Bank of America Selenium Interview Questions with Answers (2026) | Saravana Tech

64 questions • Java • Selenium • TestNG • Maven • Jenkins • Manual Testing

Important: Adapt project, client-call, tool, Java-version, and experience answers to your real experience. Never disclose client secrets or commit passwords, session cookies, or tokens.

1. Tell me about yourself

Sample answer (adapt to your real experience): “I have around 13 years of experience in software testing across manual and automation testing. My automation skills include Java, Selenium WebDriver, Cucumber, TestNG, and API testing, with experience in test design, framework maintenance, defect analysis, and Agile delivery. In my current QA lead/project lead responsibilities, I work with business analysts, developers, and stakeholders to clarify requirements, plan coverage, maintain automation, investigate failures, and communicate release risks. I am looking for a role where I can contribute both hands-on automation and quality ownership.” Keep this to 60–90 seconds and be ready to give a concrete project example.

2. Explain your experience in automation

Explain the business flows automated, tools used, framework design, test data, reporting, CI execution, and your contribution. A strong answer describes an end-to-end example: requirement → scenario/test case → automation code → execution → defect triage → report. Mention measurable benefits only when you have evidence. Do not claim tools or ownership you have not personally used.

3. Role in automation

Typical responsibilities can include analysing automation candidates, estimating effort, creating reusable page objects/utilities, writing and reviewing scripts, managing test data, running smoke/regression suites, debugging flaky failures, logging defects, integrating with CI, and reporting results. For a lead role, include code reviews, prioritisation, mentoring, risk tracking, and coordination. Tailor the answer to your actual responsibilities.

4. Registration page: click link, fill fields from external data, submit

Use a Page Object for locators and a data reader for external values. This example uses a CSV file to keep the Selenium flow independent of the data source. Create `src/test/resources/registration.csv` with a header row and one data row: `firstName,lastName,email,password` then the values. For real applications, use synthetic test accounts and never commit real passwords.

```
// Java/Selenium example (CSV reader shown below)
WebDriver driver = new ChromeDriver();
driver.manage().window().maximize();
driver.get("https://example.test");

RegistrationData data = CsvReader.readFirst("src/test/resources/registration.csv");
driver.findElement(By.linkText("Register")).click();
driver.findElement(By.id("firstName")).sendKeys(data.firstName);
driver.findElement(By.id("lastName")).sendKeys(data.lastName);
driver.findElement(By.id("email")).sendKeys(data.email);
driver.findElement(By.id("password")).sendKeys(data.password);
driver.findElement(By.id("submit")).click();
```

5. Explain every line in the registration code

`new ChromeDriver()` starts a Chrome session; `maximize()` enlarges the window; `get()` navigates to the application; `CsvReader.readFirst(...)` reads one record from the external file into a data object; `findElement(By.linkText(...))` locates the Register link and `click()` opens the form; each `findElement(By.id(...)).sendKeys(...)` locates a field by its ID and types the matching value; the final click submits the form. In production, add explicit waits, validate the result, and close the driver in teardown.

6. Read 10 different test data sets for the registration scenario

Use a loop or TestNG `@DataProvider` to supply one row per test execution. The preferred approach is to read all rows once and return an `Object[][]` so TestNG reports each dataset as a separate invocation. Each row should use a unique email if the application prevents duplicate registrations. Do not share mutable data between parallel tests.

```
@DataProvider(name = "registrationData")
public Object[][] registrationData() throws IOException {
    List<String[]> rows = CsvReader.readAll("src/test/resources/registration.csv");
    Object[][] data = new Object[rows.size()][4];
    for (int i = 0; i < rows.size(); i++) {
        data[i] = rows.get(i);
    }
    return data;
}

@Test(dataProvider = "registrationData")
public void register(String first, String last, String email, String password) {
    // Open form, fill fields, submit, and assert success.
}
```

7. Get the count of all links in a web page

Use `findElements(By.tagName("a"))` and read the list size. `findElements` returns an empty list when nothing matches, so it is safer than `findElement` for counting. If the page loads links dynamically, wait for the relevant page state first.

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

8. Read data from Notepad/text file

A plain text file can be read using Java NIO `Files.readAllLines` or `Files.lines`. Use a path relative to the project or a configured external path. Ensure file encoding is known, and close streams (try-with-resources is preferred). For multiple test records, define a delimiter and validate each row.

```
Path path = Path.of("src/test/resources/testdata.txt");
List<String> lines = Files.readAllLines(path, StandardCharsets.UTF_8);
for (String line : lines) {
    System.out.println(line);
}
```

9. Read a file from a network location such as SharePoint

The approach depends on how the organisation exposes the document. If it is a synced SharePoint library, the file may be accessible as a local path, subject to permissions and sync status. For a secure cloud SharePoint site, prefer the approved SharePoint/Graph API or a company connector using OAuth and least-privilege access; do not hard-code passwords or scrape login pages. Download to a controlled temporary location, validate the file, and delete it when no longer needed. Network access, MFA, permissions, and company policy determine the implementation.

10. Manual testing cycle from requirements

1) Requirement analysis: review requirements, acceptance criteria, dependencies, and ambiguities. 2) Test planning: define scope, approach, resources, schedule, environments, risks, and entry/exit criteria. 3) Test design: prepare scenarios, cases, test data, and traceability matrix. 4) Environment setup: verify builds, configuration, accounts, and integrations. 5) Execution: run tests, record evidence, compare actual vs expected, and log defects. 6) Defect retest and regression: verify fixes and check for side effects. 7) Closure: summarise coverage, defects, residual risks, lessons learned, and sign-off. In Agile these activities repeat each sprint.

11. Defect life cycle

A typical lifecycle is New/Logged → Assigned → Open/In Progress → Fixed/Resolved → Retest → Verified/Closed. If the fix fails, the defect is Reopened. Other possible states include Rejected/Invalid, Duplicate, Deferred, Cannot Reproduce, and Won't Fix. Exact states depend on the team's workflow. A defect report should include steps, expected/actual results, environment/build, severity/priority, and evidence.

12. Instantiate ChromeDriver

In Selenium 4, Selenium Manager can resolve a compatible driver in supported environments. Configure ChromeOptions when needed, create the driver, and quit it in teardown. Browser and driver compatibility plus network restrictions can affect setup.

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--start-maximized");
WebDriver driver = new ChromeDriver(options);
driver.get("https://example.com");
// In teardown: driver.quit();
```

13. Client calls with onsite team

Answer truthfully. If yes, describe the purpose (requirement clarification, defect triage, demo, status update), your contribution, and the outcome. If no, say you have mainly coordinated through your delivery team and are comfortable joining client calls; give an example of stakeholder communication you have handled. Never invent client-facing experience.

14. Maven, Git, Jenkins

Maven manages dependencies and build lifecycle through `pom.xml`, and can compile/package/run tests. Git is distributed version control used to track changes, branch, merge, and review code. Jenkins is a CI automation server that can check out code, run Maven tests, publish results, and notify the team. Together: a code change is committed to Git; Jenkins checks it out and runs `mvn clean test`; results and reports are published.

15. Round 2: Limitations of Selenium

Selenium automates web browsers; it does not directly automate native desktop applications. It does not include a complete test runner, reporting platform, or test-data management solution, so teams integrate TestNG/JUnit, reporting tools, and build systems. CAPTCHA/OTP are security controls and should be handled through approved test hooks or test environments, not bypassed in production. Dynamic DOMs, timing, browser differences, and remote execution can create flaky tests unless locators, waits, data, and sessions are designed carefully.

16. Code for taking screenshot

Cast the driver to `TakesScreenshot`, capture a temporary image file, create the target folder, and copy the image to the chosen path. Use unique names for repeated/parallel tests.

```
File source = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
Path target = Path.of("test-output", "failure.png");
Files.createDirectories(target.getParent());
Files.copy(source, target, StandardCopyOption.REPLACE_EXISTING);
```

17. Handle JavaScript alert

Use `driver.switchTo().alert()` to accept, dismiss, read text, or enter a prompt value. Wait for the alert to be present when it appears asynchronously. This is for browser JavaScript alerts, not HTML modal dialogs.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
System.out.println(alert.getText());
alert.accept(); // or alert.dismiss()
```

18. Handle multiple windows

Store the parent handle, trigger the child window, wait for handle count to increase, switch to the handle that differs from the parent, perform the action, then return to the parent. If several children can open, identify the correct one by title or URL rather than assuming handle order.

```
String parent = 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(parent)) {
        driver.switchTo().window(handle);
        // interact with child window
        break;
    }
}
driver.switchTo().window(parent);
```

19. Annotations in TestNG

Common annotations: `@BeforeSuite`/`@AfterSuite`, `@BeforeTest`/`@AfterTest`, `@BeforeClass`/`@AfterClass`, `@BeforeMethod`/`@AfterMethod`, `@Test`, `@DataProvider`, `@Parameters`, `@Factory`, `@Listeners`, and `@BeforeGroups`/`@AfterGroups`. Setup/teardown annotations run at different scopes. `@Test` marks a test method; `@DataProvider` supplies datasets. The exact order should be verified against the suite structure and TestNG version.

20. Hierarchy/order of TestNG annotations

At a high level: suite setup (`@BeforeSuite`), test-tag setup (`@BeforeTest`), group setup (`@BeforeGroups` when applicable), class setup (`@BeforeClass`), method setup (`@BeforeMethod`), test method (`@Test`), then matching after-method, after-class, after-test, and after-suite methods. Group hooks only apply to relevant groups, and factories/data providers can affect invocation counts. Do not confuse `@BeforeTest` with “before every @Test”: it runs before the `<test>` section in testng.xml.

21. Defect retesting vs regression testing

Retesting (confirmation testing) reruns the failed test after a specific defect fix to confirm the original issue is resolved. Regression testing runs related or broader existing tests to ensure the change has not broken other functionality. Example: after fixing a login lockout bug, retest the lockout steps; regression might include valid login, password reset, session timeout, and role-based access.

22. Database testing

Database testing checks data integrity, constraints, CRUD operations, stored procedures, transactions, migration, and consistency between UI/API and database. Use SQL to verify records and joins, confirm primary/foreign keys, test null/duplicate rules, and validate rollback/commit behaviour where applicable. Use read-only access and approved test environments; avoid changing production data. Example: submit an order through the UI, then verify the order row and status in the test database.

```
SELECT order_id, status
FROM orders
WHERE order_id = 10025;
```

23. Which project is in which domain?

Explain the business domain, product users, and important workflows rather than only naming the project. Examples: banking—accounts, payments, lending; travel/visa services—application submission, appointment booking, document scanning and verification; retail—catalogue, cart, payment, fulfilment. Use the domain that genuinely matches your project, and explain two business rules you tested.

24. Run smoke tests alone from 100 mixed scenarios using TestNG

Use TestNG groups: mark smoke methods with `groups = {"smoke"}` and run only that group through `testng.xml` or Maven Surefire configuration. Keep smoke tests independent and fast. Sanity is a separate group if the team defines it separately.

```
@Test(groups = {"smoke"})
public void loginSmokeTest() { }

@Test(groups = {"sanity"})
public void updatedProfileSanityTest() { }

// testng.xml: include only smoke group
// <groups><run><include name="smoke"/></run></groups>
```

25. Reports maintained in a project

Answer according to your project. Common outputs include TestNG/JUnit execution reports, Cucumber HTML/JSON reports, ExtentReports, screenshots for failures, defect dashboards, daily execution status, traceability/coverage reports, and CI build reports. Explain what the report shows, who uses it, and how failures are investigated. Avoid claiming a reporting tool you did not use.

26. Tell me about yourself (second version)

For a second-round interview, make it more technical: “My background combines functional testing and automation. I use Java and Selenium to build maintainable browser tests, with page objects, explicit waits, data-driven execution, assertions, and reporting. I also participate in requirement analysis, defect triage, regression planning, and Agile ceremonies. I focus on making automation reliable and useful for release decisions rather than just increasing script count.” Add one real achievement if you can substantiate it.

27. Read Excel using Apache POI

Apache POI reads Excel workbooks. Use `WorkbookFactory` to support `.xls` and `.xlsx`, then access sheet/row/cell. Handle blank cells and types carefully; for dates/formatted values, `DataFormatter` often provides a display string. Always close the workbook/input stream with try-with-resources.

```
try (InputStream in = Files.newInputStream(Path.of("testdata.xlsx"));
     Workbook workbook = WorkbookFactory.create(in)) {
    Sheet sheet = workbook.getSheet("Registration");
    DataFormatter formatter = new DataFormatter();
    Row row = sheet.getRow(1); // second row, after header
    String firstName = formatter.formatCellValue(row.getCell(0));
    String email = formatter.formatCellValue(row.getCell(2));
    System.out.println(firstName + " / " + email);
}
```

28. What annotations are available in TestNG?

See Question 19 for the common lifecycle annotations. In addition, `@Test` attributes include `groups`, `dependsOnMethods`, `priority`, `dataProvider`, `enabled`, `timeOut`, `invocationCount`, `expectedExceptions`, and `successPercentage` (with caveats). Use annotations to express test metadata and lifecycle, not as a substitute for clear test design.

29. When to use successPercentage in TestNG

`successPercentage` defines an allowed success rate for repeated invocations of a test, often alongside `invocationCount`. It is rarely appropriate for deterministic functional tests because a flaky test should normally be fixed rather than accepted. It can be useful for a deliberately probabilistic or reliability-style test when the team has defined an acceptable pass threshold. Explain the risk of masking intermittent failures.

```
@Test(invocationCount = 10, successPercentage = 90)
public void transientServiceCheck() {
    // Use only when intermittent outcomes are an intentional test design.
}
```

30. Defect lifecycle (repeat)

Explain the flow: logged/new → triage/assigned → open/in progress → fixed/resolved → retest → verified/closed. If it still fails, reopen it. Other states depend on the tool and process. Severity describes impact; priority describes urgency. Include evidence, clear reproduction steps, environment/build, expected/actual results, and logs/screenshots.

31. Click a link in another window

Switch to the target window before finding/clicking the link. Store the original handle, wait for the new window, switch, then locate the link in that window. If the link itself opens a third window, repeat the same pattern. You cannot interact with elements in a non-current window context.

```
String parent = driver.getWindowHandle();
driver.findElement(By.id("openChild")).click();
new WebDriverWait(driver, Duration.ofSeconds(10))
    .until(d -> d.getWindowHandles().size() > 1);

String child = driver.getWindowHandles().stream()
    .filter(h -> !h.equals(parent)).findFirst()
    .orElseThrow(() -> new NoSuchWindowException("Child window not found"));
driver.switchTo().window(child);
driver.findElement(By.linkText("Continue")).click();
driver.switchTo().window(parent);
```

32. Selenium locators: least and most preferred

Selenium's standard locator strategies are `id`, `name`, `className`, `tagName`, `linkText`, `partialLinkText`, `cssSelector`, and `xpath` (eight). There is no universal “least prepared” or “most preferred” locator ranking: stability and uniqueness matter. A stable unique ID or dedicated test attribute is usually easiest to maintain. Absolute XPath is often fragile; relative XPath can be useful for relationships. Choose the locator that uniquely identifies the element and survives UI changes.

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

33. Challenges faced in Selenium

Choose real examples and explain problem → root cause → action → outcome. Common issues: flaky timing, changing DOM/locators, overlays, test data conflicts, OTP/captcha restrictions, browser compatibility, and parallel execution. Prefer explicit waits, stable locators, isolated data, diagnostics, and collaboration with developers over adding fixed sleeps. Mention only challenges you have genuinely faced.

34. Report used in your project

State the actual report tool, e.g., TestNG default reports, Cucumber HTML, ExtentReports, or CI test-result publishing. Explain how a report shows passed/failed/skipped tests, error details, execution time, and screenshots/logs where configured. If you used multiple reports, describe the audience for each (developer debugging vs stakeholder status).

35. BDD, TDD, and DDD

BDD (Behaviour-Driven Development) describes behaviour using examples understandable to business and technical stakeholders; Gherkin Given/When/Then is commonly used. TDD (Test-Driven Development) writes a failing test first, then code to pass it, then refactors. DDD (Domain-Driven Design) structures software around a shared domain model and ubiquitous language. They are different concepts and can coexist; Cucumber is commonly associated with BDD, not automatically with TDD or DDD.

36. Explain a feature file

A Cucumber feature file is a `.feature` text file written in Gherkin. It describes a feature and scenarios using business-readable steps. Step definitions bind the steps to Java code. Keep scenarios focused on observable behaviour and avoid embedding implementation details or excessive UI mechanics.

```
Feature: Registration
  Scenario: Register with valid details
    Given the user opens the registration page
    When the user enters valid registration details
    And submits the registration form
    Then a registration confirmation should be displayed
```

37. Keywords in a Gherkin feature file

Common keywords: `Feature`, `Rule`, `Background`, `Scenario`, `Scenario Outline`, `Examples`, `Given`, `When`, `Then`, `And`, `But`, and `*`. `Given` sets context, `When` describes an action/event, and `Then` describes an expected outcome. `Background` steps run before each scenario in the feature; `Scenario Outline` runs the same scenario with rows from `Examples`. `@tag` labels scenarios for selection.

38. Pass parameters from outside a class

Common ways are constructor parameters, method parameters, configuration files/environment variables, dependency injection, or TestNG `@Parameters`/`@DataProvider`. Prefer explicit constructor/method injection for dependencies and configuration for environment-specific settings. Avoid mutable global variables where possible because they make tests harder to isolate.

```
class LoginPage {
    private final WebDriver driver;
    LoginPage(WebDriver driver) { this.driver = driver; }
    void enterUser(String username) {
        driver.findElement(By.id("username")).sendKeys(username);
    }
}
// caller passes the driver and username into the class/method.
```

39. findElement vs findElements

`findElement(By)` returns the first matching WebElement and throws `NoSuchElementException` if no match exists. `findElements(By)` returns a list of all matches, or an empty list if none exist. Use `findElement` when one element is required; use `findElements` for counting, iterating, or optional elements. Neither method by itself waits for a specific business condition.

40. What is relative XPath?

Relative XPath starts from any matching node in the DOM using `//`, rather than following the full path from the document root. It is usually more maintainable than an absolute XPath such as `/html/body/div[2]/...`. Use meaningful attributes, text, and relationships, and avoid long index-dependent paths.

```
//input[@id='email']
//button[normalize-space()='Submit']
//label[normalize-space()='Email']/following-sibling::input
```

41. Explain following-sibling

The XPath `following-sibling::` axis selects sibling nodes that appear after the context node and share the same parent. It does not select descendants or elements under a different parent. Add a predicate to select the desired type or index.

```
//label[normalize-space()='Email']/following-sibling::input[1]
//td[normalize-space()='Pending']/following-sibling::td[1]
```

42. Eight Selenium locator strategies with examples

The standard strategies are ID, name, class name, tag name, link text, partial link text, CSS selector, and XPath. Prefer stable unique identifiers. `className` expects one class token, not a string containing multiple classes. Link-text locators apply to anchor links. XPath is powerful for text and DOM relationships but should remain readable.

```
By.id("username")
By.name("password")
By.className("submit-button")
By.tagName("input")
By.linkText("Register")
By.partialLinkText("Reg")
By.cssSelector("input[name='email']")
By.xpath("//button[normalize-space()='Submit']")
```

43. Drag and drop

Use Selenium's `Actions` API. The source and target must be visible and interactable. Some modern HTML5 or JavaScript drag-and-drop widgets may not respond to native actions in every browser; in that case inspect the widget's supported interaction and avoid JavaScript hacks unless justified.

```
WebElement source = driver.findElement(By.id("dragItem"));
WebElement target = driver.findElement(By.id("dropZone"));
new Actions(driver).dragAndDrop(source, target).perform();
```

44. Find total number of URLs/links on a page

If “URLs” means anchor destinations, count anchor elements and optionally read each `href`. Some anchors may have no href, and links may be duplicated. Clarify whether the requirement is number of anchor tags, unique URL values, or only valid HTTP(S) links.

```
List<WebElement> links = driver.findElements(By.tagName("a"));
Set<String> urls = new HashSet<>();
for (WebElement link : links) {
    String href = link.getAttribute("href");
    if (href != null && !href.isBlank()) urls.add(href);
}
System.out.println("Anchor count = " + links.size());
System.out.println("Unique non-empty URLs = " + urls.size());
```

45. Current Java version and previous-version differences

State the version actually used by your project and confirm it from `java -version`, Maven compiler settings, and CI agent configuration. For example, if the project uses Java 17 and the previous LTS was Java 11, discuss features introduced between those versions. Java 17 highlights include records (finalised in 16), sealed classes (finalised in 17), text blocks (finalised in 15), switch expressions (finalised in 14), and pattern matching for `instanceof` (finalised in 16). Java 17 also removed the old Applet API and Security Manager was deprecated for removal. Do not say your project uses the latest version unless verified.

```
java -version
javac -version
// Check pom.xml maven-compiler-plugin source/target or release settings.
```

46. Given an array

The prompt is incomplete: the interviewer likely supplied an array and asked for a particular operation (sort, reverse, find max, remove duplicates, etc.). Ask what output is expected. A common baseline example is to iterate through an array and print each value.

```
int[] values = {4, 2, 7, 2};
for (int value : values) {
    System.out.println(value);
}
```

47. Create a Jenkins build

Create a Freestyle or Pipeline job, connect source control, configure credentials, choose an agent, run build/test commands, and publish reports. Prefer Pipeline-as-Code with a Jenkinsfile checked into the repository. Test the job manually, then configure webhook or schedule triggers. Store secrets in Jenkins credentials, not source code.

```
pipeline {
  agent any
  stages {
    stage('Checkout') { steps { checkout scm } }
    stage('Build and Test') { steps { sh 'mvn clean test' } }
  }
  post {
    always { junit '**/target/surefire-reports/*.xml' }
  }
}
```

48. Jenkins plugins required

Plugins depend on the pipeline and organisation. Common choices are Pipeline, Git, Credentials Binding, JUnit (often bundled), and plugins for the specific report format or notifications. Maven Integration may be useful for some job styles. Install only approved, maintained plugins, review permissions and updates, and avoid installing unnecessary plugins because each adds maintenance/security exposure.

49. Version control software used in projects

If you use Git, explain repository, working tree, staging, commit, push, pull/fetch, branch, merge/rebase, pull request, and conflict resolution. Typical flow: create a feature branch, commit small changes, push, open a pull request, get review, resolve conflicts, and merge according to team policy. Keep secrets and generated test artifacts out of the repository.

50. What is Bitbucket?

Bitbucket is a Git-based source code hosting and collaboration platform. Teams use it for repositories, branches, pull requests, code review, access controls, and integrations with build systems such as Jenkins. It hosts Git repositories; Git is the version-control system itself. Bitbucket Cloud and Bitbucket Data Center have different hosting and administration models.

51. Prime number program

A prime number is an integer greater than 1 with no positive divisors other than 1 and itself. It is enough to test divisors up to the square root of the number. Handle values less than 2 as not prime.

```
int n = 29;
boolean prime = n >= 2;
for (int i = 2; i <= Math.sqrt(n) && prime; i++) {
    if (n % i == 0) prime = false;
}
System.out.println(prime ? "Prime" : "Not prime");
```

52. Change lowercase to uppercase and uppercase to lowercase

Use `Character.isLowerCase`, `Character.isUpperCase`, and the corresponding conversion methods. Preserve digits and punctuation unchanged. This example handles Java `char` values; advanced Unicode text may need code-point-aware processing.

```
String input = "Java QA 123!";
StringBuilder output = new StringBuilder();
for (char ch : input.toCharArray()) {
    if (Character.isLowerCase(ch)) output.append(Character.toUpperCase(ch));
    else if (Character.isUpperCase(ch)) output.append(Character.toLowerCase(ch));
    else output.append(ch);
}
System.out.println(output); // jAVA qa 123!
```

53. Open browser with customised settings

Use browser-specific Options classes to configure arguments and preferences before creating the driver. Common settings include headless mode, download directory, browser language, proxy, certificate handling, and window size. Apply only settings approved for the test environment; disabling security checks can hide real issues.

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--start-maximized");
options.addArguments("--lang=en-US");
// options.addArguments("--headless=new"); // for headless execution
Map<String, Object> prefs = new HashMap<>();
prefs.put("download.default_directory", "C:\\test-downloads");
options.setExperimentalOption("prefs", prefs);
WebDriver driver = new ChromeDriver(options);
```

54. Go directly to homepage if previous session was logged in

This depends on how the application manages sessions. If a valid session cookie is persisted by the browser, the application may redirect directly to the home page; test this through supported application behaviour. Selenium cookies can be saved and restored in a controlled test, but cookies are authentication secrets and must never be committed or shared. A safer approach is often API-based login or a dedicated test-session mechanism provided by the application. Avoid bypassing production authentication controls.

```
// Example only for an approved test environment:
driver.get("https://example.test");
for (Cookie cookie : savedTestCookies) {
    driver.manage().addCookie(cookie);
}
driver.navigate().refresh();
// Verify that the authenticated home page is displayed.
```

55. Secure Maven JAR artifact before pushing to Git

Do not commit generated JARs to source Git unless the repository policy explicitly requires it; publish artifacts to an approved artifact repository. Security steps include dependency vulnerability scanning (for example, approved SCA tooling), secret scanning, malware scanning of externally supplied files, signed/checksummed artifacts where supported, protected branches and pull-request review, least-privilege tokens, and CI controls. Keep credentials in a secret manager/Jenkins credentials. A JAR is not made safe simply by putting it in Git; validate dependencies, build provenance, and artifact integrity.

56. Access modifiers

`public` is accessible from anywhere allowed by module/package rules; `protected` is accessible within the same package and by subclasses (with Java's subclass access rules); package-private (no modifier) is accessible only within the same package; `private` is accessible only within the declaring class. Use the narrowest access that satisfies the design.

57. Can a protected method be accessed from a subclass?

Yes. A subclass can access an inherited protected method, including from another package, subject to Java's protected-access rules. From another package, access is through inheritance and the qualifying reference must be appropriate to the subclass context; it is not unrestricted access to any arbitrary superclass instance.

```
package a;
public class Parent {
    protected void show() { System.out.println("Parent"); }
}

// package b:
class Child extends a.Parent {
    void test() {
        show(); // accessible through inheritance
    }
}
```

58. Move browser window to a different position

Use WebDriver's window management API. Coordinates are pixels from the screen's top-left, but window managers, remote grids, and headless browsers may ignore or constrain positioning.

```
driver.manage().window().setPosition(new Point(100, 80));
driver.manage().window().setSize(new Dimension(1200, 800));
```

59. XPath text inside a paragraph inside a div

If the text is in a `<p>` nested inside a `<div>`, select the paragraph by its text or by a stable parent relationship. `normalize-space(.)` checks combined descendant text and normalises whitespace. If you want the paragraph element itself, use the first expression; use `getText()` to read the rendered text.

```
//div/p[normalize-space(.)='Your target paragraph text']
//div[@id='content']//p[contains(normalize-space(.), 'target phrase')]

String text = driver.findElement(
    By.xpath("//div[@id='content']//p[contains(normalize-space(.), 'target phrase')]")
).getText();
```

60. Set vs List

List preserves sequence and allows duplicates and index access. Set models uniqueness and has no index-based access. ArrayList is a common List; HashSet is common for uniqueness but has no guaranteed order; LinkedHashSet preserves insertion order; TreeSet sorts elements. Choose based on the requirement.

```
List<String> list = Arrays.asList("A", "A", "B");
Set<String> unique = new LinkedHashSet<>(list);
System.out.println(list.size());   // 3
System.out.println(unique.size()); // 2
```

61. StringBuffer vs StringBuilder

Both are mutable character sequences. StringBuffer synchronizes methods and is thread-safe for individual operations; StringBuilder is not synchronized and is usually faster when only one thread uses it. Use StringBuilder for loops that build strings; use StringBuffer only when its synchronization fits the shared-access need.

```
StringBuilder sb = new StringBuilder("Test");
sb.append(" Automation");
System.out.println(sb);
```

62. Interface

An interface defines a contract that implementing classes must fulfil. A class can implement multiple interfaces but extend only one class. Interfaces support loose coupling: code can depend on a contract rather than a concrete implementation. In Selenium, WebDriver is an interface implemented by browser-specific drivers.

```
WebDriver driver = new ChromeDriver();
driver.get("https://example.com");
System.out.println(driver.getTitle());
driver.quit();
```

63. Are public static void main and static public void main both valid?

Yes. Java permits modifiers such as `public` and `static` in either order in a method declaration, so both are valid. The conventional and most commonly used order is `public static void main(String[] args)`. The method is the standard entry point for launching a Java application.

```
public static void main(String[] args) {
    System.out.println("Hello");
}

// Also valid:
static public void main(String[] args) {
    System.out.println("Hello");
}
```

Final revision tips

- Practise the registration workflow and explain each line of code.
- Be prepared to explain why explicit waits are safer than fixed Thread.sleep calls.
- For project answers, describe the business flow, your contribution, tools, challenges, and evidence of impact.
- For Jenkins/Git/security questions, distinguish what you have personally configured from what you understand conceptually.
- For incomplete questions such as “Given an array”, ask what output or operation is expected.