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

Elaborate answers to the 58 interview questions

Profile basis for sample answers: around 13 years in software testing; experience with manual and automation testing, approximately 5 years with Selenium/Java, 1 year with Playwright, 1 year ETL testing and 6 months JMeter; current framework described as Cucumber + POM hybrid. Personalise the sample answers with your actual project, tools and achievements. Do not claim experience or metrics that you cannot verify.

A. Introduction and testing fundamentals

Can you brief about yourself?

I have around 13 years of experience in software testing, covering manual testing, automation and QA activities across the SDLC. My core automation experience includes Java, Selenium WebDriver, TestNG, Cucumber and a Page Object Model hybrid framework. I have also worked with Playwright, ETL testing and JMeter, and I use requirement analysis, test design, execution, defect management and regression testing to support releases. I collaborate with developers, business analysts and QA colleagues in Agile teams, investigate failures and focus on maintainable automation and meaningful coverage. I am looking for a role where I can contribute to automation strategy, framework improvement and QA leadership. Add your actual project/domain and one genuine achievement.

1. HashMap vs Hashtable

Both store key-value pairs. HashMap is not synchronized, is generally faster for single-threaded use, and permits one null key and multiple null values. Hashtable is a legacy synchronized class and does not permit null keys or values. For concurrent access, ConcurrentHashMap is usually a better modern choice. In both maps, keys rely on consistent hashCode() and equals() implementations.

2. StringBuffer vs String

String is immutable: a modification creates a new String object. StringBuffer is mutable and can append, insert or delete characters in the same buffer; its methods are synchronized. For repeated concatenation in single-threaded code, StringBuilder is usually preferred because it is mutable without synchronization overhead. StringBuffer and String are separate classes.

3. What is testing?

Software testing evaluates a product to find defects and provide evidence about quality, risk and fitness for use. It includes analysing requirements, designing tests, preparing data, executing tests, comparing actual with expected results, logging defects and reporting risk. Testing cannot prove there are no defects; it reduces the likelihood and impact of failures. Testing can be static, such as reviews, or dynamic, by executing software.

4. Different types of testing

By level: unit, integration, system and acceptance. By purpose: functional, regression, smoke, sanity, exploratory and confirmation testing. Non-functional types include performance, security, compatibility, usability, accessibility, reliability, scalability, recovery, installation and localization testing. Testing can also be black-box, white-box or grey-box. The chosen types depend on the application's risks and requirements.

5. Testing apart from functional and performance

Examples include security, compatibility, usability, accessibility, reliability, scalability, stress, endurance/soak, recovery, installation/upgrade, localization, interoperability, failover, backup/restore, data migration, API, database, exploratory, regression, smoke, sanity, acceptance and compliance testing. Select tests based on business and technical risk.

6. Security testing

Security testing checks whether an application protects data and services from unauthorized access and misuse. It covers authentication, authorization, sessions, input validation, encryption, sensitive-data exposure, access control and configuration. QA examples include verifying role-based access, rejecting invalid API tokens, preventing one user from viewing another user's records, and checking session expiry. Penetration testing and scanning should be authorized and performed in controlled environments.

7. Where do you see yourself in two years?

In two years, I see myself taking broader ownership of QA delivery—leading test planning, improving automation coverage and reliability, mentoring team members and contributing to release-risk decisions. I want to deepen my Java/Selenium and Playwright skills, strengthen API and CI/CD integration, and grow toward QA Lead or Test Manager responsibilities while remaining technically hands-on.

8. Compatibility testing

Compatibility testing verifies that an application works correctly across supported browsers, browser versions, operating systems, devices, screen sizes, databases and integrations. For a web application, I would create a risk-based browser/device matrix and validate critical workflows, rendering, downloads, responsive layout and browser-specific behaviour. Testing every possible combination is usually impractical, so production usage and risk guide selection.

9. Code coverage

Code coverage measures how much source code is exercised by tests. Measures include statement, branch/decision, condition and method coverage. For example, 80% statement coverage means 80% of measured statements were executed. Coverage does not prove test quality: a line may execute without meaningful assertions. Combine coverage with risk analysis, requirements coverage, defect trends and assertion quality. JaCoCo is commonly used for Java.

10. Why are you looking for a change?

I appreciate the opportunities in my current organisation. At this stage, I am looking for broader technical ownership, opportunities to improve automation strategy and framework design, and a path to contribute to QA delivery and mentoring. I want to apply my Java, Selenium and Cucumber experience to new quality challenges. My decision is based on career growth and role alignment, rather than negative feelings about my current employer.

11. Project achievements

Use a real situation-action-result example. For instance, if repeated regression took significant manual effort, explain how you selected stable high-value scenarios, automated them, improved reporting and helped the team get feedback sooner. Other achievements might include finding a critical defect before production, reducing flaky tests, improving test data setup or mentoring colleagues. Add metrics only when you can substantiate them, such as scenarios automated or execution time reduced.

12. Project challenges

Typical challenges include changing requirements, unstable environments, test-data constraints, dynamic UI elements, asynchronous updates, flaky scripts and tight regression windows. Explain one genuine case using Situation, Task, Action and Result. For example, investigate a failing locator, choose a stable selector, wait for the correct UI state, isolate test data and rerun tests to confirm the root cause was addressed.

B. Java concepts and coding

13. Rate yourself in Java

A sample answer is 3.5–4 out of 5, if accurate: I am comfortable with OOP, collections, strings, arrays, exception handling, methods, file operations and writing automation utilities. I continue to deepen my knowledge of concurrency, JVM internals, streams and advanced design patterns. Be prepared to justify the rating with code.

14. Garbage collector in Java

The garbage collector reclaims heap memory occupied by objects that are no longer reachable from GC roots. Java developers generally do not free objects manually. Setting a reference to null does not guarantee immediate collection, and System.gc() is only a request, not a guarantee. GC does not close files or database connections; use try-with-resources or explicit close methods. Memory leaks can still happen when references are unintentionally retained.

15. Multiple inheritance in Java

Java does not allow a class to extend multiple classes, partly to avoid ambiguity in inherited implementations. A class can implement multiple interfaces. Interfaces can have default methods, and if two interfaces provide conflicting defaults, the class must override the method and resolve the conflict. Composition is another common alternative.

16. Interface vs abstract class

An interface defines a contract and a class can implement multiple interfaces. It can contain abstract, default, static and private methods depending on Java version. An abstract class can hold instance state, constructors, concrete methods and abstract methods, but a class can extend only one class. Use interfaces for capabilities/contracts across different classes; use abstract classes when related classes need shared state or implementation.

17. Prime number program

A prime is an integer greater than 1 divisible only by 1 and itself. Testing divisors up to the square root is sufficient.

```
public static boolean isPrime(int n) {
    if (n < 2) return false;
    for (int i = 2; i <= n / i; i++) {
        if (n % i == 0) return false;
    }
    return true;
}
public static void main(String[] args) {
    System.out.println(isPrime(29)); // true
}
```

18. Reasoning and probability

For equally likely outcomes, probability = favourable outcomes / total outcomes. A fair die has three even outcomes (2, 4, 6) out of six, so P(even)=3/6=1/2. For two fair coin tosses, outcomes are HH, HT, TH, TT; exactly one head has probability 2/4=1/2. For 'at least one', use the complement: P(at least one)=1−P(none). For reasoning questions, state assumptions and work step by step.

19. How do you update your knowledge?

I learn from official documentation, release notes, technical articles, code reviews and hands-on practice. I build small proof-of-concepts, investigate automation failures and discuss approaches with teammates. Before adopting a tool, I consider reliability, maintainability, integration, team skills and business value. Mention only topics you have genuinely studied recently.

20. What is STLC?

STLC means Software Testing Life Cycle. A typical flow includes requirement analysis, test planning, test case development, environment setup, execution, defect retesting/regression and test closure. Phases may overlap in Agile projects and should be adapted to the delivery model.

21. Outputs of each STLC component

Requirement analysis: clarified requirements, risks and traceability inputs. Planning: scope, strategy, estimates, roles, schedule, tools and entry/exit criteria. Test development: scenarios, test cases, data and RTM updates. Environment setup: configured environment, build details and smoke results. Execution: results, evidence, defects and updated traceability. Retest/regression: verified fixes and regression outcomes. Closure: summary report, coverage, open risks, metrics and lessons learned.

22. Innovation in the past three years

Choose a real improvement and explain the problem, your idea, implementation and outcome. Example structure: repeated manual checks were slow and inconsistent; you identified stable high-value scenarios, automated them using reusable methods and improved failure logs; the team then received more repeatable

Describe an example from your own work and support claimed improvements with evidence.

C. Selenium, TestNG and framework

23. QA automation tools

Tools may include Selenium or Playwright for UI automation; Java with TestNG/JUnit and Cucumber; Maven for build/dependencies; Git for version control; Jenkins or another CI tool; Postman or Rest Assured for APIs; SQL clients for database checks; Jira and test management tools; and Extent Reports or Allure. Describe how the tools work together in your actual project and only claim hands-on experience you have.

24. What is Selenium?

Selenium is an open-source suite for browser automation. WebDriver drives supported browsers through browser automation interfaces, and Selenium Grid can distribute tests across machines and browser configurations. Selenium supports several languages and browsers but is primarily for web applications. Stable locators, waits, assertions, test data and maintainable framework design are essential to reliable Selenium automation.

25. Explain your Selenium framework

My current framework is a Cucumber + POM hybrid. Gherkin feature files describe scenarios, step definitions connect steps to Java code, and page classes encapsulate locators and page actions. Utilities handle common tasks such as waits, dropdowns and configuration; hooks manage setup/teardown; Maven manages dependencies and execution; Git supports collaboration; and reporting captures outcomes and evidence. Mention only components actually present in your framework.

26. Repository used for objects

If 'object repository' means locator storage, in POM locators are commonly maintained in page classes alongside page actions, rather than in one central repository file. Some frameworks store locators in properties, JSON or YAML. If the interviewer means source-code repository, answer with the actual tool your team uses, such as Git. Clarify the intended meaning if necessary.

27. WebDriver script for Gmail login

For an interview, demonstrate a generic login flow on an application you own or are authorised to test. Real Gmail sign-in may involve MFA, CAPTCHA and anti-automation protections; never hard-code credentials.

Use URLs and locators from a controlled test environment.

```
WebDriver driver = new ChromeDriver();
try {
    driver.manage().window().maximize();
    driver.get("https://your-test-app.example/login");
    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.cssSelector("button[type='submit']")).click();
    wait.until(ExpectedConditions.urlContains("/dashboard"));
} finally {
    driver.quit();
}
```

28. TestNG annotations

Common annotations: @Test, @BeforeSuite/@AfterSuite, @BeforeTest/@AfterTest, @BeforeClass/@AfterClass, @BeforeMethod/@AfterMethod, @BeforeGroups/@AfterGroups, @DataProvider, @Parameters, @Factory, @Listeners and @Optional. Test attributes include groups, priority, dependsOnMethods, invocationCount, timeOut and enabled.

29. Use of @BeforeTest

@BeforeTest runs before test methods associated with the relevant <test> element in testng.xml. It is useful for setup shared by classes in that XML test. @BeforeMethod runs before each test method; @BeforeClass runs before the first test method in a class. Choose the annotation according to the intended setup scope.

30. Access methods with parameters in TestNG

Use @Parameters for named values declared in testng.xml, such as browser or base URL, and @DataProvider for multiple data sets. @Optional can provide a default for a missing XML parameter.

```
@Test
@Parameters({"browser", "baseUrl"})
public void openApplication(String browser, String baseUrl) {
    System.out.println(browser + " : " + baseUrl);
}
```

31. Run first six classes in parallel and remaining four sequentially

Put the first six classes in one XML <test> with parallel='classes' and thread-count='6'; put the other four in a second <test> with parallel='none'. Keep suite-level parallelism disabled so the two <test> groups do not overlap. Confirm the first group’s tests are independent and WebDriver/test data are thread-safe.

32. TestNG XML with parallel='tests' and 'classes'

For the requested sequential groups, use suite parallel='none', parallel='classes' for the first group, and parallel='none' for the second. parallel='tests' runs separate XML <test> groups concurrently, so it conflicts with the requirement that group two wait for group one. Example:

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="GroupedExecution" parallel="none">
  <test name="FirstSixParallel" parallel="classes" thread-count="6">
    <classes>
      <class name="tests.Class1"/><class name="tests.Class2"/>
      <class name="tests.Class3"/><class name="tests.Class4"/>
      <class name="tests.Class5"/><class name="tests.Class6"/>
    </classes>
  </test>
  <test name="RemainingFourSequential" parallel="none">
    <classes>
      <class name="tests.Class7"/><class name="tests.Class8"/>
      <class name="tests.Class9"/><class name="tests.Class10"/>
    </classes>
  </test>
</suite>
```

33. Parameterization in TestNG

Use @Parameters for run configuration from testng.xml and @DataProvider for multiple test data rows. DataProvider often returns Object[][] and causes the same test method to execute once per row. Keep passwords and tokens in environment variables or a secrets manager, not source code.

34. DataProvider vs XML parameters

@Parameters supplies named configuration values such as browser or environment. @DataProvider supplies multiple data sets to a test, such as different users, inputs and expected results. Use XML parameters for execution settings and DataProvider for data-driven testing.

35. Dependency tests in TestNG

Use dependsOnMethods or dependsOnGroups when a test genuinely requires a prerequisite. If the prerequisite fails, the dependent test is normally skipped. Avoid long dependency chains because they make suites fragile; prefer independent tests with their own setup where possible. Ensure cleanup runs reliably.

36. String palindrome program

A palindrome reads the same forwards and backwards. This example ignores case, spaces and punctuation.

```
public static boolean isPalindrome(String s) {
    if (s == null) return false;
    String clean = s.replaceAll("[^A-Za-z0-9]", "").toLowerCase();
    return clean.equals(new StringBuilder(clean).reverse().toString());
}
// isPalindrome("Madam") -> true
```

37. Integer palindrome

Reverse the digits and compare with the original. Negative numbers are treated as not palindromes in this example.

```
public static boolean isIntegerPalindrome(int n) {
    if (n < 0) return false;
    int original = n;
    long reversed = 0;
    while (n > 0) {
        reversed = reversed * 10 + n % 10;
        n /= 10;
    }
    return original == reversed;
}
// 12321 -> true; 123 -> false; 0 -> true
```

38. HashMap and Hashtable sample

HashMap allows a null key and null values; Hashtable does not. Hashtable synchronizes its methods but is a legacy choice.

```
Map<String, Integer> map = new HashMap<>();
map.put("Java", 1);
map.put("Java", 3); // replaces prior value
map.put(null, 10);  // permitted by HashMap

Hashtable<String, Integer> table = new Hashtable<>();
table.put("Java", 1);
System.out.println(map.get("Java"));   // 3
System.out.println(table.get("Java")); // 1
// table.put(null, 10); // NullPointerException
```

39. Sample code for threads

Calling start() schedules a new thread; calling run() directly does not start a separate thread. Avoid sharing a non-thread-safe WebDriver instance between parallel tests.

```
Runnable task = () -> System.out.println(
    "Running on " + Thread.currentThread().getName());
Thread t1 = new Thread(task, "worker-1");
Thread t2 = new Thread(task, "worker-2");
t1.start();
t2.start();
t1.join();
t2.join();
```

40. Given array {1,2,2,3,4,4,5,5,6,7}

Use sets or a frequency map to identify duplicate values. The array's unique duplicate values are 2, 4 and 5; values occurring exactly once are 1, 3, 6 and 7.

41. Store duplicates in a separate array

This solution preserves first-seen order and adds each duplicate value only once.

```
int[] a = {1,2,2,3,4,4,5,5,6,7};
Set<Integer> seen = new LinkedHashSet<>();
Set<Integer> duplicates = new LinkedHashSet<>();
for (int x : a) {
    if (!seen.add(x)) duplicates.add(x);
}
int[] duplicateArray = duplicates.stream()
    .mapToInt(Integer::intValue).toArray();
// {2, 4, 5}
```

D. Experience, framework and scenario questions

42. Store non-duplicates in another array

If non-duplicates means values occurring exactly once, the result is {1, 3, 6, 7}. A LinkedHashMap preserves input order.

```
int[] a = {1,2,2,3,4,4,5,5,6,7};
Map<Integer,Integer> counts = new LinkedHashMap<>();
for (int x : a) counts.put(x, counts.getOrDefault(x, 0) + 1);
int[] unique = counts.entrySet().stream()
    .filter(e -> e.getValue() == 1)
    .mapToInt(Map.Entry::getKey).toArray();
// {1, 3, 6, 7}
```

43. Professional experience

I have around 13 years of experience in software testing, including manual and automation testing, test planning, execution, defect lifecycle management and regression. My automation background includes Java, Selenium WebDriver, TestNG, Cucumber and POM-based hybrid framework design. I also have experience with Playwright, ETL testing and JMeter. In Agile teams, I clarify requirements, assess risk, investigate failures and work with developers and business stakeholders to support releases. Add your real project/domain and a measurable result.

44. What framework are you currently using?

My current framework is a Cucumber + POM hybrid. Feature files contain Gherkin scenarios, step definitions connect scenarios to Java code, and page classes encapsulate locators and UI actions. Reusable utilities support waits, dropdowns and configuration, while Maven and Git support build execution and collaboration. Align this description with the actual framework repository.

45. Why POM?

POM separates page-specific locators and actions from test logic. When the UI changes, updates are usually isolated to the relevant page class instead of duplicated across tests. It improves reuse, readability and maintenance. POM is a design pattern, not a complete framework; test data, configuration, assertions, reporting and execution still need to be designed.

46. Other frameworks worked with

Be truthful and distinguish hands-on experience from familiarity. You might discuss TestNG-based Selenium automation, reusable utility-driven frameworks, Cucumber/BDD and Playwright if you have used them. For each, explain the project context, your contribution and one advantage or trade-off.

47. Current framework vs previous framework

Compare concrete features: Cucumber can make workflows readable to business stakeholders; POM centralises locators and actions; utilities reduce duplication; configuration supports environments; reports and screenshots help diagnose failures; and CI runs give faster feedback. Also mention trade-offs such as extra step-definition maintenance or overly large page classes. Support the comparison with an actual project example.

48. Rate yourself in Selenium

A sample rating is 4/5 if justified: I can build and maintain WebDriver scripts, select locators, handle waits, frames, windows, alerts, dropdowns and dynamic elements, and integrate tests into a framework. I am continuing to improve in advanced browser behaviour, large-suite optimisation and intermittent-failure diagnosis. Be ready to demonstrate these skills.

49. Types of locators

Standard By strategies are id, name, className, tagName, linkText, partialLinkText, cssSelector and xpath. Selenium 4 also offers relative locators such as above, below, toLeftOf, toRightOf and near. Prefer stable unique attributes; avoid absolute XPath and fragile positional locators where possible.

50. Locators used most

I prefer id or name when stable and unique, CSS selectors for concise attribute matching, and XPath when relationships or text-based selection are needed. Stable test attributes such as data-testid are useful when available. I verify that a locator matches the intended single element and avoid generated IDs that change between runs.

51. Why prefer id?

An ID is usually readable and intended to be unique, so it can be easy to maintain and efficient. However, IDs may be duplicated or generated dynamically. I check stability across sessions and environments; otherwise I use a stable CSS selector, test attribute or carefully designed XPath.

52. Major issues encountered

Common issues include dynamic elements, synchronisation, stale element references, duplicate locators, flaky tests, unstable environments, test-data conflicts, uploads/downloads, iframes, multiple windows, OTP dependencies and changing requirements. Select two or three that you personally faced and explain impact, root cause, action and result.

53. Steps taken to overcome issues

I reproduce the failure and collect screenshots, logs, console/network evidence and test data. I determine whether the cause is a product defect, environment, data or automation. For timing, I use explicit waits for the expected state rather than fixed sleeps; for locators, I select stable attributes and check uniqueness; for stale elements, I re-locate after DOM updates; for data conflicts, I isolate data and clean up. I rerun tests and document the root cause. Increasing timeouts alone is not a robust fix.

54. Rate yourself in Java

Keep this consistent with your answer to question 13. For example, 3.5–4/5 if accurate: I am comfortable with OOP, collections, strings, arrays, exceptions and reusable automation methods, and I am improving in concurrency, streams, JVM behaviour and design patterns. Be ready to code examples.

55. Rate yourself in DB/query formation

A sample is 3/5 if accurate: I can write SELECT queries with WHERE, ORDER BY, GROUP BY, HAVING, joins, aggregate functions and subqueries, and validate application data. I am improving in complex query optimisation, window functions and execution plans. Practise joins, duplicate-row detection and second-highest salary queries.

56. Have you worked in SoapUI?

If yes, discuss WSDL, SOAP/XML requests, headers, authentication, response assertions and test suites. If not, be honest: “My stronger hands-on API experience is with [actual tool]. I understand SoapUI's use for SOAP and REST service testing, including WSDL, XML payloads and response assertions, but I would not claim extensive hands-on experience unless I have used it directly.”

57. Types of websites and design-specific issues

Describe only application types you have actually tested. Single-page apps may have asynchronous rendering and changing DOMs; data grids can involve pagination, sorting and virtual scrolling; booking systems can have slot contention and time-zone issues; responsive sites have layout breakpoints; upload workflows have format/size/network failures; and role-based portals have session and access-control risks. Explain how you chose test cases based on design and business risk.

58. Challenges faced in your project

Use one genuine story in Situation, Task, Action and Result format. Example: asynchronous rendering caused intermittent automation failures. I investigated the failure, replaced fixed sleeps with explicit waits tied to expected states, improved locator stability, and added useful failure evidence. I reran the scenarios to confirm

Describe a genuine incident and include only results you can substantiate.