter
    @Given("user has {float} amount")
```

```
    public void userHasAmount(float amount) {
        System.out.println("Amount: " + amount);
    }
}

// Data table parameter
public class DataTableStepDefinitions {

    @Given("user has following dat")
    public void userHasData(DataTable dataTable) {
        List<Map<String, String>> data = dataTable.asMaps(String.class,
String.class);
        for (Map<String, String> row : data) {
            System.out.println("Name: " + row.get("name") + ", Age: " +
row.get("age"));
        }
    }

    @Given("user has following browsers:")
    public void userHasBrowsers(List<String> browsers) {
        for (String browser : browsers) {
            System.out.println("Browser: " + browser);
        }
    }
}
```

Q74. What are Hooks in Cucumber? Explain @Before and @After.

Hooks are code blocks that execute before/after each scenario without being tied to specific steps.

```
import io.cucumber.java.Before;
import io.cucumber.java.After;
import io.cucumber.java.Scenario;

public class Hooks {
    private WebDriver driver;

    // @Before - Executes before each scenario
    @Before
    public void setupBrowser() {
        System.out.println("Starting browser");
        driver = new ChromeDriver();
        DriverManager.setDriver(driver);
    }

    // @After - Executes after each scenario
    @After
```

```java
    public void closeBrowser(Scenario scenario) {
        System.out.println("Closing browser");

        // Take screenshot if scenario failed
        if (scenario.isFailed()) {
            byte[] screenshot = ((TakesScreenshot) driver)
                .getScreenshotAs(OutputType.BYTES);
            scenario.attach(screenshot, "image/png", "Failure Screenshot");
        }

        driver.quit();
    }

    // Conditional hooks - Execute for specific tags
    @Before("@database")
    public void setupDatabase() {
        System.out.println("Setting up test database");
        // Database setup
    }

    @After("@database")
    public void cleanupDatabase() {
        System.out.println("Cleaning up test database");
        // Database cleanup
    }

    // Multiple hooks with order
    @Before(order = 1)
    public void firstSetup() {
        System.out.println("First setup");
    }

    @Before(order = 2)
    public void secondSetup() {
        System.out.println("Second setup");
    }

    // Hooks with multiple tags
    @Before("@smoke and @critical")
    public void setupForSmokeTests() {
        System.out.println("Setup for smoke tests");
    }

    // Hooks with negation
    @Before("not @skip")
    public void setupForNonSkippedTests() {
        System.out.println("Setup for tests not marked @skip");
    }
}
```

```
// Feature file using tags
/*
@database
Scenario: Test with database
  Given database is initialized
  When test executes
  Then database is cleaned

@smoke @critical
Scenario: Critical smoke test
  Given system is ready
  Then test runs with priority setup

@skip
Scenario: Skipped setup
  Given this uses normal setup
  Then no special setup applied
*/
```

Q75. Explain tags in Cucumber and how to run specific tags.

Tags mark scenarios for selective execution and organization.

```
# Feature file: login.feature

@smoke @critical
Scenario: User can login with valid credentials
  Given user is on login page
  When user enters valid credentials
  Then user sees dashboard

@regression
Scenario: User cannot login with invalid credentials
  Given user is on login page
  When user enters invalid credentials
  Then user sees error message

@database @slow
Scenario: Login with database verification
  Given database is connected
  When user logs in
  Then database records are updated

@skip
Scenario: Temporarily disabled test
  Given some precondition
  When action is performed
```

```
  Then result is verified

@mobile @ios
Scenario: Mobile app login
  Given mobile app is open
  When user logs in
  Then dashboard is displayed
```

Running Specific Tags:

```java
// Runner class
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/resources/features",
    glue = "com.automation.stepdefinitions",

    // Run specific tags
    tags = "@smoke",           // Run only @smoke tests

    // Run multiple tags (OR)
    tags = "@smoke or @critical",

    // Run multiple tags (AND)
    tags = "@smoke and @critical",

    // Negation
    tags = "not @skip",        // Exclude @skip tests
    tags = "@regression and not @slow",

    // Complex tag expressions
    tags = "(@smoke or @critical) and not @skip",

    monochrome = true,
    stepNotifications = true,
    plugin = {"pretty", "html:target/cucumber-report.html"}
)
public class CucumberRunner {
}

// Running from command line
// mvn test -Dcucumber.filter.tags="@smoke"
// mvn test -Dcucumber.filter.tags="@smoke and @critical"
// mvn test -Dcucumber.filter.tags="@smoke or @regression"
// mvn test -Dcucumber.filter.tags="not @skip"
// mvn test -Dcucumber.filter.tags="(@smoke or @critical) and not @slow"
```

Q76. What are Data Tables in Cucumber and how to use them?

Data Tables pass structured data to step definitions.

```
# Feature file: data_table.feature

Feature: User Management

Scenario: Create multiple users
  Given following users exist:
    | name  | email | role |
    | John  | john@example.com | admin |
    | Jane  | jane@example.com | user |
    | Bob   | bob@example.com | user |

  When system processes users
  Then total users should be 3

Scenario: Verify list of browsers
  Given browser list contains:
    | Chrome |
    | Firefox |
    | Safari |

  When user selects browser
  Then browser is compatible
```

Step Definitions for Data Tables:

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.datatable.DataTable;
import java.util.List;
import java.util.Map;

public class DataTableStepDefinitions {
    private List<Map<String, String>> userData;
    private List<String> browsers;

    // Method 1: Maps (List of Maps)
    @Given("following users exist:")
    public void followingUsersExist(DataTable dataTable) {
        userData = dataTable.asMaps(String.class, String.class);

        for (Map<String, String> user : userData) {
            String name = user.get("name");
            String email = user.get("email");
            String role = user.get("role");
```

```
            System.out.println("User: " + name + ", Email: " + email + ",
Role: " + role);
        }
    }

    // Method 2: Lists (Single column)
    @Given("browser list contains:")
    public void browserListContains(DataTable dataTable) {
        browsers = dataTable.asList(String.class);

        for (String browser : browsers) {
            System.out.println("Browser: " + browser);
        }
    }

    // Method 3: Raw list (All data as list of lists)
    @Given("user dat")
    public void userData(DataTable dataTable) {
        List<List<String>> rawData = dataTable.asLists(String.class);

        for (int i = 0; i < rawData.size(); i++) {
            List<String> row = rawData.get(i);
            System.out.println("Row " + i + ": " + row);
        }
    }

    // Method 4: Transform to custom object
    @Given("following users with custom object:")
    public void followingUsersCustom(DataTable dataTable) {
        List<User> users = dataTable.asList(User.class);

        for (User user : users) {
            System.out.println("User: " + user);
        }
    }

    @Then("total users should be {int}")
    public void totalUsersShouldBe(int expectedCount) {
        Assert.assertEquals(userData.size(), expectedCount);
    }
}

// Custom class for transformation
public class User {
    private String name;
    private String email;
    private String role;

    // Getters/Setters
```

```
    public void setName(String name) { this.name = name; }
    public void setEmail(String email) { this.email = email; }
    public void setRole(String role) { this.role = role; }

    @Override
    public String toString() {
        return "User{" + "name='" + name + '\'' +
               ", email='" + email + '\'' +
               ", role='" + role + '\'' + '}';
    }
}
```

Q77. Explain Scenario Outline and Examples in Cucumber.

Scenario Outline is a template for running same scenario with different data sets.

```
# Feature file: scenario_outline.feature

Feature: Search Functionality

Scenario Outline: Search for products
  Given user is on search page
  When user searches for "<product>"
  Then results should display "<product>"
  And number of results should be "<count>"

  Examples:
    | product | count |
    | Laptop | 15 |
    | Phone | 25 |
    | Tablet | 10 |
    | Monitor | 8 |

Scenario Outline: Login with multiple credentials
  Given user is on login page
  When user enters username "<username>"
  And user enters password "<password>"
  Then login result is "<result>"
  And message displays "<message>"

  Examples: Valid Users
    | username | password | result | message |
    | user1 | pass1 | success | Welcome user1 |
    | user2 | pass2 | success | Welcome user2 |

  Examples: Invalid Users
    | username | password | result | message |
```

```
    | invalid | wrong | failure | Invalid credentials |
    | empty | empty | failure | Username required |
```

Step Definitions for Scenario Outline:

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;

public class ScenarioOutlineSteps {
    private String searchTerm;
    private int resultsCount;
    private String loginResult;
    private String message;

    @Given("user is on search page")
    public void userIsOnSearchPage() {
        System.out.println("User navigated to search page");
    }

    // Parameters from Examples
    @When("user searches for {string}")
    public void userSearchesFor(String product) {
        searchTerm = product;
        System.out.println("Searching for: " + product);
    }

    @Then("results should display {string}")
    public void resultsShouldDisplay(String expectedProduct) {
        System.out.println("Verifying results display: " + expectedProduct);
        Assert.assertEquals(searchTerm, expectedProduct);
    }

    @Then("number of results should be {string}")
    public void numberOfResultsShouldBe(String count) {
        resultsCount = Integer.parseInt(count);
        System.out.println("Results count: " + resultsCount);
    }

    @Given("user is on login page")
    public void userIsOnLoginPage() {
        System.out.println("User is on login page");
    }

    @When("user enters username {string}")
    public void userEntersUsername(String username) {
        System.out.println("Entering username: " + username);
    }

    @When("user enters password {string}")
```

```
    public void userEntersPassword(String password) {
        System.out.println("Entering password: " + password);
    }

    @Then("login result is {string}")
    public void loginResultIs(String result) {
        loginResult = result;
        System.out.println("Login result: " + result);
    }

    @Then("message displays {string}")
    public void messageDisplays(String msg) {
        message = msg;
        System.out.println("Message: " + msg);
    }
}

// Execution:
// Scenario 1: Search for Laptop, 15 results
// Scenario 2: Search for Phone, 25 results
// Scenario 3: Search for Tablet, 10 results
// ... (continues for all Examples rows)
```

Q78. Explain Cucumber runner class and its configuration.

Runner Class controls Cucumber test execution and configuration.

```
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    // Feature files location
    features = {
        "src/test/resources/features",
        "src/test/resources/features/login",
        "src/test/resources/features/dashboard"
    },

    // Step definitions location
    glue = {
        "com.automation.stepdefinitions",
        "com.automation.hooks"
    },

    // Output format and location
    plugin = {
```

```
        "pretty",  // Console output
        "html:target/cucumber-report.html",  // HTML report
        "json:target/cucumber-report.json",  // JSON report
        "junit:target/cucumber-report.xml",  // JUnit XML
        "com.automation.listeners.CustomFormatter"  // Custom formatter
    },

    // Tag expressions
    tags = "@smoke and not @skip",

    // Publish options
    publish = true,  // Publish to Cucumber Reports

    // Scenario ordering
    dryRun = false,  // Check step definitions without executing

    // Monochrome output
    monochrome = false,

    // Step notification
    stepNotifications = true,

    // Strict mode (fail if undefined steps)
    strict = false,

    // Snippets format
    snippets = SnippetType.CAMELCASE
)
public class CucumberRunner {
    // Empty class - just configuration
}

// Maven POM configuration
/*
<plugin>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>7.0.0</version>
</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <includes>
            <include>**/CucumberRunner.java</include>
        </includes>
    </configuration>
</plugin>
```

```
*/

// Running tests
// mvn clean test
// mvn test -Dcucumber.filter.tags="@smoke"
// mvn test -Dcucumber.options="--dryRun"
```

Q79. How to implement background in Cucumber feature files?

Background contains common steps that execute before each scenario in a feature.

```
# Feature file: feature_with_background.feature

Feature: User Management
  Background: User is logged in
    Given user is on home page
    And user clicks login link
    When user enters valid credentials
    And user clicks submit button
    Then user should see dashboard

  Scenario: User can view profile
    When user clicks profile link
    Then user should see profile page
    And profile contains user information

  Scenario: User can update profile
    When user clicks edit profile button
    And user enters new email "newemail@example.com"
    And user clicks save button
    Then profile should be updated with new email

  Scenario: User can change password
    When user clicks settings link
    And user clicks change password
    And user enters old password
    And user enters new password
    Then password should be changed successfully

# Execution order for each scenario:
# Background steps execute first
# Then specific scenario steps execute
```

Step Definitions:

```
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
```

```java
public class BackgroundSteps {
    private WebDriver driver;
    private LoginPage loginPage;
    private DashboardPage dashboardPage;

    @Given("user is on home page")
    public void userIsOnHomePage() {
        driver.get("https://example.com");
        System.out.println("User navigated to home page");
    }

    @Given("user clicks login link")
    public void userClicksLoginLink() {
        loginPage = new LoginPage(driver);
        loginPage.clickLoginLink();
    }

    @When("user enters valid credentials")
    public void userEntersValidCredentials() {
        loginPage.enterUsername("testuser");
        loginPage.enterPassword("password123");
    }

    @When("user clicks submit button")
    public void userClicksSubmitButton() {
        dashboardPage = loginPage.clickSubmitButton();
    }

    @Then("user should see dashboard")
    public void userShouldSeeDashboard() {
        Assert.assertTrue(dashboardPage.isDisplayed());
    }

    // Scenario-specific steps
    @When("user clicks profile link")
    public void userClicksProfileLink() {
        dashboardPage.clickProfileLink();
    }

    @Then("user should see profile page")
    public void userShouldSeeProfilePage() {
        System.out.println("Profile page displayed");
    }
}

// Execution order:
/*
Scenario 1: User can view profile
```

```
1. Background: user is on home page
2. Background: user clicks login link
3. Background: user enters valid credentials
4. Background: user clicks submit button
5. Background: user should see dashboard
6. When: user clicks profile link
7. Then: user should see profile page
*/
```

Q80. Explain different ways to pass parameters in Cucumber step definitions.

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import io.cucumber.datatable.DataTable;
import java.util.List;
import java.util.Map;

public class ParameterPassingSteps {

    // 1. String parameter
    @Given("user enters {string}")
    public void userEnters(String text) {
        System.out.println("User entered: " + text);
        // Handles: Given user enters "Hello World"
    }

    // 2. Integer parameter
    @Given("user has {int} items")
    public void userHasItems(int count) {
        System.out.println("Item count: " + count);
        // Handles: Given user has 5 items
    }

    // 3. Float parameter
    @Given("price is {float}")
    public void priceIs(float price) {
        System.out.println("Price: " + price);
        // Handles: Given price is 10.50
    }

    // 4. Double parameter
    @Given("amount is {double}")
    public void amountIs(double amount) {
        System.out.println("Amount: " + amount);
        // Handles: Given amount is 100.99
```

```
    }

    // 5. Word parameter
    @When("user selects {word} option")
    public void userSelectsOption(String option) {
        System.out.println("Selected: " + option);
        // Handles: When user selects Chrome option
        // Note: {word} doesn't allow spaces
    }

    // 6. Multiple parameters
    @When("user enters {string} in {string} field")
    public void userEntersInField(String value, String fieldName) {
        System.out.println("Entering " + value + " in " + fieldName);
        // Handles: When user enters "test@example.com" in "email" field
    }

    // 7. Regular expression
    @Given("user has (\\d+) dollars")
    public void userHasDollars(int amount) {
        System.out.println("Amount in dollars: " + amount);
        // Handles: Given user has 100 dollars
    }

    // 8. Data Table parameter - Maps
    @Given("following user dat")
    public void followingUserData(DataTable dataTable) {
        List<Map<String, String>> data = dataTable.asMaps(String.class,
String.class);
        // Handles:
        // Given following user dat
        //   | name | email |
        //   | John | john@example.com |
    }

    // 9. Data Table parameter - Lists
    @When("user selects browsers:")
    public void userSelectsBrowsers(DataTable dataTable) {
        List<String> browsers = dataTable.asList(String.class);
        // Handles:
        // When user selects browsers:
        //   | Chrome |
        //   | Firefox |
        //   | Safari |
    }

    // 10. Optional parameters with default
    @When("user clicks button with {string} label")
    public void userClicksButton(String label) {
```

```
        System.out.println("Clicking button: " + label);
        // Handles: When user clicks button with "Submit" label
    }
}

// Feature file examples
/*
Feature: Parameter Examples

Scenario: Various parameter types
  Given user enters "Hello World"
  And user has 5 items
  And price is 10.50
  And amount is 100.99
  When user selects Chrome option
  And user enters "test@example.com" in "email" field
  And user has 100 dollars
  Then test completes

Scenario: Data tables
  Given following user dat
    | name | email | role |
    | John | john@example.com | admin |
    | Jane | jane@example.com | user |
  When user selects browsers:
    | Chrome |
    | Firefox |
    | Safari |
  Then test completes
*/
```

Q81. Explain Cucumber reports and how to integrate with reporting tools.

```
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/resources/features",
    glue = "com.automation.stepdefinitions",
    plugin = {
        "pretty",  // Console report
        "html:target/cucumber-html-report.html",  // HTML report
        "json:target/cucumber-report.json",  // JSON report (for further
processing)
```

```
        "junit:target/cucumber-report.xml",  // JUnit XML report
        "io.qameta.allure.cucumber7jvm.AllureCucumber7Jvm"  // Allure report
    },
    publish = true  // Publish to Cucumber Report Portal
)
public class CucumberRunner {
}

// Generated reports location
// HTML: target/cucumber-html-report.html
// JSON: target/cucumber-report.json
// JUnit XML: target/cucumber-report.xml
// Allure: target/allure-results/

// Maven POM for reports
/*
<plugin>
    <groupId>io.qameta.allure</groupId>
    <artifactId>allure-maven</artifactId>
    <version>2.11.2</version>
</plugin>

<dependency>
    <groupId>io.qameta.allure</groupId>
    <artifactId>allure-cucumber7-jvm</artifactId>
    <version>2.17.2</version>
</dependency>
*/

// View reports
// mvn clean test
// Report automatically generated in target/ folder

// Allure report
// mvn clean test
// mvn allure:report
// mvn allure:serve  // Open in browser
```

Q82. How to skip scenarios in Cucumber?

```
# Feature file: skip_scenarios.feature

Feature: Skip Examples

# Skip entire scenario with @skip tag
@skip
```

```
Scenario: This scenario is skipped
  Given precondition
  When action executes
  Then result is verified

# Skip specific scenario
@skip @manual
Scenario: Manual testing required
  Given test setup
  When manual action required
  Then manual verification needed

# Conditional skip in step definition
Scenario: Conditional skip
  Given system is ready
  When precondition check passes
  Then test continues
```

Step Definitions:

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import org.testng.SkipException;

public class SkipSteps {

    @Given("system is ready")
    public void systemIsReady() {
        System.out.println("System ready");
    }

    // Skip using SkipException
    @When("precondition check passes")
    public void preconditionCheckPasses() {
        String environment = System.getProperty("env", "dev");

        if ("dev".equals(environment)) {
            throw new SkipException("Skipping test in development
environment");
        }

        System.out.println("Precondition met");
    }

    @Then("test continues")
    public void testContinues() {
        System.out.println("Test executed");
    }
}
```

```java
// Runner configuration to exclude @skip scenarios
@CucumberOptions(
    features = "src/test/resources/features",
    glue = "com.automation.stepdefinitions",
    tags = "not @skip"  // Exclude @skip scenarios
)
public class CucumberRunner {
}

// Run from command line
// mvn test -Dcucumber.filter.tags="not @skip"
```

Q83. Explain World context/context object in Cucumber.

World/Context shares state between step definitions using Dependency Injection.

```java
// Create context class to hold shared data
import io.cucumber.java.Scenario;

public class ScenarioContext {
    private WebDriver driver;
    private Map<String, Object> testData = new HashMap<>();
    private Scenario scenario;

    public ScenarioContext(Scenario scenario) {
        this.scenario = scenario;
    }

    // Browser management
    public WebDriver getDriver() { return driver; }
    public void setDriver(WebDriver driver) { this.driver = driver; }

    // Test data management
    public void setTestData(String key, Object value) {
        testData.put(key, value);
    }

    public Object getTestData(String key) {
        return testData.get(key);
    }

    // Scenario info
    public String getScenarioName() {
        return scenario.getName();
    }

    public void log(String message) {
```

```
        System.out.println("[" + scenario.getName() + "] " + message);
    }
}

// Use context in step definitions
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;

public class ContextualSteps {
    private ScenarioContext context;

    // Constructor injection
    public ContextualSteps(ScenarioContext context) {
        this.context = context;
    }

    @Given("browser is initialized")
    public void browserIsInitialized() {
        WebDriver driver = new ChromeDriver();
        context.setDriver(driver);
        context.log("Browser initialized");
    }

    @Given("user data is prepared")
    public void userDataIsPrepared() {
        Map<String, String> userData = new HashMap<>();
        userData.put("username", "testuser");
        userData.put("password", "password123");

        context.setTestData("userData", userData);
        context.log("User data prepared");
    }

    @When("user logs in")
    public void userLogsIn() {
        WebDriver driver = context.getDriver();
        Map<String, String> userData =
            (Map<String, String>) context.getTestData("userData");

        driver.findElement(By.id("username"))
            .sendKeys(userData.get("username"));
        driver.findElement(By.id("password"))
            .sendKeys(userData.get("password"));

        context.log("User logged in");
    }

    @Then("user should be on dashboard")
```

```
    public void userShouldBeOnDashboard() {
        WebDriver driver = context.getDriver();
        Assert.assertEquals(driver.getTitle(), "Dashboard");
        context.log("Dashboard verified");
    }
}

// PicoContainer dependency injection setup
// Add to POM:
/*
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-picocontainer</artifactId>
    <version>7.0.0</version>
</dependency>
*/

// Runner configuration with glue path
@CucumberOptions(
    glue = "com.automation.stepdefinitions"
    // PicoContainer automatically handles dependency injection
)
public class CucumberRunner {
}
```

Q84. How to implement parallel execution in Cucumber?

```
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

// TestNG parallel execution
@CucumberOptions(
    features = "src/test/resources/features",
    glue = "com.automation.stepdefinitions",
    plugin = {"pretty", "json:target/cucumber-report.json"}
)
public class CucumberRunner {
}

// testng.xml with parallel execution
/*
<suite name="Parallel Suite" parallel="tests" thread-count="3">
    <test name="Smoke Tests">
        <classes>
            <class name="com.automation.runners.CucumberRunner"/>
```

```
        </classes>
    </test>

    <test name="Regression Tests">
        <classes>
            <class name="com.automation.runners.CucumberRunner"/>
        </classes>
    </test>
</suite>
*/

// Maven POM for parallel execution
/*
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-failsafe-plugin</artifactId>
    <version>3.0.0-M5</version>
    <configuration>
        <parallel>methods</parallel>
        <threadCount>4</threadCount>
        <includes>
            <include>**/CucumberRunner.java</include>
        </includes>
    </configuration>
</plugin>
*/

// Split features by runner classes
// Create multiple runner classes for parallel execution

@CucumberOptions(
    features = "src/test/resources/features/smoke",
    glue = "com.automation.stepdefinitions"
)
public class SmokeTestRunner {
}

@CucumberOptions(
    features = "src/test/resources/features/regression",
    glue = "com.automation.stepdefinitions"
)
public class RegressionTestRunner {
}

// Maven command for parallel execution
// mvn clean verify -Dparallel=methods -DthreadCount=4

// Feature file organization for parallel execution
/*
```

```
src/test/resources/features/
   ├── smoke/
   │   ├── login.feature
   │   ├── dashboard.feature
   ├── regression/
   │   ├── advanced_search.feature
   │   ├── user_management.feature
   └── integration/
       ├── database.feature
       └── api.feature
*/
```

Q85. Explain step definition best practices and how to create maintainable step definitions.

```
// GOOD: Clear, maintainable step definitions

// 1. Single Responsibility - Each step does one thing
@Given("user is on login page")
public void userIsOnLoginPage() {
    driver.get(ConfigReader.getProperty("base.url") + "/login");
    loginPage.verifyPageLoaded();
}

// BAD: Multiple actions in one step
@Given("user is on login page and database is connected and browser is
ready")
public void complexSetup() {
    // Too many responsibilities
}

// 2. Use Page Object Model
private LoginPage loginPage;
private DashboardPage dashboardPage;

@Given("user enters {string} in username field")
public void userEntersUsernameField(String username) {
    loginPage.enterUsername(username);
}

// 3. Use context/world for state management
private ScenarioContext context;

@When("user clicks login button")
public void userClicksLoginButton() {
    dashboardPage = loginPage.clickLoginButton();
    context.setTestData("currentPage", dashboardPage);
```

```java
}

// 4. Clear naming conventions
@Given("user has valid {string}")
public void userHasValidData(String dataType) {
    // dataType: credentials, email, phone, etc.
}

// 5. Avoid hard-coded values
// GOOD: Use parameters and configuration
@Given("user waits for {int} seconds")
public void userWaits(int seconds) {
    Thread.sleep(seconds * 1000);
}

// BAD: Hard-coded values
@Given("user waits")
public void userWaits() {
    Thread.sleep(5000);  // Hard-coded 5 seconds
}

// 6. Proper exception handling
@Then("element {string} should be visible")
public void elementShouldBeVisible(String locator) {
    try {
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(locator)));
    } catch (TimeoutException e) {
        throw new AssertionError("Element not visible: " + locator, e);
    }
}

// 7. Reusable step definitions
@When("user fills form:")
public void userFillsForm(DataTable dataTable) {
    Map<String, String> formData =
        dataTable.asMap(String.class, String.class);

    for (Map.Entry<String, String> entry : formData.entrySet()) {
        String fieldName = entry.getKey();
        String fieldValue = entry.getValue();
        loginPage.fillField(fieldName, fieldValue);
    }
}

// 8. Logging for debugging
@Given("user performs action {string}")
```

```java
public void userPerformsAction(String action) {
    System.out.println("Executing action: " + action);
    // Execute action
    System.out.println("Action " + action + " completed successfully");
}

// 9. Separation of concerns
public class StepDefinitions {
    // Step definitions - only coordinate actions
    @When("user logs in")
    public void userLogsIn() {
        loginActions.performLogin("testuser", "password");
    }
}

// Separate class for actual logic
public class LoginActions {
    public void performLogin(String username, String password) {
        // Implementation
    }
}

// 10. Documentation
/**
 * Verifies that user is on login page
 * - Navigates to login URL
 * - Validates page title
 * - Checks for login form elements
 */
@Given("user is on login page")
public void userIsOnLoginPage() {
    // Implementation
}

// Feature file demonstrating best practices
/*
Feature: User Login
  Background:
    Given user is on login page

  Scenario: Successful login
    When user fills form:
      | username | testuser |
      | password | password123 |
    And user clicks login button
    Then user should see dashboard
*/
```

Section 6: REST API Automation (16 Questions)

Q86. What is REST API and its principles?

REST (Representational State Transfer) is an architectural style for APIs using HTTP methods.

REST Principles: 1. **Client-Server**: Separation of client and server 2. **Statelessness**: Each request contains all information needed 3. **Uniform Interface**: Consistent API contract 4. **Cacheability**: Responses can be cached 5. **Layered System**: API can have multiple layers 6. **Code on Demand** (optional): Server can extend client functionality

HTTP Methods: | Method | Purpose | Idempotent | |———–|———|———–| | GET | Retrieve data | Yes | | POST | Create new resource | No | | PUT | Replace entire resource | Yes | | PATCH | Partial update | No | | DELETE | Remove resource | Yes |

Example:

```
GET /api/users/123          - Get user with ID 123
POST /api/users             - Create new user
PUT /api/users/123          - Replace user 123
PATCH /api/users/123        - Update user 123 partially
DELETE /api/users/123       - Delete user 123
```

Q87. Explain RestAssured library and its basic structure.

RestAssured is Java library for REST API testing.

```
import io.restassured.RestAssured;
import io.restassured.response.Response;
import io.restassured.response.ValidatableResponse;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class RestAssuredBasics {

    // Set base URL
    static {
        RestAssured.baseURI = "https://jsonplaceholder.typicode.com";
    }

    // Basic GET request
    @Test
    public void testGetRequest() {
        Response response = get("/posts/1");

        System.out.println("Status Code: " + response.getStatusCode());
        System.out.println("Response Body: " +
response.getBody().asString());
```

```
}

// GET with validation
@Test
public void testGetWithValidation() {
    given()
        .when()
            .get("/posts/1")
        .then()
            .statusCode(200)
            .body("userId", equalTo(1))
            .body("id", equalTo(1));
}

// GET with headers
@Test
public void testGetWithHeaders() {
    given()
        .header("Accept", "application/json")
        .header("Authorization", "Bearer token123")
        .when()
            .get("/posts/1")
        .then()
            .statusCode(200);
}

// GET with query parameters
@Test
public void testGetWithQueryParams() {
    given()
        .queryParam("userId", 1)
        .queryParam("_limit", 5)
        .when()
            .get("/posts")
        .then()
            .statusCode(200)
            .body("size()", lessThanOrEqualTo(5));
}

// POST request with JSON body
@Test
public void testPostRequest() {
    String requestBody = "{"
        + "\"title\": \"Test Post\","
        + "\"body\": \"This is a test post\","
        + "\"userId\": 1"
        + "}";

    given()
```

```
            .header("Content-Type", "application/json")
            .body(requestBody)
            .when()
                .post("/posts")
            .then()
                .statusCode(201)
                .body("title", equalTo("Test Post"));
    }

    // PUT request
    @Test
    public void testPutRequest() {
        String updateBody = "{"
            + "\"id\": 1,"
            + "\"title\": \"Updated Title\","
            + "\"body\": \"Updated body\","
            + "\"userId\": 1"
            + "}";

        given()
            .header("Content-Type", "application/json")
            .body(updateBody)
            .when()
                .put("/posts/1")
            .then()
                .statusCode(200)
                .body("title", equalTo("Updated Title"));
    }

    // DELETE request
    @Test
    public void testDeleteRequest() {
        given()
            .when()
                .delete("/posts/1")
            .then()
                .statusCode(200);
    }
}
```

Q88. Explain different HTTP status codes and their meanings.

| Code | Category | Meaning | |——|———-|———| | 200 | Success | OK - Request successful | | 201 | Success | Created - Resource created | | 204 | Success | No Content - Successful but no content returned | | 400 | Client Error | Bad Request - Invalid request syntax | | 401 | Client Error | Unauthorized - Authentication required | | 403 | Client Error | Forbidden - Authenticated but not authorized | | 404 | Client Error | Not Found - Resource

not found | | 405 | Client Error | Method Not Allowed | | 500 | Server Error | Internal Server Error | | 502 | Server Error | Bad Gateway | | 503 | Server Error | Service Unavailable |

Testing Examples:

```
public class StatusCodeTest {

    @Test
    public void test200StatusCode() {
        given()
            .when()
                .get("/posts/1")
            .then()
                .statusCode(200);
    }

    @Test
    public void test201CreatedStatusCode() {
        given()
            .header("Content-Type", "application/json")
            .body("{\"title\": \"Test\"}")
            .when()
                .post("/posts")
            .then()
                .statusCode(201);
    }

    @Test
    public void test404NotFoundStatusCode() {
        given()
            .when()
                .get("/posts/99999")
            .then()
                .statusCode(404);
    }

    @Test
    public void test400BadRequest() {
        given()
            .header("Content-Type", "application/json")
            .body("{invalid json}")
            .when()
                .post("/posts")
            .then()
                .statusCode(400);
    }
}
```

Q89. Explain JSON and XML response parsing in RestAssured.

```
import io.restassured.response.Response;
import static io.restassured.RestAssured.*;

public class ResponseParsingTest {

    // JSON Response Parsing
    @Test
    public void testJsonResponseParsing() {
        Response response =
get("https://jsonplaceholder.typicode.com/posts/1");

        // Get entire JSON
        String jsonBody = response.getBody().asString();

        // Parse specific values
        int userId = response.path("userId");
        String title = response.path("title");
        String body = response.path("body");

        System.out.println("User ID: " + userId);
        System.out.println("Title: " + title);

        // Assert values
        Assert.assertEquals(userId, 1);
        Assert.assertTrue(title.contains("sunt"));
    }

    // JSON Array Parsing
    @Test
    public void testJsonArrayParsing() {
        Response response =
get("https://jsonplaceholder.typicode.com/posts?userId=1");

        // Get list size
        int size = response.path("size()");

        // Get first element
        int firstId = response.path("[0].id");
        String firstTitle = response.path("[0].title");

        // Get specific element
        String thirdTitle = response.path("[2].title");

        Assert.assertEquals(firstId, 1);
        Assert.assertNotNull(firstTitle);
```

```
    }

    // Extract using JsonPath
    @Test
    public void testJsonPathExtraction() {
        Response response =
get("https://jsonplaceholder.typicode.com/posts/1");

        // Using JsonPath
        JsonPath jsonPath = response.jsonPath();

        int userId = jsonPath.getInt("userId");
        String title = jsonPath.getString("title");
        int id = jsonPath.getInt("id");

        Assert.assertEquals(userId, 1);
    }

    // Nested JSON Parsing
    @Test
    public void testNestedJsonParsing() {
        String responseBody = "{"
            + "\"user\": {"
            + "\"id\": 1,"
            + "\"name\": \"John\","
            + "\"address\": {"
            + "\"city\": \"New York\","
            + "\"zip\": \"10001\""
            + "}"
            + "}"
            + "}";

        Response response = given()
            .body(responseBody)
            .when()
                .post("https://example.com/api")
            .then()
                .extract()
                .response();

        String city = response.path("user.address.city");
        Assert.assertEquals(city, "New York");
    }

    // XML Response Parsing
    @Test
    public void testXmlResponseParsing() {
        Response response = given()
            .header("Accept", "application/xml")
```

```
            .when()
                .get("https://example.com/api/data.xml");

        // Using XmlPath
        XmlPath xmlPath = response.xmlPath();

        String title = xmlPath.getString("root.title");
        int userId = xmlPath.getInt("root.userId");

        Assert.assertEquals(userId, 1);
    }

    // Complex JSON Parsing with Collections
    @Test
    public void testComplexJsonParsing() {
        Response response =
get("https://jsonplaceholder.typicode.com/posts?userId=1");

        JsonPath jsonPath = response.jsonPath();

        // Get all IDs in array
        List<Integer> allIds = jsonPath.getList("id");

        // Get all titles where userId = 1
        List<String> allTitles = jsonPath.getList("title");

        // Verify list is not empty
        Assert.assertFalse(allIds.isEmpty());
    }
}

import io.restassured.path.json.JsonPath;
import io.restassured.path.xml.XmlPath;
```

Q90. How to handle request/response headers and authentication in RestAssured?

```
import io.restassured.RestAssured;
import io.restassured.builder.RequestSpecBuilder;
import io.restassured.builder.ResponseSpecBuilder;
import io.restassured.specification.RequestSpecification;
import io.restassured.specification.ResponseSpecification;
import static io.restassured.RestAssured.*;

public class HeadersAndAuthTest {

    // Basic Header Handling
```

```java
@Test
public void testRequestWithHeaders() {
    given()
        .header("Content-Type", "application/json")
        .header("Accept", "application/json")
        .header("User-Agent", "RestAssured-Test")
        .when()
            .get("https://jsonplaceholder.typicode.com/posts/1")
        .then()
            .statusCode(200);
}

// Multiple Headers using Map
@Test
public void testMultipleHeaders() {
    Map<String, String> headers = new HashMap<>();
    headers.put("Content-Type", "application/json");
    headers.put("Authorization", "Bearer token123");
    headers.put("X-API-Key", "api-key-123");

    given()
        .headers(headers)
        .when()
            .get("https://jsonplaceholder.typicode.com/posts/1")
        .then()
            .statusCode(200);
}

// Response Headers Validation
@Test
public void testResponseHeaders() {
    Response response = given()
        .when()
            .get("https://jsonplaceholder.typicode.com/posts/1")
        .then()
            .header("Content-Type", containsString("application/json"))
            .extract()
            .response();

    String contentType = response.getHeader("Content-Type");
    System.out.println("Response Content-Type: " + contentType);
}

// Bearer Token Authentication
@Test
public void testBearerTokenAuth() {
    String token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...";

    given()
```

```
            .header("Authorization", "Bearer " + token)
            .when()
                .get("https://api.example.com/protected-resource")
            .then()
                .statusCode(200);
    }

    // Basic Authentication
    @Test
    public void testBasicAuth() {
        given()
            .auth().basic("username", "password")
            .when()
                .get("https://api.example.com/protected")
            .then()
                .statusCode(200);
    }

    // OAuth 2.0 Token
    @Test
    public void testOAuth2Token() {
        // First, get access token
        Response tokenResponse = given()
            .contentType("application/x-www-form-urlencoded")
            .formParam("grant_type", "client_credentials")
            .formParam("client_id", "your_client_id")
            .formParam("client_secret", "your_client_secret")
            .when()
                .post("https://auth.example.com/oauth/token");

        String accessToken =
tokenResponse.jsonPath().getString("access_token");

        // Use token in API request
        given()
            .header("Authorization", "Bearer " + accessToken)
            .when()
                .get("https://api.example.com/data")
            .then()
                .statusCode(200);
    }

    // RequestSpecBuilder for reusable headers
    @Test
    public void testRequestSpecBuilder() {
        RequestSpecification requestSpec = new RequestSpecBuilder()
            .setBaseUri("https://jsonplaceholder.typicode.com")
            .addHeader("Content-Type", "application/json")
            .addHeader("Accept", "application/json")
```

```
            .build();

        given()
            .spec(requestSpec)
            .when()
                .get("/posts/1")
            .then()
                .statusCode(200);
    }

    // API Key Authentication
    @Test
    public void testApiKeyAuth() {
        given()
            .header("X-API-Key", "your-api-key-123")
            .when()
                .get("https://api.example.com/data")
            .then()
                .statusCode(200);
    }

    // Custom Headers with RestAssured Configuration
    static {
        RestAssured.defaultParser = Parser.JSON;
    }

    @Before
    public void setup() {
        RestAssured.baseURI = "https://jsonplaceholder.typicode.com";
    }
}
```

Q91. How to validate JSON schema in REST API tests?

```
import io.restassured.RestAssured;
import io.restassured.response.Response;
import static io.restassured.RestAssured.*;
import static io.restassured.module.jsv.JsonSchemaValidator.*;
import java.io.File;

public class JsonSchemaValidationTest {

    // Validate against JSON schema file
    @Test
    public void testJsonSchemaValidation() {
        given()
```

```
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .body(matchesJsonSchemaInClasspath("schemas/post-
schema.json"));
    }

    // Inline JSON schema validation
    @Test
    public void testInlineJsonSchema() {
        String jsonSchema = "{"
            + "\"$schema\": \"http://json-schema.org/draft-04/schema#\","
            + "\"type\": \"object\","
            + "\"properties\": {"
            + "\"userId\": {\"type\": \"integer\"},"
            + "\"id\": {\"type\": \"integer\"},"
            + "\"title\": {\"type\": \"string\"},"
            + "\"body\": {\"type\": \"string\"}"
            + "},"
            + "\"required\": [\"userId\", \"id\", \"title\", \"body\"]"
            + "}";

        given()
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .body(matchesJsonSchema(jsonSchema));
    }

    // Schema file: schemas/post-schema.json
    /*
    {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "type": "object",
      "title": "Post",
      "description": "Post schema",
      "properties": {
        "userId": {
          "type": "integer",
          "description": "User ID"
        },
        "id": {
          "type": "integer",
          "description": "Post ID"
        },
        "title": {
          "type": "string",
          "description": "Post title"
        },
        "body": {
```

```
      "type": "string",
      "description": "Post body"
    }
  },
  "required": ["userId", "id", "title", "body"],
  "additionalProperties": false
}
*/

// Advanced schema with array
@Test
public void testArrayJsonSchema() {
    String arraySchema = "{"
        + "\"$schema\": \"http://json-schema.org/draft-04/schema#\","
        + "\"type\": \"array\","
        + "\"items\": {"
        + "\"type\": \"object\","
        + "\"properties\": {"
        + "\"id\": {\"type\": \"integer\"},"
        + "\"title\": {\"type\": \"string\"}"
        + "},"
        + "\"required\": [\"id\", \"title\"]"
        + "}"
        + "}";

    given()
        .queryParam("userId", 1)
        .queryParam("_limit", 5)
        .when()
            .get("https://jsonplaceholder.typicode.com/posts")
        .then()
            .body(matchesJsonSchema(arraySchema));
}

// POM dependency
/*
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>5.3.1</version>
</dependency>
*/
}
```

Q92. How to test API with different content types (JSON, XML, Form Data)?

```java
import io.restassured.RestAssured;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class ContentTypeTest {

    // JSON Content Type
    @Test
    public void testJsonContentType() {
        String requestBody = "{"
            + "\"title\": \"Test Post\","
            + "\"body\": \"Test Body\","
            + "\"userId\": 1"
            + "}";

        given()
            .contentType("application/json")
            .body(requestBody)
            .when()
                .post("https://jsonplaceholder.typicode.com/posts")
            .then()
                .statusCode(201)
                .contentType(containsString("json"));
    }

    // XML Content Type
    @Test
    public void testXmlContentType() {
        String xmlBody = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            + "<post>"
            + "<title>Test Post</title>"
            + "<body>Test Body</body>"
            + "<userId>1</userId>"
            + "</post>";

        given()
            .contentType("application/xml")
            .accept("application/xml")
            .body(xmlBody)
            .when()
                .post("https://example.com/api/posts")
            .then()
                .statusCode(201);
    }

    // Form Data (URL Encoded)
    @Test
    public void testFormDataContentType() {
        given()
```

```
            .contentType("application/x-www-form-urlencoded")
            .formParam("username", "testuser")
            .formParam("password", "password123")
            .formParam("email", "test@example.com")
            .when()
                .post("https://example.com/api/login")
            .then()
                .statusCode(200);
    }

    // Multipart Form Data (File Upload)
    @Test
    public void testMultipartFormData() {
        File file = new File("path/to/file.txt");

        given()
            .multiPart("file", file)
            .multiPart("description", "Test file upload")
            .multiPart("userId", "1")
            .when()
                .post("https://example.com/api/upload")
            .then()
                .statusCode(200);
    }

    // Accept Header for response format
    @Test
    public void testResponseContentType() {
        given()
            .accept("application/json")
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .contentType(containsString("json"))
                .statusCode(200);
    }

    // Accept multiple content types
    @Test
    public void testMultipleAcceptTypes() {
        given()
            .accept("application/json, application/xml, text/plain")
            .when()
                .get("https://example.com/api/data")
            .then()
                .statusCode(200);
    }
}
```

Q93. Explain parameterization and data-driven testing in REST API tests.

```
import io.restassured.RestAssured;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
import static io.restassured.RestAssured.*;

public class DataDrivenRestTest {

    // DataProvider with multiple test cases
    @DataProvider(name = "postData")
    public Object[][] getPostData() {
        return new Object[][] {
            {1, "Test Post 1", "Body 1"},
            {2, "Test Post 2", "Body 2"},
            {3, "Test Post 3", "Body 3"},
            {4, "Test Post 4", "Body 4"}
        };
    }

    @Test(dataProvider = "postData")
    public void testCreatePostWithMultipleData(int userId, String title,
String body) {
        String requestBody = "{"
            + "\"userId\": " + userId + ","
            + "\"title\": \"" + title + "\","
            + "\"body\": \"" + body + "\""
            + "}";

        given()
            .header("Content-Type", "application/json")
            .body(requestBody)
            .when()
                .post("https://jsonplaceholder.typicode.com/posts")
            .then()
                .statusCode(201)
                .body("userId", equalTo(userId))
                .body("title", equalTo(title));
    }

    // DataProvider with CSV data
    @DataProvider(name = "csvData")
    public Object[][] readCSVData() {
        List<String[]> csvData = readCSVFile("src/test/resources/api-test-
data.csv");
        return csvData.toArray(new Object[0][]);
    }
```

```
    @Test(dataProvider = "csvData")
    public void testWithCSVData(String endpoint, String method, String
expectedStatus) {
        System.out.println("Testing: " + method + " " + endpoint);
    }

    // DataProvider with JSON data
    @DataProvider(name = "jsonData")
    public Object[][] getJsonTestData() throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        List<Map> data = mapper.readValue(
            new File("src/test/resources/test-data.json"),
            List.class
        );

        Object[][] testData = new Object[data.size()][1];
        for (int i = 0; i < data.size(); i++) {
            testData[i][0] = data.get(i);
        }
        return testData;
    }

    @Test(dataProvider = "jsonData")
    public void testWithJsonData(Map<String, Object> testData) {
        String endpoint = (String) testData.get("endpoint");
        String method = (String) testData.get("method");

        System.out.println("Endpoint: " + endpoint + ", Method: " + method);
    }

    // Parameterized endpoints
    @DataProvider(name = "endpoints")
    public Object[][] getEndpoints() {
        return new Object[][] {
            {"https://jsonplaceholder.typicode.com/posts/1"},
            {"https://jsonplaceholder.typicode.com/posts/2"},
            {"https://jsonplaceholder.typicode.com/posts/3"}
        };
    }

    @Test(dataProvider = "endpoints")
    public void testMultipleEndpoints(String endpoint) {
        given()
            .when()
                .get(endpoint)
            .then()
                .statusCode(200);
    }
```

```
    // Read from external file
    private List<String[]> readCSVFile(String filePath) {
        List<String[]> data = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                data.add(line.split(","));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return data;
    }
}

// CSV file: src/test/resources/api-test-data.csv
/*
endpoint,method,expectedStatus
/posts/1,GET,200
/posts/2,GET,200
/posts/invalid,GET,404
/posts,POST,201
*/

// JSON file: src/test/resources/test-data.json
/*
[
  {
    "endpoint": "/posts/1",
    "method": "GET",
    "expectedStatus": 200
  },
  {
    "endpoint": "/posts/2",
    "method": "GET",
    "expectedStatus": 200
  }
]
*/
```

Q94. How to handle API request/response logging in RestAssured?

```
import io.restassured.RestAssured;
import io.restassured.builder.RequestSpecBuilder;
import io.restassured.filter.log.RequestLoggingFilter;
```

```java
import io.restassured.filter.log.ResponseLoggingFilter;
import java.io.PrintStream;

public class ApiLoggingTest {

    // Enable logging for all requests/responses
    @Before
    public void setupLogging() {
        RestAssured.filters(new RequestLoggingFilter(), new
ResponseLoggingFilter());
    }

    // Request logging
    @Test
    public void testWithRequestLogging() {
        given()
            .log().all()  // Log all request details
            .header("Content-Type", "application/json")
            .body("{\"title\": \"Test\"}")
            .when()
                .post("https://jsonplaceholder.typicode.com/posts")
            .then()
                .statusCode(201);
    }

    // Response logging
    @Test
    public void testWithResponseLogging() {
        given()
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .log().all()  // Log all response details
                .statusCode(200);
    }

    // Log on condition (e.g., on failure)
    @Test
    public void testWithConditionalLogging() {
        given()
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .log().ifError()  // Log only if error
                .statusCode(200);
    }

    // Specific logging options
    @Test
```

```java
public void testWithSpecificLogging() {
    given()
        .log().headers()      // Log only headers
        .log().params()       // Log only parameters
        .log().body()         // Log only body
        .when()
            .get("https://jsonplaceholder.typicode.com/posts/1")
        .then()
            .log().headers()  // Log response headers
            .log().body()     // Log response body
            .statusCode(200);
}

// Log to file
@Test
public void testWithFileLogging() throws FileNotFoundException {
    PrintStream logStream = new PrintStream("api-logs.txt");

    given()
        .log().all(logStream)
        .when()
            .get("https://jsonplaceholder.typicode.com/posts/1")
        .then()
            .statusCode(200);
}

// Request specification with logging
@Test
public void testWithRequestSpec() {
    RequestSpecification requestSpec = new RequestSpecBuilder()
        .setBaseUri("https://jsonplaceholder.typicode.com")
        .addFilter(new RequestLoggingFilter())
        .addFilter(new ResponseLoggingFilter())
        .build();

    given()
        .spec(requestSpec)
        .when()
            .get("/posts/1")
        .then()
            .statusCode(200);
}

// Custom logging filter
@Test
public void testWithCustomFilter() {
    given()
        .filter((requestSpec, responseSpec, ctx) -> {
            System.out.println("=== REQUEST ===");
```

```
                System.out.println("Method: " + requestSpec.getMethod());
                System.out.println("Path: " +
requestSpec.getUserDefinedPath());

                var response = ctx.send(requestSpec, responseSpec);

                System.out.println("=== RESPONSE ===");
                System.out.println("Status: " + response.getStatusCode());
                System.out.println("Body: " + response.getBody().asString());

                return response;
            })
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .statusCode(200);
    }
}
```

Q95. How to handle timeouts, retries, and error scenarios in API testing?

```
import io.restassured.RestAssured;
import org.testng.annotations.Test;
import static io.restassured.RestAssured.*;

public class TimeoutRetryTest {

    // Timeout handling
    @Test
    public void testWithTimeout() {
        given()
            .socketTimeout(5000)      // 5 second timeout
            .connectionTimeout(3000)   // 3 second connection timeout
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .statusCode(200);
    }

    // Timeout with try-catch
    @Test
    public void testTimeoutWithExceptionHandling() {
        try {
            given()
                .socketTimeout(1000)
                .when()
```

```
                    .get("https://httpbin.org/delay/5")  // API delays for 5
seconds
                .then()
                    .statusCode(200);
        } catch (Exception e) {
            System.out.println("Request timed out: " + e.getMessage());
        }
    }

    // Retry mechanism
    @Test(retryAnalyzer = CustomRetryAnalyzer.class)
    public void testWithRetry() {
        given()
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/1")
            .then()
                .statusCode(200);
    }

    // Manual retry logic
    @Test
    public void testWithManualRetry() {
        int maxRetries = 3;
        int retryCount = 0;
        Response response = null;

        while (retryCount < maxRetries) {
            try {
                response = given()
                    .when()
                        .get("https://jsonplaceholder.typicode.com/posts/1");

                if (response.getStatusCode() == 200) {
                    break;  // Success
                } else {
                    retryCount++;
                }
            } catch (Exception e) {
                retryCount++;
                if (retryCount >= maxRetries) {
                    throw e;
                }
                try {
                    Thread.sleep(2000);  // Wait before retry
                } catch (InterruptedException ie) {
                    Thread.currentThread().interrupt();
                }
            }
        }
```

```
        Assert.assertNotNull(response);
        Assert.assertEquals(response.getStatusCode(), 200);
    }

    // Error scenario testing
    @Test
    public void testErrorScenario404() {
        given()
            .when()
                .get("https://jsonplaceholder.typicode.com/posts/99999")
            .then()
                .statusCode(404);
    }

    @Test
    public void testErrorScenario500() {
        given()
            .when()
                .get("https://httpbin.org/status/500")
            .then()
                .statusCode(500);
    }

    // Handle connection errors
    @Test
    public void testConnectionError() {
        try {
            given()
                .socketTimeout(2000)
                .when()
                    .get("https://invalid-domain-12345.com/api")
                .then()
                    .statusCode(200);
        } catch (Exception e) {
            System.out.println("Connection error: " + e.getMessage());
        }
    }
}

// Custom Retry Analyzer
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class CustomRetryAnalyzer implements IRetryAnalyzer {
    int count = 0;
    int maxRetry = 3;

    @Override
```

```
    public boolean retry(ITestResult result) {
        if (!result.isSuccess()) {
            if (count < maxRetry) {
                count++;
                return true;
            }
        }
        return false;
    }
}
```

Q96. Explain SOAP API testing and how it differs from REST.

| Feature | SOAP | REST | |———|——|——| | Protocol | XML-based | HTTP-based | | Data Format | XML | JSON, XML, plain text | | Method | Single POST endpoint | Multiple HTTP methods | | Communication | RPC style | Resource-oriented | | Standards | WSDL, XSD | OpenAPI, Swagger | | Complexity | Complex | Simple |

SOAP API Testing:

```
import javax.xml.soap.*;

public class SoapApiTest {

    @Test
    public void testSoapRequest() throws Exception {
        // Create SOAP message
        SOAPConnectionFactory soapConnectionFactory =
SOAPConnectionFactory.newInstance();
        SOAPConnection soapConnection =
soapConnectionFactory.createConnection();

        // Create request
        URL endpoint = new URL("https://example.com/soap");
        SOAPMessage soapRequest = createSOAPRequest();

        // Send request
        SOAPMessage soapResponse = soapConnection.call(soapRequest,
endpoint);

        // Print response
        soapResponse.writeTo(System.out);
    }

    private static SOAPMessage createSOAPRequest() throws Exception {
        MessageFactory messageFactory = MessageFactory.newInstance();
        SOAPMessage soapMessage = messageFactory.createMessage();
```

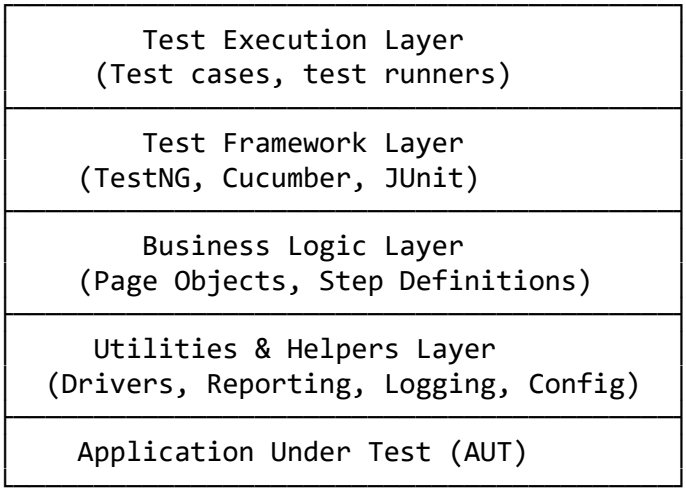
```
        SOAPPart soapPart = soapMessage.getSOAPPart();
        SOAPEnvelope envelope = soapPart.getEnvelope();
        envelope.addNamespaceDeclaration("example", "http://example.com/");

        SOAPBody soapBody = envelope.getBody();
        SOAPElement soapBodyElem = soapBody.addChildElement(
            "GetUser", "example"
        );
        SOAPElement soapBodyElem1 = soapBodyElem.addChildElement(
            "userId", "example"
        );
        soapBodyElem1.addTextNode("1");

        return soapMessage;
    }
}

// Using RestAssured for SOAP testing
public class SoapWithRestAssuredTest {

    @Test
    public void testSoapWithRestAssured() {
        String soapRequest = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            + "<soap:Envelope
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"
            + "<soap:Body>"
            + "<GetUser xmlns=\"http://example.com/\">"
            + "<userId>1</userId>"
            + "</GetUser>"
            + "</soap:Body>"
            + "</soap:Envelope>";

        given()
            .header("Content-Type", "text/xml; charset=UTF-8")
            .header("SOAPAction", "http://example.com/GetUser")
            .body(soapRequest)
            .when()
                .post("https://example.com/soap")
            .then()
                .statusCode(200)
                .body(

hasXPath("//soap:Envelope/soap:Body/GetUserResponse/User/Id")
                );
    }
}
```

Section 7: Common Framework Design & Best Practices (10 Questions)

Q97. Explain the concept of test automation framework architecture.

Test Automation Framework is a structured set of guidelines and tools for creating reliable, maintainable tests.

Layered Architecture:



Framework Components:

```
// 1. Driver Manager (Singleton)
public class DriverManager {
    private static WebDriver driver;

    public static WebDriver getDriver() {
        if (driver == null) {
            driver = new ChromeDriver();
        }
        return driver;
    }

    public static void quitDriver() {
        if (driver != null) {
            driver.quit();
            driver = null;
        }
    }
}

// 2. Configuration Reader
public class ConfigReader {
    private static Properties properties;
```

```
    static {
        properties = new Properties();
        try {
            properties.load(new FileInputStream("config.properties"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String getProperty(String key) {
        return properties.getProperty(key);
    }
}

// 3. Page Object Model
public class LoginPage {
    private WebDriver driver;
    private By usernameField = By.id("username");
    private By passwordField = By.id("password");
    private By loginButton = By.id("loginBtn");

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

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

    public void enterPassword(String password) {
        driver.findElement(passwordField).sendKeys(password);
    }

    public DashboardPage clickLogin() {
        driver.findElement(loginButton).click();
        return new DashboardPage(driver);
    }
}

// 4. Test Base Class
public class BaseTest {
    protected WebDriver driver;

    @BeforeMethod
    public void setUp() {
        driver = DriverManager.getDriver();
        driver.manage().window().maximize();
        driver.get(ConfigReader.getProperty("base.url"));
    }
```

```
    @AfterMethod
    public void tearDown(ITestResult result) {
        if (ITestResult.FAILURE == result.getStatus()) {
            takeScreenshot(result.getName());
        }
        DriverManager.quitDriver();
    }

    protected void takeScreenshot(String testName) {
        File screenshot = ((TakesScreenshot) driver)
            .getScreenshotAs(OutputType.FILE);
        try {
            FileUtils.copyFile(screenshot,
                new File("screenshots/" + testName + ".png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// 5. Test Class
public class LoginTest extends BaseTest {
    @Test
    public void testValidLogin() {
        LoginPage loginPage = new LoginPage(driver);
        loginPage.enterUsername("testuser");
        loginPage.enterPassword("password123");
        DashboardPage dashboardPage = loginPage.clickLogin();
        Assert.assertTrue(dashboardPage.isDashboardDisplayed());
    }
}
```

Q98. What is the Page Object Model (POM) and why is it important?

Page Object Model represents each web page as Java class with page elements and actions.

Benefits: - **Maintainability**: Changes in UI need updates in one place only - **Reusability**: Page objects can be used across multiple tests - **Readability**: Tests are more readable with meaningful method names - **Scalability**: Easy to add new pages and tests - **Reduced Duplication**: Locators and actions centralized

Implementation:

```
// Page Object Class
public class LoginPage {
    private WebDriver driver;
```

```
    // Locators
    private By usernameLocator = By.id("username");
    private By passwordLocator = By.id("password");
    private By loginButtonLocator = By.id("loginBtn");
    private By errorMessageLocator = By.className("error-message");

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

    // Page actions
    public void enterUsername(String username) {
        driver.findElement(usernameLocator).sendKeys(username);
    }

    public void enterPassword(String password) {
        driver.findElement(passwordLocator).sendKeys(password);
    }

    public DashboardPage clickLoginButton() {
        driver.findElement(loginButtonLocator).click();
        return new DashboardPage(driver);
    }

    public String getErrorMessage() {
        return driver.findElement(errorMessageLocator).getText();
    }

    public boolean isErrorMessageDisplayed() {
        try {
            return driver.findElement(errorMessageLocator).isDisplayed();
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}

// Dashboard Page Object
public class DashboardPage {
    private WebDriver driver;
    private By dashboardHeaderLocator = By.id("dashboardHeader");
    private By userGreetingLocator = By.className("user-greeting");

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

```
    public boolean isDashboardDisplayed() {
        try {
            WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(dashboardHeaderLocat
or));
            return true;
        } catch (TimeoutException e) {
            return false;
        }
    }

    public String getUserGreeting() {
        return driver.findElement(userGreetingLocator).getText();
    }
}

// Test using Page Objects
public class LoginTest {
    private WebDriver driver;

    @BeforeMethod
    public void setUp() {
        driver = new ChromeDriver();
        driver.get("https://example.com/login");
    }

    @Test
    public void testValidLogin() {
        LoginPage loginPage = new LoginPage(driver);
        loginPage.enterUsername("testuser");
        loginPage.enterPassword("password123");

        DashboardPage dashboardPage = loginPage.clickLoginButton();
        Assert.assertTrue(dashboardPage.isDashboardDisplayed());

Assert.assertTrue(dashboardPage.getUserGreeting().contains("testuser"));
    }

    @Test
    public void testInvalidLogin() {
        LoginPage loginPage = new LoginPage(driver);
        loginPage.enterUsername("invaliduser");
        loginPage.enterPassword("wrongpassword");
        loginPage.clickLoginButton();

        Assert.assertTrue(loginPage.isErrorMessageDisplayed());
        Assert.assertEquals(loginPage.getErrorMessage(), "Invalid
```

```
credentials");
    }

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

Q99. Explain CI/CD integration with test automation frameworks.

CI/CD integrates automated tests into continuous integration and deployment pipelines.

CI/CD Pipeline with Tests:

```
Developer Push Code
    ↓
Webhook Trigger
    ↓
Build Project
    ↓
Run Unit Tests
    ↓
Run Integration Tests
    ↓
Run Automation Tests (Smoke)
    ↓
Code Analysis
    ↓
Deploy to Staging
    ↓
Run Automation Tests (Full)
    ↓
Performance Tests
    ↓
Deploy to Production
    ↓
Monitor & Alert
```

Jenkins Pipeline Example:

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building project...'
                sh 'mvn clean compile'
```

```
        }
    }

    stage('Unit Tests') {
        steps {
            echo 'Running unit tests...'
            sh 'mvn test'
        }
    }

    stage('Smoke Tests') {
        steps {
            echo 'Running smoke tests...'
            sh 'mvn test -Dgroups=smoke'
        }
    }

    stage('Regression Tests') {
        steps {
            echo 'Running regression tests...'
            sh 'mvn test -Dgroups=regression'
        }
    }

    stage('Deploy to Staging') {
        steps {
            echo 'Deploying to staging...'
            sh 'docker build -t app:${BUILD_NUMBER} .'
            sh 'docker push app:${BUILD_NUMBER}'
            sh 'kubectl set image deployment/app app=app:${BUILD_NUMBER}'
        }
    }

    stage('E2E Tests') {
        steps {
            echo 'Running E2E tests...'
            sh 'mvn verify -Dcucumber.filter.tags="@e2e"'
        }
    }
}

post {
    always {
        // Generate reports
        junit 'target/surefire-reports/*.xml'
        publishHTML([
            reportDir: 'target/surefire-reports',
            reportFiles: 'index.html',
            reportName: 'Test Report'
```

```
            ])

            // Archive test results
            archiveArtifacts artifacts: 'target/**', allowEmptyArchive: true
        }
        success {
            echo 'Pipeline succeeded!'
            // Deploy to production
        }
        failure {
            echo 'Pipeline failed!'
            // Send notification
            emailext(
                to: '${DEFAULT_RECIPIENTS}',
                subject: 'Build Failed: ${BUILD_NUMBER}',
                body: 'Test execution failed. Check logs for details.'
            )
        }
    }
}
```

POM Configuration:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M5</version>
    <configuration>
        <includes>
            <include>**/*Test.java</include>
        </includes>
        <suiteXmlFiles>
            <suiteXmlFile>${testngFile}</suiteXmlFile>
        </suiteXmlFiles>
        <parallel>methods</parallel>
        <threadCount>4</threadCount>
    </configuration>
</plugin>

<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.7</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
```

```
    </executions>
</plugin>
```

Q100. What are best practices for creating scalable and maintainable test automation frameworks?

Best Practices:

1. **Modular Architecture**: Separate concerns into different layers

```
// Good structure
src/
├── main/
│   ├── java/
│   │   ├── com/automation/
│   │   │   ├── pages/          // Page objects
│   │   │   ├── actions/        // User actions
│   │   │   ├── utils/          // Utilities
│   │   │   ├── config/         // Configuration
│   │   │   └── api/            // API clients
│   │   └── resources/
│   │       └── config.properties
│   └── test/
│       ├── java/
│       │   ├── com/automation/tests/  // Test cases
│       │   └── com/automation/listeners/ // Listeners
│       └── resources/
│           └── features/       // Feature files
```

2. **Centralized Configuration**:

```
public class ConfigReader {
    private static Properties properties = new Properties();

    static {
        loadProperties("src/main/resources/config.properties");
        loadEnvironmentSpecificConfig();
    }

    private static void loadEnvironmentSpecificConfig() {
        String env = System.getenv("ENVIRONMENT");
        if (env != null) {
            loadProperties("src/main/resources/" + env +
".properties");
        }
    }
```

```
    public static String getProperty(String key) {
        return properties.getProperty(key);
    }
}
```

3. **Data-Driven Testing**:

```
@DataProvider(name = "testData")
public Object[][] getTestData() {
    return readTestDataFromExternalSource();
}

@Test(dataProvider = "testData")
public void testWithMultipleDataSets(Map<String, String> testData) {
    // Test implementation
}
```

4. **Robust Waits and Synchronization**:

```
public class WaitUtility {
    public static WebElement waitForElement(WebDriver driver, By
locator, int seconds) {
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(seconds));
        return
wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
    }

    public static void waitForElementToDisappear(WebDriver driver, By
locator) {
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));

wait.until(ExpectedConditions.invisibilityOfElementLocated(locator));
    }
}
```

5. **Error Handling & Logging**:

```
public class Logger {
    private static final org.slf4j.Logger logger =
        LoggerFactory.getLogger(Logger.class);

    public static void info(String message) {
        logger.info(message);
    }

    public static void error(String message, Exception e) {
        logger.error(message, e);
    }
```

```
}

try {
    // Test code
} catch (Exception e) {
    Logger.error("Test failed", e);
    throw new AssertionError("Test execution error", e);
}
```

6. **Screenshot & Video Capture**:

```
public class ScreenshotUtility {
    public static void takeScreenshot(String testName) {
        File screenshot = ((TakesScreenshot) DriverManager.getDriver())
            .getScreenshotAs(OutputType.FILE);
        try {
            FileUtils.copyFile(screenshot,
                new File("screenshots/" + testName + ".png"));
        } catch (IOException e) {
            Logger.error("Screenshot failed", e);
        }
    }
}
```

7. **Reusable Utilities**:

```
public class ActionUtility {
    public static void switchToFrame(WebDriver driver, WebElement
frame) {
        driver.switchTo().frame(frame);
    }

    public static void handleAlert(WebDriver driver, String action) {
        Alert alert = driver.switchTo().alert();
        if ("ACCEPT".equals(action)) {
            alert.accept();
        } else {
            alert.dismiss();
        }
    }

    public static List<String> getDropdownOptions(WebDriver driver, By
locator) {
        Select dropdown = new Select(driver.findElement(locator));
        return dropdown.getOptions()
            .stream()
            .map(WebElement::getText)
            .collect(Collectors.toList());
    }
}
```

8. **Version Control Best Practices**:

```
.gitignore:
target/
.classpath
.project
screenshots/
reports/
*.log
```

9. **Code Quality Tools**:

```
<!-- POM: SonarQube integration -->
<plugin>
    <groupId>org.sonarsource.scanner.maven</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
    <version>3.9.0.2155</version>
</plugin>
```

10. **Documentation**: `java /** * Comprehensive documentation for framework usage, * setup instructions, and best practices */`

Bonus: Common Framework Design & Best Practices Questions (10 Additional)

Q101. How do you implement cross-browser testing?

```
import org.testng.annotations.Parameters;
import org.testng.annotations.BeforeClass;

public class CrossBrowserTest {
    protected WebDriver driver;

    @Parameters("browser")
    @BeforeClass
    public void setUp(String browser) throws Exception {
        if ("chrome".equals(browser)) {
            ChromeOptions options = new ChromeOptions();
            driver = new ChromeDriver(options);
        } else if ("firefox".equals(browser)) {
            FirefoxOptions options = new FirefoxOptions();
            driver = new FirefoxDriver(options);
        } else if ("edge".equals(browser)) {
            EdgeOptions options = new EdgeOptions();
            driver = new EdgeDriver(options);
```

```
        } else if ("safari".equals(browser)) {
            driver = new SafariDriver();
        }

        driver.manage().window().maximize();
        driver.get(ConfigReader.getProperty("base.url"));
    }

    @Test
    public void testCrossBrowser() {
        // Test runs on multiple browsers
    }

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

// testng.xml for cross-browser
/*
<suite name="Cross Browser Suite">
    <test name="Chrome Tests">
        <parameter name="browser" value="chrome"/>
        <classes>
            <class name="com.automation.tests.CrossBrowserTest"/>
        </classes>
    </test>

    <test name="Firefox Tests">
        <parameter name="browser" value="firefox"/>
        <classes>
            <class name="com.automation.tests.CrossBrowserTest"/>
        </classes>
    </test>

    <test name="Edge Tests">
        <parameter name="browser" value="edge"/>
        <classes>
            <class name="com.automation.tests.CrossBrowserTest"/>
        </classes>
    </test>
</suite>
*/
```

Q102. How to implement test environment management?

```java
public enum TestEnvironment {
    DEV("https://dev.example.com", "dev_db"),
    STAGING("https://staging.example.com", "staging_db"),
    PRODUCTION("https://prod.example.com", "prod_db");

    private String baseUrl;
    private String database;

    TestEnvironment(String baseUrl, String database) {
        this.baseUrl = baseUrl;
        this.database = database;
    }

    public String getBaseUrl() { return baseUrl; }
    public String getDatabase() { return database; }
}

public class EnvironmentManager {
    private static TestEnvironment currentEnvironment;

    static {
        String env = System.getProperty("env", "DEV");
        currentEnvironment = TestEnvironment.valueOf(env.toUpperCase());
    }

    public static TestEnvironment getEnvironment() {
        return currentEnvironment;
    }

    public static String getBaseUrl() {
        return currentEnvironment.getBaseUrl();
    }
}

// Usage
public class BaseTest {
    @BeforeClass
    public void setUp() {
        String baseUrl = EnvironmentManager.getBaseUrl();
        driver.get(baseUrl);
    }
}

// Run with specific environment
// mvn test -Denv=staging
```

Q103. How to implement custom assertions and validations?

```
public class CustomAssertions {

    public static void assertElementPresent(WebDriver driver, By locator) {
        try {
            driver.findElement(locator);
        } catch (NoSuchElementException e) {
            throw new AssertionError("Element not found: " + locator, e);
        }
    }

    public static void assertElementText(WebDriver driver, By locator, String
expectedText) {
        String actualText = driver.findElement(locator).getText();
        Assert.assertEquals(actualText, expectedText,
            "Text mismatch for element: " + locator);
    }

    public static void assertPageTitle(WebDriver driver, String
expectedTitle) {
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle, "Page title
mismatch");
    }

    public static void assertUrlContains(WebDriver driver, String
expectedUrl) {
        String currentUrl = driver.getCurrentUrl();
        Assert.assertTrue(currentUrl.contains(expectedUrl),
            "URL does not contain: " + expectedUrl);
    }
}

// Usage
public class ValidationTest {
    @Test
    public void testWithCustomAssertions() {
        CustomAssertions.assertElementPresent(driver, By.id("username"));
        CustomAssertions.assertPageTitle(driver, "Login Page");
        CustomAssertions.assertUrlContains(driver, "/login");
    }
}
```

Q104. How to implement reporting with screenshots and logs?

Already covered in Q66 and Q94. Here's a summary of integrated approach:

```
public class AdvancedReporter {
    private ExtentReports extentReports;
    private ExtentTest extentTest;
    private static final Logger logger =
LoggerFactory.getLogger(AdvancedReporter.class);

    public void startReport() {
        extentReports = new ExtentReports();
        extentReports.attachReporter(
            new ExtentSparkReporter("target/ExtentReport.html")
        );
    }

    public void createTest(String testName) {
        extentTest = extentReports.createTest(testName);
    }

    public void logPass(String message) {
        logger.info(message);
        extentTest.pass(message);
    }

    public void logFail(String message, Throwable exception) {
        logger.error(message, exception);
        extentTest.fail(exception);
        attachScreenshot();
    }

    public void attachScreenshot() {
        String screenshotPath = takeScreenshot();
        extentTest.addScreenCaptureFromPath(screenshotPath);
    }

    public void endReport() {
        extentReports.flush();
    }

    private String takeScreenshot() {
        File screenshot = ((TakesScreenshot) DriverManager.getDriver())
            .getScreenshotAs(OutputType.FILE);
        String path = "screenshots/" + System.currentTimeMillis() + ".png";
        try {
            FileUtils.copyFile(screenshot, new File(path));
        } catch (IOException e) {
            logger.error("Screenshot capture failed", e);
        }
```

```
        return path;
    }
}
```

Q105. How to handle test data management and test data cleanup?

```
public class TestDataManager {
    private DatabaseHelper dbHelper;
    private List<String> createdTestData = new ArrayList<>();

    public TestDataManager() {
        this.dbHelper = new DatabaseHelper();
    }

    // Create test data
    public Map<String, String> createTestUser(String username, String email)
{
        Map<String, String> userData = new HashMap<>();
        userData.put("username", username);
        userData.put("email", email);
        userData.put("password", "TestPass123!");

        // Insert into database
        String userId = dbHelper.insertUser(userData);
        createdTestData.add(userId);

        return userData;
    }

    // Cleanup test data
    public void cleanupAllTestData() {
        for (String testDataId : createdTestData) {
            dbHelper.deleteById(testDataId);
        }
        createdTestData.clear();
    }

    // Read test data from external source
    public List<Map<String, String>> getTestDataFromCSV(String filePath) {
        List<Map<String, String>> testData = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
            String line;
            String[] headers = reader.readLine().split(",");

            while ((line = reader.readLine()) != null) {
```

```
                String[] values = line.split(",");
                Map<String, String> row = new HashMap<>();
                for (int i = 0; i < headers.length; i++) {
                    row.put(headers[i], values[i]);
                }
                testData.add(row);
            }
        } catch (IOException e) {
            throw new RuntimeException("Failed to read test data", e);
        }
        return testData;
    }
}

// Usage
public class TestDataTest {
    private TestDataManager testDataManager;

    @BeforeClass
    public void setUp() {
        testDataManager = new TestDataManager();
    }

    @Test
    public void testWithManagedData() {
        Map<String, String> userData =
testDataManager.createTestUser("testuser", "test@example.com");
        // Use userData for test
    }

    @AfterClass
    public void tearDown() {
        testDataManager.cleanupAllTestData();
    }
}
```
