CTS Selenium Interview Questions with Answers Part 1 (2026) | Saravana Tech

Exact numbered coverage: Questions 1·100 | Elaborate answers and examples

> This guide preserves the numbering and subject of all 100 questions. For project-, version-, or DOM-dependent topics, answers state assumptions and provide adaptable examples. Use credentials, paths, URLs, and dependency versions approved for your environment.

1. What framework are you using?

Sample answer to adapt truthfully: “I use a Java-based Selenium WebDriver automation framework. Maven manages dependencies and builds, TestNG controls test execution and lifecycle, and Page Object Model separates page locators/actions from test logic. Where required, Cucumber/Gherkin is used for business-readable scenarios. Reusable utilities handle waits, configuration, test data, logging, screenshots, and reporting. The framework can be executed locally and through a CI pipeline.” Only name components you actually use.

2. Explain the framework functionality in depth

Walk through one test end to end: the developer or tester triggers execution locally or through CI; Maven resolves dependencies and invokes the test runner; TestNG/Cucumber selects tests; setup reads environment configuration and creates a browser session; page objects perform actions; assertions verify outcomes; listeners/hooks capture failures and evidence; reporting writes HTML/XML; teardown closes the WebDriver session. The architecture separates test intent from page details and reusable infrastructure.

EXAMPLE

```
CI / Eclipse -> Maven -> TestNG/Cucumber runner -> hooks / DriverFactory
-> page objects + utilities + test data -> Selenium WebDriver -> browser
-> assertions/listeners -> reports, logs, screenshots -> teardown
```

3. Your role in the automation team

Answer using your actual responsibilities. A QA/Test Lead may review requirements, select automation candidates, design/maintain framework components, write or review scripts, manage test data, triage failures, coordinate defect resolution, monitor regression runs, mentor team members, and communicate release risks. Give one real example of your contribution and its outcome; do not claim duties you did not perform.

4. Do you know Robot Framework?

Robot Framework is an open-source, keyword-driven automation framework. It provides readable test cases and supports libraries for web/API and other testing. SeleniumLibrary supports Selenium-based browser automation; Browser is a Playwright-based option. Robot Framework is not the same as Selenium WebDriver: Robot Framework is the test authoring/execution framework, while Selenium is a browser automation API.

EXAMPLE

```
*** Settings ***
Library    SeleniumLibrary
*** Test Cases ***
Open Example
    Open Browser    https://example.com    chrome
    Title Should Be    Example Domain
    Close Browser
```

> **Note:** If you have not used it hands-on, say you understand the concepts but have not used it professionally.

5. Explain TestNG XML structure

testng.xml defines a suite and its test blocks. It can specify parameters, classes/packages, groups, listeners, exclusions, and parallel execution. The suite tag is the top-level container; each test tag can select classes. Fully qualified Java class names are used in class name attributes.

EXAMPLE

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Regression" parallel="tests" thread-count="2">
  <parameter name="browser" value="chrome"/>
  <test name="Login">
    <classes>
      <class name="tests.LoginTest"/>
    </classes>
  </test>
</suite>
```

6. How do you connect to a database in Java?

Use JDBC (Java Database Connectivity). The usual steps are: have the vendor JDBC driver on the classpath; build the JDBC URL; open a Connection; create a PreparedStatement; bind parameters; execute a query/update; read the ResultSet if it is a query; close resources. Use try-with-resources and keep credentials outside source control.

EXAMPLE

```
String url = "jdbc:postgresql://localhost:5432/hr";
String sql = "SELECT emp_id, emp_name FROM employee WHERE emp_id = ?";
try (Connection con = DriverManager.getConnection(url, user, password);
     PreparedStatement ps = con.prepareStatement(sql)) {
    ps.setInt(1, 101);
    try (ResultSet rs = ps.executeQuery()) {
        while (rs.next())
            System.out.println(rs.getInt("emp_id") + " " + rs.getString("emp_name"));
    }
}
```

7. Database URL types apart from http://

Database URLs are not usually HTTP URLs. In Java JDBC, common forms are jdbc:mysql://host:3306/database, jdbc:postgresql://host:5432/database, jdbc:sqlserver://host:1433;databaseName=database, jdbc:oracle:thin:@//host:1521/service, and jdbc:sqlite:path/to/file.db. The exact syntax depends on the database and driver.

8. Is it mandatory to close Statement in DB code?

Yes, close ResultSet, Statement/PreparedStatement, and Connection when finished. They consume resources; leaks can exhaust database cursors or connection pools. Prefer try-with-resources, which closes resources even when an exception occurs.

9. Write database connection code and walk through it

The URL identifies the database. DriverManager creates a connection. PreparedStatement represents parameterized SQL and helps avoid SQL injection. setInt binds a value to the first ? placeholder. executeQuery returns a ResultSet. next advances row by row, and getString/getInt reads columns. try-with-resources closes everything.

EXAMPLE

```
String sql = "SELECT name FROM employee WHERE emp_id = ?";
try (Connection c = DriverManager.getConnection(url, user, password);
     PreparedStatement p = c.prepareStatement(sql)) {
    p.setInt(1, 25);
    try (ResultSet r = p.executeQuery()) {
        while (r.next()) System.out.println(r.getString("name"));
    }
}
```

10. Explain the List interface

List is an ordered Java collection. It supports index-based access and normally allows duplicate elements. Common implementations are ArrayList and LinkedList. Common methods include add, get, set, remove, size, contains, and iterator. List is an interface, so it is typically used through an implementation such as new ArrayList<>.

11. List interface usage with Selenium Select

Select represents a native HTML <select>. Its getOptions() method returns List<WebElement>, which lets you inspect the options. The List stores the option elements; the Select object provides selectByVisibleText, selectByValue, and selectByIndex.

EXAMPLE

```
WebElement e = driver.findElement(By.id("country"));
Select country = new Select(e);
List<WebElement> options = country.getOptions();
System.out.println(options.size());
country.selectByVisibleText("India");
```

12. Can you call selectByVisibleText/selectByIndex on a List reference?

No. List<WebElement> has collection methods such as get() and size(), but not Selenium Select methods. Those methods belong to the Select class. Use the Select object to select options in a native HTML select.

13. Select dropdown values without those Select methods

For a native select, find its option elements and click the option matching the required text. For a custom dropdown built from div/li/button elements, click the dropdown and then click the matching option. Inspect the HTML first; the native Select class does not work on custom dropdowns.

EXAMPLE

```
WebElement e = driver.findElement(By.id("country"));
for (WebElement option : e.findElements(By.tagName("option"))) {
    if (option.getText().trim().equals("India")) {
        option.click();
        break;
    }
}
```

14. Select the second dropdown value by index

Selenium uses zero-based indexes: index 0 is the first option, index 1 is the second. The element must be a native HTML select.

EXAMPLE

```
Select s = new Select(driver.findElement(By.id("country")));
s.selectByIndex(1); // second option
```

15. How do you parameterize in TestNG?

TestNG supports XML parameters through @Parameters and data-driven execution through @DataProvider. XML parameters are useful for browser/environment values; DataProvider is useful for multiple test data sets. Use configuration for non-secret settings and a secure secret store/environment variables for credentials.

EXAMPLE

```
@Parameters({"browser"})
@BeforeMethod
public void setup(String browser) { /* initialize selected browser */ }
```

16. Give an example of DataProvider

DataProvider returns a two-dimensional Object array. TestNG invokes the annotated test once for each row, passing row values as method arguments.

EXAMPLE

```
@DataProvider(name = "loginData")
public Object[][] data() {
    return new Object[][] {
        {"user1", "pass1"},
        {"user2", "pass2"}
    };
}
@Test(dataProvider = "loginData")
public void loginTest(String username, String password) {
    // perform login and assert the result
}
```

17. Object class and arrays

Object is the root superclass of ordinary Java classes. Arrays are also objects, so an array reference can be assigned to a variable of type Object. Object provides methods such as toString, equals, hashCode, getClass, wait, notify, and notifyAll. Array types are special JVM types; they are not declared as normal classes.

EXAMPLE

```
Object value = new int[] {1, 2, 3};
int[] numbers = (int[]) value;
System.out.println(numbers.length); // 3
```

18. How did you parameterize data in your framework?

Describe the real implementation: for example, DataProvider for combinations, testng.xml for browser/environment, Excel/CSV/JSON for test data, and properties/environment variables for configuration. Explain how rows are mapped to test arguments and how test data is isolated. Adapt this answer to your actual project.

19. Why JXL instead of Apache POI?

JXL is an older library that mainly supports .xls and has limited modern Excel feature support. Apache POI supports both .xls and .xlsx and is generally preferred for new Java projects. If your existing project uses JXL, explain that it may be legacy code or an .xls-only constraint; do not claim JXL is more capable.

20. Difference between JXL and Apache POI

JXL primarily supports Excel 97–2003 .xls files. Apache POI supports .xls through HSSF and .xlsx through XSSF and has broader workbook, formatting, and formula support. POI is more suitable for modern Excel files. For large .xlsx files, consider POI streaming APIs.

21. JXL sample code and method walkthrough

Legacy JXL example: Workbook.getWorkbook(File) opens the workbook; getSheet(0) selects the first sheet; getRows/getColumns report dimensions; getCell(column,row) takes the column first and row second, both zero-based; getContents returns cell text; close releases resources. JXL does not read .xlsx.

EXAMPLE

```
Workbook wb = null;
try {
    wb = Workbook.getWorkbook(new File("data.xls"));
    Sheet sh = wb.getSheet(0);
    for (int row = 0; row < sh.getRows(); row++) {
        for (int col = 0; col < sh.getColumns(); col++) {
            System.out.print(sh.getCell(col, row).getContents() + "\t");
        }
        System.out.println();
    }
} finally {
    if (wb != null) wb.close();
}
```

22. What reports do you generate?

Common choices include TestNG HTML/XML reports, Cucumber reports, ExtentReports, Allure, and CI summaries. Explain the report you actually use, including pass/fail/skip status, execution duration, failure stack trace, screenshots, logs, and where the report is published.

23. Ways to create customized HTML reports

Options include writing HTML directly from Java, transforming XML test results into HTML with XSLT, or using a reporting library/template engine. A report should escape inserted text and show test name, status, duration, failure details, and links to evidence. Writing HTML manually is possible, but a reporting library is easier to maintain for larger suites.

24. Can you design a framework? What would you consider?

Assess application type, test inventory, risk, stability, browser/environment matrix, data setup, test frequency, team skills, CI, reporting, maintainability, and execution time. Start with a small end-to-end slice. Separate runner, driver management, page objects, data/config, utilities, assertions/listeners, reporting, and CI. Add parallel execution only after driver/data/report components are thread-safe.

25. Compare Selenium JAR versions 2.45.0 and 2.46.0

These appear to be legacy Selenium 2 versions. The exact fixes and features between them depend on the specific artifact and release notes. Do not guess features from the version numbers alone; identify whether the interviewer means selenium-java or another JAR and compare its official release notes. Selenium 2 is obsolete for new projects.

26. Find India and click the checkbox beside it when its position changes

Anchor the locator on the stable text, locate the nearest shared row/container, and find the checkbox inside it. This avoids relying on a changing row index or screen coordinate. The exact ancestor depends on the actual DOM.

EXAMPLE

```
WebElement row = driver.findElement(
    By.xpath("//tr[.//*[normalize-space()='India']]"));
WebElement cb = row.findElement(By.cssSelector("input[type='checkbox']"));
if (!cb.isSelected()) cb.click();
```

> **Note:** If the page uses div/li rather than table rows, replace tr with the appropriate container.

27. Verify the expected heading colour

Use getCssValue("color") to read the computed CSS value. Browsers commonly return rgb()/rgba() even if the CSS source uses a hex colour. Compare normalized expected and actual values.

EXAMPLE

```
WebElement h = driver.findElement(By.cssSelector("h1.page-title"));
String actual = h.getCssValue("color");
Assert.assertEquals(actual, "rgba(0, 0, 0, 1)"); // example only
```

28. Find an element in three nested frames

Switch from the top document into frame 1, then frame 2, then frame 3, waiting for each frame. Find the element only after entering the innermost frame. Return to the main document with defaultContent().

EXAMPLE

```
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("frame1")));
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("frame2")));
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("frame3")));
String text = driver.findElement(By.id("target")).getText();
driver.switchTo().defaultContent();
```

29. What is WebElement: interface or class?

WebElement is an interface in Selenium's Java API representing a DOM element. It declares methods such as click, sendKeys, getText, getAttribute, getCssValue, isDisplayed, isEnabled, and isSelected. WebDriver implementations return objects that implement this interface.

30. Compile-time vs runtime polymorphism

Compile-time polymorphism is commonly achieved through method overloading: the compiler selects the method based on parameter types/count. Runtime polymorphism is achieved through overriding: the actual object's class determines which overridden instance method executes.

EXAMPLE

```
// Overloading: compile-time
void search(String text) {}
void search(String text, int page) {}

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

31. Why might an element not be identifiable if it is not an alert?

Possible reasons include wrong/stale locator, element in an iframe or shadow DOM, wrong window, element not loaded, hidden/conditionally rendered content, duplicate matches, wrong page state, or incorrect selector syntax. Inspect the DOM, count matches, verify frame/window context, and wait for the relevant condition.

32. Where do you keep locators in your framework?

In a POM framework, page-specific locators usually belong in that page's class; reusable widgets can have component classes. This keeps locators close to their actions and avoids scattering selectors across test cases. Avoid a single giant locator file unless the project has a clear reason for it.

33. Pass data without sendKeys

For ordinary user-like typing, sendKeys is the preferred WebDriver API. Alternatives depend on the requirement: Actions for keyboard interaction, clipboard paste, JavaScript to set a DOM value, or API/database setup to seed data. JavaScript may bypass input events and validation, so it should not be the default for user-flow testing.

34. Methods available in Alert

Switch using driver.switchTo().alert(). The common methods are accept(), dismiss(), getText(), and sendKeys() for prompt alerts. Use ExpectedConditions.alertIsPresent() to wait for an alert.

EXAMPLE

```
Alert a = wait.until(ExpectedConditions.alertIsPresent());
String message = a.getText();
a.accept();
```

35. Click a button without click() or Enter

Use Actions to move to the target and perform a mouse click. JavaScript click is possible but bypasses normal interaction checks and should generally be a fallback. If the interviewer specifically prohibits WebElement.click() and Enter, Actions.click() is a suitable answer.

EXAMPLE

```
new Actions(driver).moveToElement(button).click().perform();
```

36. Interface vs abstract class

An interface defines a contract and a class can implement multiple interfaces. An abstract class can have constructors, instance fields, concrete methods, and abstract methods, but Java allows a class to extend only one class. Interfaces cannot have constructors or per-instance fields; they can have default, static, and private methods.

37. Launch a browser and go to Google

Create a WebDriver, call get(), and close the session in finally or test teardown. Selenium Manager in supported Selenium 4 versions can manage compatible drivers.

EXAMPLE

```
WebDriver driver = new ChromeDriver();
try {
    driver.get("https://www.google.com");
    System.out.println(driver.getTitle());
} finally {
    driver.quit();
}
```

38. Estimate 100 test cases

Classify by complexity, stability, data/setup, automation feasibility, reusable components, browsers, integrations, and maintenance risk. Estimate analysis, coding, review, debugging, test data, CI, and stabilization separately. Provide assumptions and a range rather than estimating from count alone.

39. How many scripts can you automate per day?

There is no universal number. A simple stable test using existing page objects may be quick; a complex multi-page flow with OTP, file upload, database setup, or unstable UI may take much longer. Explain that you estimate by complexity, reuse, dependencies, data, and validation.

40. Why do you need a framework?

A framework provides reusable setup, conventions, utilities, configuration, reporting, and execution control. It reduces duplication and improves maintainability as the suite grows. Its complexity should be proportional to the suite; a throwaway script may not need a large framework.

41. Have you used Java Executor?

In Selenium interviews, this may mean JavascriptExecutor; clarify if needed. JavascriptExecutor executes JavaScript in the current page. java.util.concurrent.Executor is a different Java interface for executing tasks.

EXAMPLE

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("window.scrollTo(0, document.body.scrollHeight);");
```

42. Have you created an abstract class?

Answer based on your real project. A BaseTest may hold shared setup/teardown; an abstract base page may provide common behavior. Abstract classes are useful when shared state/implementation is needed, but composition and utility classes may be more suitable in other designs.

43. What are driver and By in driver.findElement(By.id(...))?

driver is a WebDriver reference representing the browser session. By is Selenium's locator factory class; By.id("username") describes how to locate an element. findElement returns the first matching WebElement or throws NoSuchElementException if none exists.

EXAMPLE

```
WebDriver driver = new ChromeDriver();
WebElement username = driver.findElement(By.id("username"));
```

44. Which IDE and how do you get JAR files?

Common IDEs include Eclipse and IntelliJ IDEA. In a Maven project, add dependencies to pom.xml and run Maven Update Project in Eclipse; Maven resolves the JARs. Manually adding JARs to the build path is possible but makes version management harder.

45. Read Excel data and explain

Apache POI is a common choice for .xls and .xlsx. Open the workbook, select a sheet, iterate rows/cells, and close resources. DataFormatter reads cell values as displayed text; handle formulas and blank cells according to the requirement.

EXAMPLE

```
try (InputStream in = new FileInputStream("testdata.xlsx");
     Workbook wb = WorkbookFactory.create(in)) {
    Sheet sh = wb.getSheet("LoginData");
    DataFormatter f = new DataFormatter();
    for (Row row : sh) {
        List<String> values = new ArrayList<>();
        for (Cell c : row) values.add(f.formatCellValue(c));
        System.out.println(values);
    }
}
```

46. Framework folder structure

One common Maven layout: src/test/java contains tests, pages, step definitions, hooks, utilities, and listeners; src/test/resources contains feature files, configuration, and test data; pom.xml contains dependencies/plugins. Reports and logs are generated artifacts and often excluded from source control.

EXAMPLE

```
project/
  pom.xml
  src/test/java/{tests,pages,steps,hooks,utilities,listeners}
  src/test/resources/{features,config,testdata}
  reports/  (generated)
```

47. Get all links in a web table

Locate the table first and then find anchors within it. Iterate through the returned list and read text and href; scope to the table to avoid collecting unrelated page links.

EXAMPLE

```
WebElement table = driver.findElement(By.id("resultsTable"));
for (WebElement a : table.findElements(By.cssSelector("a"))) {
    System.out.println(a.getText() + " -> " + a.getAttribute("href"));
}
```

48. Explain OOP concepts with real-time examples

Encapsulation bundles state and controls access (private bank balance); abstraction exposes essential behavior (Payment.pay()); inheritance extends behavior (Tester extends Employee); polymorphism allows different implementations behind one contract (CardPayment/UpiPayment).

EXAMPLE

```
interface Payment { void pay(); }
class CardPayment implements Payment { public void pay(){System.out.println("Card");} }
class UpiPayment implements Payment { public void pay(){System.out.println("UPI");} }
Payment p = new UpiPayment(); p.pay();
```

49. Selenium login script with coding standards

Use meaningful names, stable locators, explicit waits, assertions, environment-based credentials, and cleanup. In a real framework, move locators/actions to LoginPage and browser lifecycle to a driver factory/base class. The code below is illustrative; add imports and configuration for your project.

EXAMPLE

```
@BeforeMethod
public void setUp() {
    driver = new ChromeDriver();
    driver.get(System.getProperty("baseUrl"));
}
@Test
public void validLogin() {
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")))
        .sendKeys(System.getenv("TEST_USERNAME"));
    driver.findElement(By.id("password")).sendKeys(System.getenv("TEST_PASSWORD"));
    driver.findElement(By.id("loginButton")).click();
    wait.until(ExpectedConditions.urlContains("/dashboard"));
    Assert.assertTrue(driver.getCurrentUrl().contains("/dashboard"));
}
@AfterMethod(alwaysRun = true)
public void tearDown() { if (driver != null) driver.quit(); }
```

50. Rerun failed cases from 200 TestNG tests

TestNG normally creates test-output/testng-failed.xml after a run with failed tests. Run that XML to rerun failed methods, subject to your runner configuration. If the suite must retry during the same run, use IRetryAnalyzer carefully for transient failures; retries should not hide real defects.

EXAMPLE

```
mvn test -DsuiteXmlFile=testng-failed.xml
```

51. Get failed method name from ITestResult

ITestResult provides the invoked test method, status, parameters, and throwable. A listener can call getMethod().getMethodName() to record the method name. Use TestNG retry/listener mechanisms rather than directly invoking a test method by string name.

EXAMPLE

```
public void onTestFailure(ITestResult result) {
    System.out.println(result.getMethod().getMethodName());
    if (result.getThrowable() != null) result.getThrowable().printStackTrace();
}
```

52. Rerun all failed cases using testng-failed.xml

After the first execution, inspect the generated testng-failed.xml and run it as the suite input. Ensure the correct output directory and build tool configuration. This is distinct from automatically retrying a failure within the same run.

53. Design a framework from scratch

Start with application and test requirements, then design runner, driver factory, config, page objects, data readers, utilities, assertions, reporting/listeners, and CI. Build a small vertical slice, validate stability, then scale. Keep secrets outside source control and ensure test independence.

54. Design for a stable application with 100–200 tests

Prioritize stable, high-value regression cases and group them by business flow/risk. Identify shared components, data needs, browser coverage, run frequency, CI capacity, and maintenance owners. Choose a minimal architecture that supports repeatability, parallelism if required, reporting, and debugging without overengineering.

55. Explain your framework structure

Explain responsibilities by layer: tests describe scenarios; page objects encapsulate UI; driver factory/base test manages sessions; utilities handle waits/config/data; listeners/hooks capture outcomes; reporting generates evidence; Maven and CI build/run/publish. Use your actual folder names in the interview.

56. Is one ExtentReports initialization line enough?

No. Initialization creates/configures the report object, but you must create test entries, log steps/statuses, attach screenshots/errors as needed, call flush/finalize, and publish the report. For parallel execution, ensure test/report nodes are managed safely. Exact API depends on ExtentReports version.

57. Draw a block diagram of the framework

Requirements / Test Cases | TestNG / Cucumber Runner | Hooks / BaseTest / DriverFactory | Page Objects <--> Utilities / Config / Test Data | Selenium WebDriver | Browser / App | Assertions / Listeners | Reports + Logs + Screenshots + CI results

58. Excel or other data sheets?

Explain the actual project source. Excel suits business-managed tabular data; CSV is lightweight for flat data; JSON suits nested/API payloads; properties/environment variables are suitable for configuration. Choose based on shape, ownership, readability, and secure handling.

59. Read Excel rows/columns into a collection

List<List<String>> preserves row order and is easy for table-like data. List<Map<String,String>> lets each row be accessed by column name. Apache POI can read cells; DataFormatter helps normalize displayed values.

EXAMPLE

```
List<List<String>> rows = new ArrayList<>();
try (InputStream in = new FileInputStream("data.xlsx");
     Workbook wb = WorkbookFactory.create(in)) {
  Sheet sh = wb.getSheetAt(0);
  DataFormatter fmt = new DataFormatter();
  for (Row row : sh) {
    List<String> values = new ArrayList<>();
    for (Cell cell : row) values.add(fmt.formatCellValue(cell));
    rows.add(values);
  }
}
```

60. Why List and not HashMap?

List is suitable when row order and repeated rows matter. HashMap maps unique keys to values and does not represent multiple repeated records naturally. LinkedHashMap preserves insertion order for one record. A useful spreadsheet structure is List<Map<String,String>>.

61. Identify which of two elements is underlined

Read computed CSS property text-decoration-line and check whether it includes underline. If the visual underline is inherited or created by a pseudo-element, inspect the DOM/CSS more deeply.

EXAMPLE

```
String d = element.getCssValue("text-decoration-line");
boolean underlined = d != null && d.contains("underline");
```

62. New window with two nested frames: get text from inner frame

Save original handle; click the link; wait for the new handle; switch to it; switch to frame 1 then frame 2; locate and read the text; switch to default content when finished.

EXAMPLE

```
String original = driver.getWindowHandle();
driver.findElement(By.id("openWindow")).click();
wait.until(d -> d.getWindowHandles().size() > 1);
String second = driver.getWindowHandles().stream()
    .filter(h -> !h.equals(original)).findFirst().orElseThrow();
driver.switchTo().window(second);
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("frame1")));
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("frame2")));
String text = driver.findElement(By.id("target")).getText();
driver.switchTo().defaultContent();
```

63. How to close the second window?

After switching to the second window, call driver.close() to close that current window, then switch back to the original handle. Ensure the original window is still open.

EXAMPLE

```
driver.switchTo().window(secondHandle);
driver.close();
driver.switchTo().window(originalHandle);
```

64. Difference between quit and close

close() closes the current window/tab in the WebDriver session. quit() ends the session and closes all windows associated with that session. Neither should be expected to close unrelated browser windows manually opened outside the WebDriver session.

65. Will quit close manually opened browsers?

Normally no. quit() closes windows belonging to the WebDriver session it controls, not unrelated browser sessions the user opened manually. Keep test sessions isolated.

66. Selenium evolution from RC to WebDriver

Selenium RC used a server/proxy and JavaScript injection approach. WebDriver controls browsers through browser-specific drivers and later standardized around the W3C WebDriver protocol. Selenium 3 transitioned toward W3C; Selenium 4 completed it and improved Grid and added relative locators. RC is obsolete.

67. Why Maven and CI?

Maven manages dependencies, builds, and test execution from pom.xml. CI tools such as Jenkins, GitHub Actions, GitLab CI, Azure Pipelines, and Bitbucket Pipelines can checkout code, run mvn clean test, publish reports, and archive logs/screenshots. Use secure credentials and controlled test environments.

68. Does Maven look locally or on the Internet?

Maven checks the local repository first (usually ~/.m2/repository). If the artifact/version is absent, it tries configured remote repositories such as Maven Central, subject to network, mirrors, settings, and offline mode.

69. Page Object Model vs normal project model

Page Object Model is a design pattern: each logical page/component is represented by a class containing locators and actions, while tests describe workflows and assertions. A script-heavy project may place all locators/actions in test classes. POM centralizes UI details and improves reuse/maintenance. It is different from Maven's pom.xml.

70. Do I need objects for home, search, and results pages?

Creating a page object per logical page/component is common, but objects need not all be created eagerly. Create the page object when navigating to/interacting with that page. Methods may return the next page object after a transition.

71. Access search page without SearchPage object?

Yes, you can use driver.findElement directly, because Selenium does not require page objects. However, page-specific locators then leak into test code and maintenance becomes harder. POM is a design choice, not a technical prerequisite.

72. If home page is unavailable, can you move to later pages?

It depends on routing, authentication, and application state. A direct destination URL may work if the user is authorized; otherwise the application may redirect or fail if earlier actions establish required state. Verify actual business flow and preconditions.

73. Exceptions encountered in Selenium

Common exceptions: NoSuchElementException, StaleElementReferenceException, TimeoutException, ElementClickInterceptedException, ElementNotInteractableException, InvalidSelectorException, NoSuchFrameException, NoSuchWindowException, and UnhandledAlertException. Explain the cause and fix for one or two from your actual experience.

74. How do you handle exceptions?

Use explicit waits, stable locators, correct frame/window context, reacquire stale elements, and log useful context. Catch only exceptions where a valid fallback exists. Do not catch every exception and continue because the test may falsely pass.

75. Alert may or may not appear after click

Wait briefly for alertIsPresent. If it appears, read and accept/dismiss it; if no alert is a valid outcome, catch TimeoutException narrowly and continue.

EXAMPLE

```
driver.findElement(By.id("continue")).click();
try {
  Alert a = new WebDriverWait(driver, Duration.ofSeconds(2))
      .until(ExpectedConditions.alertIsPresent());
  System.out.println(a.getText()); a.accept();
} catch (TimeoutException noAlert) {
  // No alert is an allowed outcome
}
```

76. Reverse Cognizant without reverse()

Loop from the last index to zero, append charAt(i) to StringBuilder, and print the result.

EXAMPLE

```
String s = "Cognizant";
StringBuilder out = new StringBuilder();
for (int i = s.length()-1; i >= 0; i--) out.append(s.charAt(i));
System.out.println(out); // tnazingoC
```

77. Cognizant

The input string is "Cognizant". In the reverse program above, it is the string being processed; its reverse is "tnazingoC".

78. Find String length

String.length() returns the number of UTF-16 code units; for ordinary English text this matches the familiar character count.

EXAMPLE

```
System.out.println("Cognizant".length()); // 9
```

79. Loop from length-1 to 0

Use for (int i = value.length() - 1; i >= 0; i--). The last valid index is length-1, and index 0 is the first character.

80. Get character using charAt(i)

String.charAt(i) returns the UTF-16 char at zero-based index i. In a reverse loop, it returns characters from right to left.

81. Concatenate characters

Use StringBuilder.append(char) for efficient concatenation in a loop. Repeated + on immutable strings can create unnecessary intermediate objects.

82. Print reversed string

After building the StringBuilder, print it with System.out.println(builder). The result for Cognizant is tnazingoC.

83. Most repeated character in Cognizant

Every character in case-sensitive "Cognizant" appears once, so all characters tie for the highest frequency (1). For a general string, count characters with a Map and find the maximum count.

EXAMPLE

```
Map<Character,Integer> counts = new LinkedHashMap<>();
for (char c : "Cognizant".toCharArray())
    counts.put(c, counts.getOrDefault(c, 0) + 1);
System.out.println(counts);
```

84. Least repeated character in Cognizant

Every character appears once, so every character ties for the least frequency too. For a general string, count characters and find the minimum count; clarify how spaces and case are handled.

85. Print diamond pattern

Print an upper pyramid then a mirrored lower pyramid. Row i uses n-i-1 spaces and 2*i+1 stars; then iterate backwards from n-2 to zero.

EXAMPLE

```
int n=4;
for(int i=0;i<n;i++){
 for(int s=0;s<n-i-1;s++) System.out.print(" ");
 for(int k=0;k<2*i+1;k++) System.out.print("*");
 System.out.println();
}
for(int i=n-2;i>=0;i--){
 for(int s=0;s<n-i-1;s++) System.out.print(" ");
 for(int k=0;k<2*i+1;k++) System.out.print("*");
 System.out.println();
}
```

86. Second-last uppercase C position in the given string

Search backward, count uppercase C occurrences, and stop at the second. Print both zero-based index and one-based position so the interviewer can see which convention is used.

EXAMPLE

```
String s = "Congizant India Deveopement Centre";
int count=0, idx=-1;
for(int i=s.length()-1;i>=0;i--)
  if(s.charAt(i)=='C' && ++count==2){idx=i;break;}
System.out.println("index="+idx+", position="+(idx+1));
```

87. Replace a/e/i/o/u with 1/2/3/4/5

Map a→1, e→2, i→3, o→4, u→5. This example is case-insensitive and preserves other characters.

EXAMPLE

```
String s="Photon Info Tech";
StringBuilder b=new StringBuilder();
for(char c:s.toCharArray()) switch(Character.toLowerCase(c)){
 case 'a': b.append('1'); break; case 'e': b.append('2'); break;
 case 'i': b.append('3'); break; case 'o': b.append('4'); break;
 case 'u': b.append('5'); break; default: b.append(c);
}
System.out.println(b);
```

88. Tell me about the framework used in your project

Use the answer to Question 1, then add a truthful project example: what the application does, your responsibilities, the tools actually used, one automation challenge, how failures are diagnosed, and a measurable result only if you can support it.

89. How to read data from Excel?

Use Apache POI for modern .xlsx and .xls files. Open the workbook with WorkbookFactory, select a sheet, iterate rows and cells, use DataFormatter where appropriate, and close resources with try-with-resources. See Question 45 for a complete example.

90. How to handle multiple windows?

Save the original handle, click the link, wait until the handle count increases, identify the new handle by set difference, switch to it, and return to the original when done. Do not assume getWindowHandles order.

EXAMPLE

```
String original = driver.getWindowHandle();
driver.findElement(By.id("open")).click();
wait.until(d -> d.getWindowHandles().size() > 1);
String child = driver.getWindowHandles().stream()
  .filter(h -> !h.equals(original)).findFirst().orElseThrow();
driver.switchTo().window(child);
```

91. TestNG annotations and their structure

Typical order: @BeforeSuite, @BeforeTest, @BeforeClass, @BeforeMethod, @Test, @AfterMethod, @AfterClass, @AfterTest, @AfterSuite. @BeforeGroups/@AfterGroups apply around selected groups. Actual ordering can be affected by inheritance, XML settings, dependencies, and group hooks.

92. How to ignore TestNG test cases?

Use @Test(enabled=false) to disable a method or exclude methods/groups in testng.xml. @Ignore is commonly associated with JUnit, not the usual TestNG mechanism. Distinguish disabled tests from skipped tests caused by failed dependencies.

93. Can an interface have a constructor?

No. An interface cannot be instantiated and has no constructor. Implementing classes have constructors. Interface fields are implicitly public static final constants.

94. Access private data members from outside a class

Prefer public methods/getters/setters or behavior methods that preserve invariants. Reflection may access private members in some situations but can be restricted by Java modules and is not the default design. Avoid breaking encapsulation solely for testing.

95. HashMap vs Hashtable

HashMap is generally unsynchronized and permits one null key and null values. Hashtable is a legacy synchronized map and rejects null keys/values. ConcurrentHashMap is often preferred for concurrent access; LinkedHashMap preserves insertion order.

96. How to rate Selenium and Java from 0 to 5?

Give a calibrated answer supported by examples. For example: “Selenium 4/5 because I can build/debug UI tests, handle waits, frames, windows, POM, and execution. Java 3/5 because I’m comfortable with OOP, collections, exceptions, strings, and test code, and I’m strengthening advanced topics.” Adjust the rating to your actual ability; be ready for a coding question.

97. findElement vs findElements and missing locator result

findElement returns the first match and throws NoSuchElementException if none is found. findElements returns a List<WebElement>; if there are no matches, it returns an empty list, not null. Invalid XPath/CSS syntax may throw InvalidSelectorException for either method.

EXAMPLE

```
WebElement one = driver.findElement(By.id("missing")); // throws
List<WebElement> all = driver.findElements(By.id("missing"));
System.out.println(all.size()); // 0
```

98. ArrayList vs LinkedList

ArrayList uses a resizable array and gives O(1) average indexed access and fast iteration; middle insertions/removals shift elements. LinkedList uses linked nodes and indexed access is O(n); insert/remove is O(1) only once the node is located. ArrayList is usually faster in typical workloads due to locality and less per-element overhead.

99. Maven in Eclipse and exact pom.xml tags

Maven uses pom.xml. Main tags include project, modelVersion, groupId, artifactId, version, properties, dependencies/dependency, build/plugins/plugin, and optionally repositories, profiles, and dependencyManagement. Add dependencies and use Maven → Update Project in Eclipse. The following shows the exact dependency/plugin nesting; replace versions with compatible actual releases.

EXAMPLE

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>selenium-tests</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties><maven.compiler.release>17</maven.compiler.release></properties>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>REPLACE_WITH_COMPATIBLE_VERSION</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>REPLACE_WITH_COMPATIBLE_VERSION</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build><plugins><plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>REPLACE_WITH_COMPATIBLE_VERSION</version>
  </plugin></plugins></build>
</project>
```

100. What is TestNG, and how is it integrated with Selenium and pom.xml?

TestNG is a Java test framework providing annotations, assertions, groups, parameters, DataProvider, listeners, dependencies, and parallel execution. Selenium WebDriver performs browser automation; TestNG orchestrates tests. Add selenium-java and testng dependencies to pom.xml; Maven resolves them and Surefire commonly runs TestNG tests during the test phase. A suite XML can select classes, parameters, and parallel settings.

Final interview checklist

- Prepare a truthful 60–90 second explanation of your actual framework and trace a test from trigger to report.
- Practise JDBC and Excel code, including resource cleanup and parameter binding.
- Know TestNG annotations, XML, DataProvider, dependency/priority behavior, and failed-test reruns.
- Practise dynamic locators, Select, nested frames, windows, alerts, CSS colour, and findElement vs findElements.
- Explain why you chose each collection, report tool, test-data format, and framework component.
- For legacy JXL/Selenium 2 questions, distinguish historical context from modern recommendations.