LTI Selenium Interview Questions with Answers (2026) | Saravana Tech

87 interview questions with explanations, code examples, architecture diagrams, and sample answers

Personalization note: Adapt project, team-size, release-frequency, salary, location, and job-change answers to your real situation. For version-sensitive or legacy topics, verify the versions and APIs used by your actual project.

Section 1 — Framework, SDLC, and core Selenium

1. Draw the Selenium implementation architecture in your project

A typical Java automation setup separates test scenarios, framework code, Selenium WebDriver, browser drivers, browsers, the application under test, test data, and reports. Present this as a representative architecture and adjust it to match the actual project.

```
Test Scenarios (Cucumber feature files / TestNG tests)
                      |
                      v
Step Definitions / Test Classes
                      |
                      v
Page Objects (locators + page actions)
                      |
                      v
Utilities: waits, config, data, screenshots, logging
                      |
                      v
Selenium WebDriver API
                      |
                      v
Browser Driver / Selenium Manager
                      |
                      v
Chrome / Firefox / Edge  <---->  Application Under Test
                      |
                      v
Assertions + TestNG/Cucumber Reports + CI (e.g., Jenkins)
```

Explain the flow: a test starts a browser, calls page-object methods, WebDriver sends commands through the driver/browser automation protocol, the browser interacts with the application, assertions validate results, and reports capture outcomes. Selenium Grid can be added when tests run on remote or parallel browser nodes.

2. Explain the SDLC model used in your project

Sample answer for an Agile/Scrum project: “We follow Agile Scrum. Requirements are refined into user stories with acceptance criteria. The team estimates and plans work in a sprint, developers build the functionality, QA prepares scenarios and test data, and testing happens throughout the sprint. Defects are logged and triaged, and the team demonstrates completed work in the sprint review and discusses improvements in the retrospective.”

Mention the real sprint duration, ceremonies, and release process. SDLC is the overall development lifecycle; Scrum is a framework for organizing iterative product delivery.

3. How do you check whether text is present on a webpage?

For a specific element, get its text and compare it. For a broad page check, inspect visible body text or use a suitable locator. Selenium does not provide one universal “contains text anywhere” method; common methods include getText(), isDisplayed(), getAttribute(), and assertions from TestNG/JUnit.

```
String actual = driver.findElement(By.id("status")).getText();
Assert.assertTrue(actual.contains("Success"),
                  "Expected text was not present");

// Check a visible element:
WebElement message = driver.findElement(By.id("message"));
Assert.assertTrue(message.isDisplayed());
Assert.assertTrue(message.getText().contains("Saved"));
```

Avoid checking the entire body when a specific locator is available, because unrelated text can produce false positives.

4. How do you check checkbox selected/unselected without using isSelected()?

Use the checkbox input’s checked property through JavaScript. This is a workaround; isSelected() is the preferred Selenium API when available.

```
WebElement checkbox = driver.findElement(By.id("terms"));
Boolean checked = (Boolean) ((JavascriptExecutor) driver)
    .executeScript("return arguments[0].checked;", checkbox);
System.out.println(checked ? "Selected" : "Unselected");
```

For custom checkboxes, inspect a stable aria-checked attribute or class that reflects state. Do not infer selection from color alone unless that is the documented UI behavior.

5. How do you determine whether an element is visible or hidden?

isDisplayed() reports whether Selenium considers the element displayed. If the element is not in the DOM, findElement() throws NoSuchElementException; use findElements() or a wait if absence is expected.

```
List<WebElement> elements = driver.findElements(By.id("message"));
if (elements.isEmpty()) {
    System.out.println("Element is not present in DOM");
} else if (elements.get(0).isDisplayed()) {
    System.out.println("Element is visible");
} else {
    System.out.println("Element exists but is hidden");
}
```

6. Difference between findElement() and findElements()

- findElement(By): returns the first matching WebElement; throws NoSuchElementException if no match exists.
- findElements(By): returns a List<WebElement>; returns an empty list when there are no matches.

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

Use findElement() for a required unique element; use findElements() to count elements, iterate a collection, or safely check whether matching elements exist. The same question appears again in Question 16.

7. How do you handle popup windows?

First identify the popup type. A JavaScript alert/confirm/prompt uses driver.switchTo().alert(); a new browser window or tab uses getWindowHandles(); a DOM modal is handled with normal locators; an operating-system dialog may require a different approach and should be avoided when a browser-native upload/download mechanism is available.

8. What is TestNG?

TestNG is a Java testing framework that provides annotations, test grouping, dependencies, data providers, parameterization, parallel execution, assertions, and reporting hooks. It integrates with Selenium to organize and execute browser tests.

```
@BeforeMethod
public void setUp() { /* create driver */ }

@Test(groups = {"smoke"})
public void loginTest() {
    // test steps and assertions
}

@AfterMethod(alwaysRun = true)
public void tearDown() { /* quit driver */ }
```

9. How do you handle keyword-driven and data-driven frameworks?

In a keyword-driven approach, a test data sheet or test definition describes actions such as CLICK, TYPE, SELECT, and VERIFY; a keyword engine maps each keyword to reusable code. In a data-driven approach, the same test logic runs with multiple data sets supplied by a DataProvider, Excel/CSV/JSON, or a database. A project can combine both approaches.

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

@Test(dataProvider = "credentials")
public void loginTest(String user, String password) {
    // invoke login page and validate result
}
```

Do not say that a framework is keyword-driven unless it actually has a keyword-to-action mapping or equivalent design.

10. What is a hybrid framework?

A hybrid framework combines useful features from multiple approaches—for example, Page Object Model for page organization, data-driven inputs for test variations, keyword-like reusable actions, TestNG for execution, and reporting/logging utilities. The exact combination depends on the project. Describe the components you really use and why.

Section 2 — Files, pagination, locators, alerts, and collections

11. How do you upload a file to a webpage?

For a standard HTML file input, use sendKeys() with the full file path; avoid interacting with the OS file chooser.

```
WebElement upload = driver.findElement(By.cssSelector("input[type='file']"));
upload.sendKeys("C:\\test-data\\sample.pdf");
driver.findElement(By.id("submitUpload")).click();
```

If the file input is hidden, check whether the application exposes it for automation or whether the UI requires a supported interaction. AutoIt/Robot is a fallback for genuine OS-level dialogs, not the first choice.

12. How do you handle a popup in WebDriver?

This duplicates Question 7. Choose the handling strategy by popup type: JavaScript alert → Alert API; new tab/window → window handles; HTML modal → DOM locator; OS dialog → supported OS automation or an alternative upload/download route.

13. What framework are you using?

Sample answer: “I use a hybrid automation framework with Selenium WebDriver and Java. Page Object Model separates page-specific locators and actions from test logic. TestNG manages test execution, grouping, data providers, and lifecycle annotations. Maven manages dependencies and builds, Git manages version control, and reporting tools capture results. We also use explicit waits and reusable utilities to reduce flaky tests.”

Describe the tools in your project stack. If your framework uses Cucumber, explain how feature files and step definitions connect to page objects.

14. How do you handle pagination?

Loop through pages until the target record is found or the Next button is disabled. Re-locate rows after every page change because the DOM often refreshes.

```
boolean found = false;
while (true) {
    List<WebElement> rows = driver.findElements(By.cssSelector("table tbody tr"));
    for (WebElement row : rows) {
        if (row.getText().contains("Target Name")) {
            row.findElement(By.linkText("Edit")).click();
            found = true;
            break;
```

```
        }
    }
    if (found) break;

    WebElement next = driver.findElement(By.cssSelector("button.next"));
    if (!next.isEnabled() || "true".equals(next.getAttribute("aria-disabled"))) break;
    next.click();
    new WebDriverWait(driver, Duration.ofSeconds(10))
        .until(ExpectedConditions.stalenessOf(rows.get(0)));
}
if (!found) System.out.println("Target not found");
```

Adjust the Next-button locator and wait condition to the actual application. Some pagination components use page numbers or asynchronous loading instead.

15. What are the Selenium locators?

- id, name, className, tagName, linkText, partialLinkText, cssSelector, xpath.

```
driver.findElement(By.id("email"));
driver.findElement(By.name("password"));
driver.findElement(By.className("primary-button"));
driver.findElement(By.tagName("button"));
driver.findElement(By.linkText("Sign in"));
driver.findElement(By.partialLinkText("Sign"));
driver.findElement(By.cssSelector("button[type='submit']"));
driver.findElement(By.xpath("//button[normalize-space()='Sign in']"));
```

Prefer stable unique IDs or test attributes when available. Use CSS or XPath when the locator requires attributes, relationships, or text matching.

16. Difference between findElements and findElement

Same as Question 6: findElement returns one match and throws if none exists; findElements returns a list and an empty list when none exists.

17. How do you handle a popup window?

Same window-handle pattern as Question 46. Save the original handle, trigger the popup, wait for an additional handle, switch to it, perform actions, close it if appropriate, and switch back to the original window.

18. How do you handle an alert?

```
Alert alert = new WebDriverWait(driver, Duration.ofSeconds(5))
    .until(ExpectedConditions.alertIsPresent());
System.out.println(alert.getText());
alert.accept();       // OK
// alert.dismiss();   // Cancel
// alert.sendKeys("value"); // Prompt only
```

Do not use switchTo().alert() for HTML modals. For a prompt, sendKeys() can enter text before accepting.

19. Difference between List and Set

- List: ordered collection, supports index-based access, and allows duplicates.
- Set: stores unique elements; the iteration order depends on implementation (HashSet has no guaranteed order, LinkedHashSet preserves insertion order, TreeSet sorts elements).

Examples in automation: List<WebElement> for all table rows or dropdown options; Set<String> for unique window handles.

20. Why use Set for window handles?

WebDriver.getWindowHandles() returns Set<String> because each open window handle is unique and no index-based order is promised. Use iteration or compare the new set against the original set; do not assume the second handle is always the popup.

```
String original = driver.getWindowHandle();
driver.findElement(By.id("openWindow")).click();

new WebDriverWait(driver, Duration.ofSeconds(10))
    .until(d -> d.getWindowHandles().size() > 1);

for (String handle : driver.getWindowHandles()) {
    if (!handle.equals(original)) {
        driver.switchTo().window(handle);
        break;
    }
}
```

Section 3 — Maps, Agile, TestNG, Waterfall, and defects

21. Explain Map

Map stores key-value pairs. Keys are unique; values may repeat. Common implementations are HashMap (no guaranteed order), LinkedHashMap (insertion order), and TreeMap (keys sorted).

```
Map<String, String> userRoles = new HashMap<>();
userRoles.put("ravi", "Admin");
userRoles.put("anu", "Tester");
userRoles.put("ravi", "Lead"); // replaces previous value for key ravi
System.out.println(userRoles.get("ravi")); // Lead
```

22. Explain Agile methodology

Agile delivers software incrementally, using feedback and collaboration to adapt to changing requirements. In Scrum, work is planned in sprints; the team refines the backlog, holds daily Scrum, reviews the increment, and reflects in a retrospective. Testing is continuous, not only a final phase.

QA contributes to acceptance-criteria review, test design, exploratory testing, automation, defect triage, regression, and release-risk communication.

23. How do you use TestNG with WebDriver?

Use lifecycle annotations to initialize and clean up browser sessions, @Test to define tests, assertions to verify results, and @DataProvider or @Parameters for input data. Keep browser instances isolated for parallel runs (commonly one driver per thread/test context).

```
public class LoginTest {
    private WebDriver driver;

    @BeforeMethod
    public void start() { driver = new ChromeDriver(); }

    @Test
    public void titleCheck() {
        driver.get("https://example.com");
        Assert.assertTrue(driver.getTitle().contains("Example"));
    }

    @AfterMethod(alwaysRun = true)
    public void stop() {
        if (driver != null) driver.quit();
    }
}
```

24. TestNG vs JUnit

Both are Java test frameworks and both support assertions and lifecycle hooks. TestNG traditionally offers built-in groups, flexible dependencies, XML suite configuration, and DataProvider features. JUnit 5 offers a modern extension model, parameterized tests, and a strong ecosystem. The best choice depends on project requirements; avoid saying one is universally better.

25. Draw your Selenium framework architecture

This is a repeat of Question 1. Show the real layers in your project and be able to explain the call path from a feature/test to a page object, WebDriver, browser, assertion, report, and CI job.

26. Explain Waterfall model

Waterfall is a sequential lifecycle commonly described as requirements → design → implementation → testing → deployment → maintenance. It can be suitable when requirements are stable and formal phase approvals are important. Late changes can be costly because testing and feedback arrive later than in iterative models.

27. Explain defect life cycle

A typical workflow is New/Open → Assigned → In Progress → Fixed/Resolved → Retest → Closed. If the fix fails, QA reopens the defect. Other states may include Rejected/Invalid, Duplicate, Deferred, Cannot Reproduce, or Not a Bug, depending on the organization's workflow.

A good defect includes steps to reproduce, expected and actual results, environment/build, severity/priority, logs/screenshots, and relevant test data.

28. Explain SDLC life cycle

Typical phases are requirements gathering and analysis, planning/feasibility, design, development, testing, deployment, and maintenance. Actual models may be sequential or iterative. Testing activities and QA reviews can occur throughout the lifecycle.

29. Do you handle web tables in your project?

Sample answer: “Yes, where the application presents tabular data, I locate the table and rows, identify the row using a unique business value, and scope actions to that row. I also validate row counts, column values, sorting, filtering, and pagination as required.”

```
List<WebElement> rows =
    driver.findElements(By.cssSelector("#users tbody tr"));
for (WebElement row : rows) {
    List<WebElement> cells = row.findElements(By.cssSelector("td"));
    if (!cells.isEmpty() && cells.get(0).getText().trim().equals("Vajravel")) {
        row.findElement(By.linkText("Edit")).click();
        break;
    }
}
```

30. Tell me about yourself

Use a short, truthful answer: experience length; manual and automation strengths; core stack; leadership responsibilities; one achievement; why the role is relevant. Example: “I have around 13 years in software testing, covering manual and automation testing. My automation stack includes Selenium with Java, Cucumber, TestNG, API automation, and Playwright. I contribute to framework maintenance, test planning, execution analysis, defect investigation, and coordination across QA, development, and business teams. I’m looking for a role where I can combine hands-on automation with quality ownership.”

Section 4 — Project explanation, Excel, execution, frames, and drivers

31. Explain your project and its architecture

Use this sequence: business purpose and users → application components → your role → automation architecture → CI/reporting → major challenge and outcome. Avoid exposing confidential client data.

Sample architecture: TestNG/Cucumber tests → step definitions/test classes → page objects → reusable utilities/configuration/test data → Selenium WebDriver → browser → application. Results flow to assertions, logs, screenshots, reports, and the CI pipeline.

32. What framework are you following? Explain it.

Use the hybrid framework answer in Question 10 and describe each component with a real example. Explain why the design separates locators/actions, test logic, test data, and reporting.

33. Excel sheet coding

Apache POI is commonly used to read and write Excel workbooks. Choose the workbook implementation based on file type, and close resources with try-with-resources.

```
// Maven dependency: org.apache.poi:poi-ooxml (version managed by your project)
try (FileInputStream fis = new FileInputStream("testdata.xlsx");
     Workbook workbook = WorkbookFactory.create(fis)) {
    Sheet sheet = workbook.getSheet("Login");
    Row row = sheet.getRow(1);
    Cell cell = row.getCell(0);
    String value = new DataFormatter().formatCellValue(cell);
    System.out.println(value);
}
```

WorkbookFactory.create() can detect supported Excel formats, which is convenient when the input may be .xls or .xlsx. For formulas, decide whether to read formula text or evaluated/displayed values.

34. Difference between HSSF and XSSF

- HSSF handles the older binary .xls Excel format.
- XSSF handles the newer XML-based .xlsx format.

HSSF and XSSF are Apache POI APIs. For large .xlsx files, SXSSF supports streaming writes to reduce memory use. WorkbookFactory can often select the correct implementation from the file content.

35. What if data can come from .xls or .xlsx?

Use WorkbookFactory.create(InputStream/File) rather than manually selecting HSSFWorkbook or XSSFWorkbook. Ensure the Apache POI dependencies support both formats and handle corrupt files, passwords, missing sheets, and blank cells.

36. How do you run 50 test cases together?

With TestNG, list tests in testng.xml or use suite groups. Maven can invoke the suite; parallel execution is possible if tests are designed to be thread-safe and each test has an isolated WebDriver instance.

```
<suite name="RegressionSuite" parallel="methods" thread-count="4">
  <test name="Regression">
    <classes>
      <class name="tests.LoginTest"/>
      <class name="tests.ProfileTest"/>
    </classes>
  </test>
</suite>
```

To run through Maven, configure the Maven Surefire Plugin and execute mvn test. Parallelism is optional; begin sequentially if shared test data or application state can cause collisions.

37. Advantages of TestNG; why use it?

- Annotations for setup and cleanup
- Groups and suite selection
- DataProvider and parameterization
- Test dependencies and priorities where appropriate

- Parallel execution when tests are thread-safe
- XML suite configuration and integration with build tools

Use it because these features help organize, parameterize, and execute a large regression suite. Mention the actual features your team uses.

38. If two methods run in parallel, does TestNG create two separate reports?

Not necessarily. TestNG normally generates suite-level reports that include results from tests/methods in that run. Parallel methods do not automatically mean one separate report per method. You can configure listeners or reporting libraries to create separate files or grouped views. Ensure report-writing and driver state are thread-safe.

39. Multiple frames: find all frames, visit each, and identify an element

Switch into each frame, search for the target, and return to default content before trying the next frame. Nested frames require recursive traversal. The example below checks direct child frames and searches for an element by ID.

```
List<WebElement> frames = driver.findElements(By.cssSelector("iframe, frame"));
for (int i = 0; i < frames.size(); i++) {
    driver.switchTo().defaultContent();
    List<WebElement> currentFrames =
        driver.findElements(By.cssSelector("iframe, frame"));
    driver.switchTo().frame(currentFrames.get(i));

    List<WebElement> target = driver.findElements(By.id("targetElement"));
    if (!target.isEmpty()) {
        System.out.println("Found target in frame index " + i);
        // Interact with target.get(0) here if appropriate.
        break;
    }
}
driver.switchTo().defaultContent();
```

This is a basic one-level example. Re-query frame elements after DOM changes; frame WebElements can become stale. For nested frames, recursively enter child frames and track the frame path.

40. What locators are used in WebDriver?

Same locator list as Question 15: id, name, className, tagName, linkText, partialLinkText, cssSelector, and xpath.

Section 5 — WebDriver, browsers, capabilities, build, and repository

41. What is WebDriver?

WebDriver is Selenium's browser automation API. It lets code navigate, locate elements, interact with controls, manage windows/frames/alerts, and inspect browser state through a browser-specific driver or supported driver-management mechanism.

42. Write code for opening different browser drivers

```
WebDriver driver;
String browser = "chrome";

switch (browser.toLowerCase()) {
    case "chrome":
        driver = new ChromeDriver();
        break;
    case "firefox":
        driver = new FirefoxDriver();
        break;
    case "edge":
        driver = new EdgeDriver();
        break;
    default:
        throw new IllegalArgumentException("Unsupported browser: " + browser);
}
driver.get("https://example.com");
```

Add the appropriate Selenium imports and ensure browser binaries and driver management are correctly configured. Selenium Manager in modern Selenium 4 can assist with driver setup, but enterprise environments may pin binaries or manage drivers centrally.

43. Do we need a driver for Firefox? Why?

Selenium needs a WebDriver implementation to communicate with Firefox. GeckoDriver acts as the bridge between Selenium's WebDriver commands and Firefox. In modern Selenium 4 setups, Selenium Manager can help obtain/configure GeckoDriver automatically, but the browser automation still uses a driver implementation.

44. DesiredCapabilities for SSL

Older Selenium examples used DesiredCapabilities. In Selenium 4, prefer browser-specific Options classes. To accept insecure certificates in a test environment, set acceptInsecureCerts.

```
FirefoxOptions options = new FirefoxOptions();
options.setAcceptInsecureCerts(true);
WebDriver driver = new FirefoxDriver(options);

// Chrome example:
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setAcceptInsecureCerts(true);
WebDriver chrome = new ChromeDriver(chromeOptions);
```

Use only in a controlled test environment; accepting invalid certificates can hide a real security problem. Legacy DesiredCapabilities snippets may not match current Selenium APIs.

45. What build tool do you use? How do you schedule with a batch file?

A common Java setup uses Maven to manage dependencies, compile tests, and run TestNG suites. A Windows .bat file can change to the project folder, run Maven, and save output; Windows Task Scheduler or a CI tool can schedule it.

```
@echo off
cd /d C:\Automation\MyProject
call mvn clean test -DsuiteXmlFile=testng.xml
if errorlevel 1 exit /b 1
exit /b 0
```

Configure the Maven command/property to match your project. In a scheduled task, specify the working directory and ensure Java/Maven paths, credentials, browser installation, and environment variables are available.

46. How do you handle multiple windows?

```
String parent = driver.getWindowHandle();
driver.findElement(By.id("open")).click();

new WebDriverWait(driver, Duration.ofSeconds(10))
    .until(d -> d.getWindowHandles().size() > 1);

Set<String> handles = driver.getWindowHandles();
for (String handle : handles) {
    if (!handle.equals(parent)) {
        driver.switchTo().window(handle);
        System.out.println(driver.getTitle());
        break;
    }
}
// Return to parent when finished:
driver.switchTo().window(parent);
```

If multiple new windows can open, compare handles before and after the action and identify the target using title, URL, or content rather than assuming a particular iteration order.

47. If a link opens in a new tab, is it a separate window?

For WebDriver window handling, a browser tab is represented as a separate browsing context and has its own window handle. Use getWindowHandles() to switch to it. The API does not require you to distinguish a tab from a separate top-level window for this purpose.

48. What are Java collections and where do you use them?

The Java Collections Framework provides interfaces and implementations for working with groups of objects.

- List / ArrayList: rows, dropdown options, test data; preserves order and allows duplicates.
- Set / HashSet: unique window handles or unique values.
- Map / HashMap: test configuration, test data keyed by field name, or lookup tables.
- Queue / Deque: ordered processing where the use case requires it.

Pick a collection based on whether you need order, uniqueness, indexed access, or key-value lookup.

49. What is an object repository? What if you change ID locator to XPath?

An object repository centralizes element locators, traditionally in properties, JSON, YAML, or page-object classes. It makes locator maintenance easier. If a locator changes from ID to XPath, update the centralized locator definition or page-object field; tests should not need changes if they call the same page method.

```
// Page Object approach
private final By loginButton = By.id("login");

// If locator changes, update only this declaration:
private final By loginButtonUpdated =
    By.xpath("//button[normalize-space()='Login']");
```

Prefer a single authoritative locator declaration; the second declaration above is illustrative only. In Page Object Model, locators often live in the page class rather than an external properties file.

50. What challenges have you faced? Example: scrolling

Explain the context, why the normal action failed, how you diagnosed it, the fix, and how you prevented recurrence. For scrolling, prefer bringing an element into view and waiting for it to become interactable; use JavaScript scrolling when needed.

```
WebElement target = driver.findElement(By.id("save"));
((JavascriptExecutor) driver).executeScript(
    "arguments[0].scrollIntoView({block:'center'});", target);
new WebDriverWait(driver, Duration.ofSeconds(10))
    .until(ExpectedConditions.elementToBeClickable(target)).click();
```

Do not claim JavaScript solved every scrolling problem; sticky headers, overlays, lazy loading, and virtualized lists may require additional handling.

Section 6 — JavaScript, AutoIt, estimation, teams, and project delivery

51. Write code for JavascriptExecutor

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("window.scrollTo(0, document.body.scrollHeight);");
js.executeScript("arguments[0].scrollIntoView({block:'center'});", element);
js.executeScript("arguments[0].click();", element); // fallback only
String title = (String) js.executeScript("return document.title;");
```

Prefer normal WebDriver interactions first. JavaScript click can bypass user-like interaction checks, so use it only when justified and after investigating why a normal click fails.

52. How do you handle a window popup? You mentioned Robot and AutoIt.

Clarify whether it is a browser alert, new tab/window, HTML modal, or native OS dialog. Robot and AutoIt are external/OS-level techniques for certain native dialogs; they are not the correct solution for JavaScript alerts or normal web modals. For file upload, sendKeys() to input[type=file] is generally more stable.

53. What is AutoIt? How do you call an AutoIt script from Selenium?

AutoIt is a Windows automation scripting tool often used to interact with native Windows dialogs that Selenium cannot control directly. Compile a .au3 script to an .exe and launch it from Java. This is Windows-specific and can be brittle in headless or remote CI environments.

```
// Java: launch a compiled AutoIt executable
Process process = new ProcessBuilder(
    "C:\\Automation\\upload-dialog.exe",
    "C:\\test-data\\sample.pdf"
).start();

boolean completed = process.waitFor(30, TimeUnit.SECONDS);
if (!completed) {
    process.destroyForcibly();
    throw new RuntimeException("AutoIt script timed out");
}
if (process.exitValue() != 0) {
    throw new RuntimeException("AutoIt failed");
}
```

Illustrative AutoIt script (upload.au3):

```
; Example only: window title and control must match the real dialog
WinWaitActive("Open", "", 10)
ControlSetText("Open", "", "Edit1", $CmdLine[1])
ControlClick("Open", "", "Button1")
```

Compile the script with AutoIt and pass the file path as a command-line argument. Control IDs and window titles vary by dialog; test this in the target environment. Prefer WebDriver-native upload where possible.

54. How do you estimate an automation project?

Break the scope into candidate scenarios and estimate by complexity. Consider application stability, scenario count, browser/platform combinations, test data, environment readiness, reusable framework work, API/database setup, dynamic controls, file handling, authentication/captcha constraints, CI integration, reporting, maintenance, code review, debugging, and contingency.

A practical estimation sequence: discovery → automation feasibility and prioritization → framework/setup effort → scenario implementation → data/environment setup → execution and debugging → CI/reporting → review and maintenance buffer. Prioritize stable, high-risk, repeatable regression scenarios with clear expected results.

Explain assumptions and risks, and track actual effort to improve future estimates. Avoid giving a single universal test-case-per-day number.

55. Who gathers client requirements?

Usually the business analyst, product owner, client/business stakeholders, or a product manager gathers and clarifies requirements, with developers and QA participating. In Agile, QA reviews user stories and acceptance criteria and asks questions early. Answer based on your team's actual responsibility split.

56. How many people are in your team? What are your responsibilities?

State the actual team size and roles. A clear answer might be: "Our team has [number] people, including [roles]. I am responsible for test planning, automation design and maintenance, reviewing scenarios/code, execution tracking, defect triage, status reporting, and coordinating with development and business teams."

Use accurate details from your experience for each bracketed item.

57. Who writes manual test cases in your team?

It depends on team practice. QA engineers typically design and maintain test cases; manual testers may author functional/exploratory cases, while automation engineers identify suitable regression cases for automation. In many Agile teams, the same QA engineer does both. Explain your team's real workflow.

58. How do you identify complex scenarios?

Look for many decision paths, complex business rules, dependent systems, role/permission combinations, dynamic controls, asynchronous behavior, data dependencies, multi-step workflows, file uploads/downloads, cross-browser behavior, and difficult recovery paths. Use risk, business impact, execution frequency, and feasibility to prioritize.

59. How often is a build released and what is the minimum time to production?

Answer with your actual project cadence. Explain the difference between a developer build, a QA deployment, a sprint release, and a production release. A team may deploy to QA daily while production releases occur less frequently, depending on approvals, risk, and release policy. Do not invent a production timeline.

60. Tell me about your experience and framework

Combine your introduction (Question 30) with the framework explanation (Questions 10 and 31). Include years of experience, core stack, framework layers, responsibilities, one challenge, and a measurable result if you can substantiate it.

Section 7 — QTP, recovery, reports, HR questions

61. Difference between QTP and Selenium

QTP is the former name of HP QuickTest Professional, later part of the UFT product family; it is a commercial testing tool with vendor-supported features and licensing. Selenium is an open-source browser automation ecosystem with language bindings and broad browser support. Selenium often requires assembling libraries for reporting, test management, and other needs, while commercial suites may bundle more functionality. Compare the exact product/version used.

62. What is recovery and how do you handle it?

In automation, recovery means restoring a test or execution environment to a known state after a failure or unexpected condition. Examples include closing unexpected alerts, returning to default content, switching back to the original window, logging diagnostic information, cleaning up browser sessions, and resetting test data. Do not silently retry every failure because that can hide real defects.

```
try {
    // test actions
} catch (WebDriverException e) {
    // capture screenshot/log details here
    throw e; // preserve failure unless a deliberate recovery is valid
} finally {
    if (driver != null) driver.quit();
}
```

63. How do you generate reports?

TestNG provides default reports and can be extended using listeners. Teams may use Extent Reports, Allure, Cucumber reports, or CI test-result publishing. Explain which report you actually use and what it contains: pass/fail/skip, stack traces, screenshots, timestamps, environment/build, and links to logs.

A TestNG listener can capture test lifecycle events; keep report generation thread-safe for parallel execution.

64. What is your automation experience?

Summarize years and tools truthfully, then quantify the work where possible: number/type of scenarios, framework ownership, browsers, CI frequency, API coverage, defect detection, or regression-time reduction. Do not state metrics you cannot substantiate.

65. Tell me about yourself

Repeat Question 30 but tailor the last sentence to the job description. Keep it professional and avoid unnecessary personal details.

66. Why do you want to change jobs?

Positive sample: “I’m looking for a role with broader technical ownership, opportunities to deepen automation and quality engineering, and a chance to contribute to framework improvements and team outcomes. I appreciate what I’ve learned in my current role, and I want my next move to align with my longer-term growth.”

Keep it honest. Avoid criticizing current colleagues or sharing confidential information.

67. You already have an offer. Why did you come here?

Sample: “I’m evaluating opportunities based on role responsibilities, technical fit, team, growth, and long-term alignment—not only compensation. I’m interested in understanding this role properly before making a decision.” Use only if it accurately describes your situation.

68. Will you attend other company interviews if we provide an offer?

Answer honestly and professionally. Example: “If I receive an offer, I will evaluate it seriously against the role and agreed terms. I prefer to communicate transparently and make a considered decision within the agreed timeline.” Do not promise exclusivity unless you mean it.

69. Can you join the Bangalore location?

Answer based on your genuine willingness and constraints. If flexible: “I’m open to discussing Bangalore relocation or hybrid expectations, depending on the role and relocation timeline.” If not, state the constraint clearly and respectfully.

70. Have you travelled abroad?

Answer truthfully. If no: “I haven’t travelled abroad yet, but I’m open to business travel if the role requires it and the arrangements are feasible.” If yes, mention relevant professional travel briefly without oversharing.

Section 8 — Leadership, Java collections/OOP, Selenium behavior, waits

71. Have you handled a team?

Describe actual team size, mentoring, task allocation, review, risk tracking, stakeholder communication, and delivery responsibilities. Distinguish formal people management from technical leadership or coordination.

72. What take-home salary are you expecting?

Clarify that take-home depends on fixed pay, variable pay, PF, tax regime, insurance, and other deductions. Give a target range based on your researched compensation and priorities, and ask for the fixed/variable breakup. Do not equate CTC directly with monthly in-hand pay.

73. ArrayList vs LinkedList vs HashSet

- ArrayList: resizable array; fast indexed reads; insertion/removal in the middle may shift elements; allows duplicates.
- LinkedList: linked structure; efficient insert/remove when the position/node is already known, but indexed access is slower; allows duplicates.
- HashSet: unique elements; no guaranteed iteration order; average-case fast membership checks.

In Selenium, ArrayList/List is common for visible text and rows; HashSet is useful for uniqueness such as window handles.

74. Explain dynamic polymorphism

Dynamic polymorphism is runtime method dispatch: an overridden instance method is selected based on the actual object type.

```
class BrowserAction {
    void open() { System.out.println("Generic"); }
}
class ChromeAction extends BrowserAction {
    @Override void open() { System.out.println("Chrome"); }
}
BrowserAction action = new ChromeAction();
action.open(); // Chrome
```

Static methods are hidden rather than dynamically overridden.

75. Difference between bug and defect

In everyday QA practice, bug and defect are often used interchangeably for behavior that does not meet requirements. Some organizations distinguish them by workflow or context, but there is no universal distinction. A failure is the observed incorrect behavior; a defect is the underlying flaw that can cause it.

76. Alert

JavaScript alerts block normal page interaction until accepted or dismissed. Use driver.switchTo().alert(), then getText(), accept(), dismiss(), or sendKeys() for a prompt. See Question 18.

77. Difference between close() and quit()

- driver.close(): closes the current browser window/tab controlled by the current context.
- driver.quit(): closes all windows in that WebDriver session and ends the session.

Use quit() in teardown to avoid orphaned browser processes.

78. Difference between navigate().to() and get()

Both navigate to a URL. driver.get(url) is the direct navigation method. driver.navigate().to(url) is equivalent in purpose and is part of the navigation interface, which also offers back(), forward(), and refresh(). There is generally no meaningful difference in the final navigation for a standard URL.

79. Window handling

Use getWindowHandle() for the current window, getWindowHandles() for all open handles, and switchTo().window(handle) to change context. Save the original handle and identify a newly opened context by comparing sets or checking title/URL/content. See Question 46.

80. TestNG annotations

- @BeforeSuite / @AfterSuite: once around the suite.
- @BeforeTest / @AfterTest: around a TestNG <test> block.
- @BeforeClass / @AfterClass: before/after test methods in a class.
- @BeforeMethod / @AfterMethod: before/after each @Test method.
- @BeforeGroups / @AfterGroups: around specified groups.
- @Test: marks a test method; @DataProvider supplies data; @Parameters reads XML parameters; @Listeners registers listeners.

Setup and teardown scope matters. Use alwaysRun where cleanup must execute even after a failure.

81. Types of waits in Selenium

- Implicit wait: global element-location timeout; applies while locating elements.
- Explicit wait: waits for a specific condition, such as visibility or clickability.
- FluentWait: configurable timeout, polling interval, and ignored exceptions; WebDriverWait is a specialized wait commonly used with WebDriver.

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

Avoid mixing implicit and explicit waits because it can create unpredictable total wait times. Prefer explicit waits for dynamic UI behavior.

82. Purpose of static and constructors

static members belong to the class and are shared across instances; constructors initialize newly created objects. Static fields are appropriate for constants or shared class-level state, but a static WebDriver is risky in parallel tests because sessions can overwrite one another.

```
class User {
    static int count = 0;
    String name;

    User(String name) { // constructor
        this.name = name;
        count++;
    }
}
```

83. How do you coordinate LoginPage, HomePage, and MyHomePage?

Create page objects using the same WebDriver instance. A login action can return a HomePage object after successful navigation; a HomePage action can return MyHomePage. This models navigation and keeps tests readable.

```
class LoginPage {
    private final WebDriver driver;
    LoginPage(WebDriver driver) { this.driver = driver; }

    HomePage login(String user, String pass) {
        driver.findElement(By.id("username")).sendKeys(user);
        driver.findElement(By.id("password")).sendKeys(pass);
        driver.findElement(By.id("login")).click();
        return new HomePage(driver);
    }
}
class HomePage {
    private final WebDriver driver;
    HomePage(WebDriver driver) { this.driver = driver; }

    MyHomePage openMyHome() {
        driver.findElement(By.id("myHome")).click();
        return new MyHomePage(driver);
    }
}
class MyHomePage {
    private final WebDriver driver;
    MyHomePage(WebDriver driver) { this.driver = driver; }
}
```

Add assertions or waits to confirm navigation before returning the next page object. In larger frameworks, PageFactory may initialize elements, but it is optional.

84. Purpose of constructor, this, and PageFactory

- Constructor: initializes object state and dependencies, such as passing WebDriver into a page object.
- this: refers to the current object; this.driver = driver distinguishes the field from the constructor parameter.
- PageFactory: Selenium support utility that can initialize @FindBy WebElement fields using PageFactory.initElements(driver, this). It is optional; many teams prefer By locators and explicit findElement calls.

```
public class LoginPage {
    private WebDriver driver;

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

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

85. How does HashMap work? What happens with a duplicate key?

HashMap uses hashCode() to identify a bucket and equals() to distinguish keys within it. When put() is called with a key that is equal to an existing key, the existing mapping's value is replaced. Different keys can have hash collisions; correct equals()/hashCode() implementations are important.

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

86. Three static variables and three objects: how many copies?

There is one class-level copy of each static variable, not one copy per object. With three static variables and three objects of the same class, there are still three static fields total, shared by all three objects. Each non-static instance field has a separate copy per object.

87. What is FluentWait? Have you used it?

FluentWait lets you set the maximum wait duration, polling interval, and exceptions to ignore while checking a condition. WebDriverWait is the common specialized wait for WebDriver. Say you have used FluentWait only if you actually have; otherwise explain its purpose and give a small example.

```
Wait<WebDriver> wait = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(20))
    .pollingEvery(Duration.ofMillis(500))
    .ignoring(NoSuchElementException.class);

WebElement result = wait.until(d -> {
    WebElement element = d.findElement(By.id("result"));
    return element.isDisplayed() ? element : null;
});
```

Use ignored exceptions narrowly. Do not ignore broad WebDriverException types without a clear reason, because doing so can hide real failures.

Final revision checklist

- Practice drawing your real framework architecture and explaining the request flow.
- Prepare truthful answers for your project, SDLC, team size, release cadence, challenges, and job-change motivation.
- Remember the popup distinction: JavaScript alert, browser tab/window, HTML modal, or OS dialog.
- Use row-scoped locators for repeated actions such as Edit links in tables.
- Use explicit waits and keep WebDriver instances isolated for parallel tests.
- Review Java collections, OOP, HashMap, static fields, constructors, and equals/hashCode.
- Verify the Selenium/Java/browser/Apache POI/TestNG versions actually used by your project.