Aon Hewitt Selenium Interview Questions with Answers (2026) | Saravana Tech

18 interview questions with detailed explanations, project examples, and Java code

Project and HR answers are adaptable examples. Describe tools, responsibilities, outcomes, and architecture that accurately reflect your project. Code examples use Selenium 4 and Java.

Detailed interview answers

1. Explain about your project

Use a 1–2 minute structure: project/business purpose, users and key workflows, technology stack, your role, testing approach, framework, and impact. Avoid confidential customer data. Adapt this sample to the project you actually work on:

“My project supports a business workflow where users submit and process applications/appointments. The QA team validates end-to-end flows, business rules, integrations, and regression scenarios. Our testing includes manual functional testing and UI/API automation. For UI automation we use Java, Selenium WebDriver, Cucumber, TestNG, and a Page Object Model-based framework; test data and configuration are managed separately, and execution results are reported for defect triage and release decisions. I analyse requirements, identify risks, design test cases, maintain automation scripts, investigate failures, raise defects, and coordinate with developers and business stakeholders. I focus on reliable regression coverage and early communication of release risks.”

Only mention tools such as Jenkins, Maven, API automation, or parallel execution if you have genuinely used them. Be ready to explain one real workflow end to end.

2. Roles and responsibilities in your project

Organise your answer into planning, execution, automation, defect management, and coordination. For a QA Lead/Project Lead profile, possible responsibilities include:

- Review requirements and clarify acceptance criteria with product/business analysts.
- Prepare test strategy, estimates, test scenarios, test data, and execution plans.
- Perform or coordinate functional, integration, regression, and exploratory testing.
- Develop and maintain Selenium automation using Java, reusable page objects, and test utilities.
- Review automation failures, investigate whether they are product defects, test-data problems, or environment issues, and log defects with evidence.
- Track test progress, coverage, risks, and blockers; communicate status and release recommendations.
- Work with developers, business stakeholders, and DevOps/CI teams to improve feedback and test stability.

Sample close: “My responsibility is not only to execute tests but also to make risks visible early, keep regression coverage maintainable, and provide evidence-based quality status.” Adjust the list to your actual level of ownership.

3. What is "this" in Java?

`this` is a reference to the current object. It is commonly used to distinguish instance variables from method/constructor parameters with the same name, call another constructor in the same class using `this(...)`, or pass the current object to another method. A `this(...)` constructor call must be the first statement in a constructor.

```
class LoginPage {
    private String pageName;

    LoginPage(String pageName) {
        this.pageName = pageName; // field = constructor parameter
    }

    void printPageName() {
        System.out.println(this.pageName);
    }
}
```

4. Difference between == and equals()

For primitive values, `==` compares values. For object references, `==` checks whether both references point to the same object. `equals()` checks logical/content equality when the class implements it appropriately. String overrides `equals()` to compare character content. Use `Objects.equals(a, b)` for null-safe equality when appropriate.

```
String a = new String("Selenium");
String b = new String("Selenium");

System.out.println(a == b);       // false: different objects
System.out.println(a.equals(b));  // true: same text

int x = 10, y = 10;
System.out.println(x == y);       // true: same primitive value
```

5. How good are you in XPath? Rate yourself out of 5

Choose a rating you can defend with examples. A sample answer, if it reflects your skills: “I would rate myself 4 out of 5. I am comfortable with relative XPath, attributes, text matching, contains(), starts-with(), parent/child relationships, and axes such as following-sibling and preceding-sibling. I use browser DevTools to validate locators and prefer stable, unique locators. I am continuing to improve on complex DOMs and dynamic locator strategies.” If you rate yourself 4/5, be ready to write XPath live.

6. XPath preceding, following, and sibling axes

`following-sibling::` selects later siblings with the same parent. `preceding-sibling::` selects earlier siblings with the same parent. `following::` selects later nodes in document order, not only siblings. `preceding::` selects earlier nodes in document order, excluding ancestors. These axes help when an element has no unique ID but has a stable nearby label or heading.

Example DOM idea: a label “Username” followed by an input as its sibling.

```
// Input after the Username label (same parent):
//label[normalize-space()='Username']/following-sibling::input

// Previous sibling element before a button:
 //button[@id='save']/preceding-sibling::input[1]

// First button later in document order after a heading:
 //h2[normalize-space()='Details']/following::button[1]

// Avoid relying on axes if a stable id or data-testid exists.
```

7. PageFactory: purpose, where used, advantages

PageFactory is a Selenium support mechanism that initialises page-object fields annotated with `@FindBy`. It is used inside page classes to declare element locators in a readable way. It can make page objects concise, but it does not automatically make tests faster or eliminate stale-element problems; modern teams may prefer `By` locators and explicit waits for clearer control. Use the approach already standardised in your framework and avoid mixing styles without a reason.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {
    private final WebDriver driver;

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

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

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

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

    public void login(String user, String pass) {
        username.sendKeys(user);
        password.sendKeys(pass);
        submit.click();
    }
}
```

8. Web table: from cell 1 fetch cell 2 value

Find the row containing the key in cell 1, then read cell 2 from that same row. This is safer than assuming the required row will always be at a fixed index. Add waits if the table is loaded dynamically, and check that the row has enough cells.

```
List<WebElement> rows =
    driver.findElements(By.cssSelector("table tbody tr"));

for (WebElement row : rows) {
    List<WebElement> cells = row.findElements(By.cssSelector("td"));
    if (cells.size() >= 2 &&
        cells.get(0).getText().trim().equals("ORD-1001")) {
        String cell2Value = cells.get(1).getText().trim();
        System.out.println("Cell 2 = " + cell2Value);
        break;
    }
}
```

9. Window handling: move to child and back to parent

Capture the parent window handle before triggering the new window. Wait until a second handle appears, switch to the handle that is not the parent, and then switch back to the stored parent handle. Do not assume that the child is always the second item in a list; identify it by title or URL when multiple windows may open.

```
String parent = driver.getWindowHandle();
driver.findElement(By.id("openChild")).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);
        System.out.println(driver.getTitle());
        // Perform child-window steps here.
        break;
    }
}

driver.switchTo().window(parent);
```

10. Take a screenshot: code and line-by-line explanation

`TakesScreenshot` captures the current browser viewport. `OutputType.FILE` returns a temporary image file; `Files.copy` saves it to a chosen path. Ensure the destination folder exists and use a unique file name in repeated or parallel runs. In remote execution, screenshot support depends on the driver/grid configuration.

```
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;

// 1. Cast the driver to TakesScreenshot.
File source = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

// 2. Define the destination path.
Path destination = Path.of("test-output", "login-failure.png");

// 3. Create the folder if it does not exist.
Files.createDirectories(destination.getParent());

// 4. Copy the temporary screenshot to the destination.
Files.copy(source, destination, StandardCopyOption.REPLACE_EXISTING);
```

11. DataProvider, why use it, Apache POI vs JXL

TestNG `@DataProvider` supplies multiple sets of input data to the same test method. It supports data-driven testing without duplicating test methods. Apache POI is commonly used to read/write modern Excel `.xlsx` and older `.xls` workbooks. JXL (JExcelAPI) is an older library mainly associated with `.xls` and does not support the modern `.xlsx` format. POI is generally preferred for new projects because of broader Excel format support and active ecosystem. If you say you use POI, be ready to explain workbook, sheet, row, and cell handling.

```
@DataProvider(name = "loginData")
public Object[][] loginData() {
    return new Object[][] {
        {"validUser", "validPassword"},
        {"lockedUser", "anyPassword"}
    };
}

@Test(dataProvider = "loginData")
public void loginTest(String username, String password) {
    // Execute login and assert expected result for each data set.
}
```

12. What is HashMap? Where can it be used in a project?

HashMap stores key-value pairs and uses a key to retrieve the associated value. Keys are unique; inserting the same key again replaces its previous value. HashMap permits one null key and null values, does not guarantee iteration order, and is not thread-safe for concurrent mutation. In a test framework, it can hold named test data, map column headers to cell values, or associate expected values with field names. For concurrent shared access, consider ConcurrentHashMap instead.

```
Map<String, String> testData = new HashMap<>();
testData.put("username", "qa_user");
testData.put("role", "agent");

String username = testData.get("username");
System.out.println(username); // qa_user
```

13. What is TestNG? Why use it and advantages

TestNG is a Java testing framework commonly used with Selenium. It provides annotations, assertions, test grouping, prioritisation, data providers, dependencies, suite XML, listeners, and configurable parallel execution. It organises and runs tests; Selenium performs browser automation. Benefits include reusable setup/teardown, data-driven tests, flexible suite selection, and integration with Maven/CI/reporting.

```
public class LoginTest {
    @BeforeMethod
    public void setUp() {
        // Create driver and prepare test state.
    }

    @Test
    public void validLogin() {
        // Perform actions and assert result.
    }

    @AfterMethod
    public void tearDown() {
        // driver.quit();
    }
}
```

14. Have you tested parallel execution?

Answer truthfully. If you have used it, explain how your suite was configured, what ran in parallel, how browser instances and test data were isolated, and how you verified that results were reliable. Parallel execution can reduce runtime but introduces race conditions if tests share a driver, users, files, or mutable state. A common design uses one WebDriver per test thread (for example ThreadLocal in a framework), unique test data, and thread-safe reporting. If you have not used it hands-on, say you understand the approach but have not configured it in production.

```
// Illustrative TestNG suite XML:
// <suite name="UI suite" parallel="tests" thread-count="3">
//   <test name="Chrome tests"> ... </test>
//   <test name="Firefox tests"> ... </test>
// </suite>
// Each parallel test must have an isolated WebDriver/session and data.
```

15. What is Maven? Purpose of pom.xml

Maven is a build and dependency-management tool. It downloads dependencies, compiles code, runs tests, packages artifacts, and supports repeatable lifecycle phases. `pom.xml` (Project Object Model) declares project coordinates, dependencies, plugins, Java/compiler settings, profiles, and build configuration. In a Selenium project it can manage Selenium, TestNG, Apache POI, reporting libraries, and test plugins with explicit versions.

```
<!-- Illustrative pom.xml dependencies -->
<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.x.x</version>
  </dependency>
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.x.x</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<!-- Run tests with: mvn test -->
```

16. Have you used Jenkins? Why use it?

Jenkins is an automation server used to run build and test jobs automatically. A UI automation pipeline can check out code from Git, resolve Maven dependencies, compile, run TestNG/Cucumber tests, archive reports/screenshots, and notify the team. It gives repeatable execution and quicker feedback; it does not guarantee test quality, so stable tests and good failure triage are still necessary. Describe only the parts you have personally used.

17. Have you configured Jenkins? What are the steps?

A typical setup is:
1. Install/obtain Jenkins on an approved server or use the team's managed instance.
2. Configure Java and install required Jenkins plugins (Git and Pipeline; Maven integration if used).
3. Configure credentials securely for source control and any required environments.
4. Create a Pipeline job or Multibranch Pipeline and connect the repository/branch.
5. Define a Jenkinsfile or build steps to check out code, run Maven, and publish results.
6. Configure agents/labels, environment variables, parameters, and browser/headless requirements.
7. Run a test build; review console output and fix environment/dependency issues.
8. Publish TestNG/JUnit results and archive HTML reports/screenshots; configure notifications.
9. Add triggers such as webhook/SCM polling or a schedule as agreed by the team.
10. Secure credentials, control access, and maintain the pipeline.

Sample Jenkinsfile below is illustrative; report plugin and browser setup may need project-specific configuration.

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps { checkout scm }
        }
        stage('Run UI Tests') {
            steps { sh 'mvn clean test' }
        }
    }
    post {
        always {
            junit '**/target/surefire-reports/*.xml'
            archiveArtifacts artifacts: 'test-output/**', allowEmptyArchive: true
        }
    }
}
```

18. Two Submit buttons: click the second one

First locate all matching buttons and check the count. If both buttons are genuinely the same type and the second in DOM order is the required one, use `.get(1)` or `.nth(1)` conceptually (Java Selenium uses `get(1)`). Indexing is zero-based: first = 0, second = 1. If possible, scope the locator to the bottom section/container because that is more robust than relying on DOM order. Ensure the target is visible and enabled before clicking.

```
List<WebElement> submitButtons =
    driver.findElements(By.xpath("//button[normalize-space()='Submit']"));

if (submitButtons.size() >= 2) {
    WebElement secondButton = submitButtons.get(1); // zero-based index
    new WebDriverWait(driver, Duration.ofSeconds(10))
        .until(ExpectedConditions.elementToBeClickable(secondButton));
    secondButton.click();
} else {
    throw new NoSuchElementException(
        "Expected at least two Submit buttons, found: " + submitButtons.size());
}

// More robust when the bottom form has a stable container:
// driver.findElement(By.id("bottomForm"))
//       .findElement(By.xpath(".//button[normalize-space()='Submit']")).click();
```

Final preparation tips

- For project questions, prepare a 90-second overview and one real end-to-end scenario.
- For code questions, explain the locator/condition, why the approach is robust, and how you handle failure.
- Be precise about what you have personally configured versus what you understand conceptually.
- Prefer explicit waits and stable locators over Thread.sleep and fragile absolute XPath.
- For Apache POI, TestNG, Maven, and Jenkins, practise explaining one concrete use from your own project.