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

33 Interview Questions • Detailed Explanations • Practical Examples

Examples are for interview preparation. Adapt framework, deployment, and production-support answers to your real experience.

1. 1. Explain your automation framework

Describe the architecture, why it was chosen, and the flow from test to report. A common Java Selenium hybrid framework combines Page Object Model, data-driven testing, reusable utilities, TestNG and/or Cucumber, configuration, and reporting. Flow: runner/test → step definition or test method → page object → Selenium WebDriver → browser → assertions → listener/report. Maven manages dependencies and build lifecycle; Git versions code; Jenkins can run the suite in CI. Mention only tools and responsibilities you genuinely use.

```
TestNG/Cucumber runner
        ↓
Test class / feature file
        ↓
Step definitions
        ↓
Page Objects (locators + page actions)
        ↓
Utilities (waits, config, data, screenshots)
        ↓
Selenium WebDriver → Browser
        ↓
Assertions → Listener/Report → CI result
```

2. 2. How to do grouping in TestNG? Explain with code

Use TestNG groups to label tests, then include or exclude groups in testng.xml or Maven configuration. Groups are useful for smoke, sanity, regression, API, or browser-specific suites. A test can belong to more than one group. Keep smoke tests fast and focused.

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

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

@Test(groups = {"smoke", "regression"})
public void searchTest() { }

// testng.xml
<suite name="QA Suite">
  <test name="Smoke Tests">
    <groups>
      <run><include name="smoke"/></run>
    </groups>
    <classes>
      <class name="tests.LoginTest"/>
      <class name="tests.SearchTest"/>
```

```
    </classes>
  </test>
</suite>
```

3. 3. How to handle different pop-ups?

First identify the popup type; each type needs a different strategy:
• JavaScript alert/confirm/prompt: switch to `Alert`, then getText/accept/dismiss/sendKeys.
• New browser window/tab: use `getWindowHandles()` and switch to the required handle.
• HTML modal: locate it in the DOM and use ordinary WebElement methods.
• File chooser: for a standard file input, use `sendKeys(absolutePath)` on `input[type=file]`.
• Browser permission/native OS dialog: use supported browser options or approved test setup; OS-level automation may be brittle.
Always wait for the popup condition rather than using a fixed sleep.

```
// JavaScript alert
Alert alert = new WebDriverWait(driver, Duration.ofSeconds(5))
    .until(ExpectedConditions.alertIsPresent());
System.out.println(alert.getText());
alert.accept();

// New window
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);
        break;
    }
}

// HTML modal example
driver.findElement(By.cssSelector(".modal button.confirm")).click();

// File upload example
driver.findElement(By.cssSelector("input[type='file']"))
      .sendKeys(Path.of("sample.pdf").toAbsolutePath().toString());
```

4. 4. String vs StringBuffer

`String` is immutable: once created, its character sequence cannot be changed; operations return a new String. `StringBuffer` is mutable and synchronizes its methods, making individual operations thread-safe but often slower. Use String for fixed text and StringBuilder for most single-threaded string construction; use StringBuffer when synchronized mutable text is specifically needed.

```
String s = "QA";
s.concat(" Automation");
System.out.println(s); // QA: original is unchanged

StringBuffer buffer = new StringBuffer("QA");
buffer.append(" Automation");
System.out.println(buffer); // QA Automation
```

5. 5. Abstract class vs interface

An abstract class can have instance fields, constructors, abstract methods, and concrete methods. A class can extend only one class. An interface defines a contract; a class can implement multiple interfaces. Modern Java interfaces can contain abstract, default, static, and private methods, but no constructors or ordinary per-instance fields. Use an abstract class for shared state/base behaviour; use an interface for a capability or contract across possibly unrelated classes.

```
interface Searchable {
    void search(String term);
}

abstract class BasePage {
    protected WebDriver driver;
    BasePage(WebDriver driver) { this.driver = driver; }
    abstract boolean isLoaded();
    void open(String url) { driver.get(url); }
}

class SearchPage extends BasePage implements Searchable {
    SearchPage(WebDriver driver) { super(driver); }
    public void search(String term) { /* locate field and search */ }
    boolean isLoaded() { return true; }
}
```

6. 6. final vs finally vs finalize

`final` is a keyword: a final variable cannot be reassigned, a final method cannot be overridden, and a final class cannot be subclassed. `finally` is a block associated with try/catch, usually used for cleanup and normally runs whether or not an exception occurs. It may not run if the JVM terminates abruptly (for example `System.exit`, a crash, or process kill). `finalize()` was a deprecated Object method associated with garbage collection; it is unreliable and deprecated for removal. Prefer try-with-resources or explicit cleanup.

7. 7. Normal class vs final class

A normal non-final class can be extended unless another restriction applies. A final class cannot be subclassed. Mark a class final when inheritance would violate its design or invariants; do not use final merely for performance assumptions.

```
class Parent { }
final class ImmutableToken { }
// class Child extends ImmutableToken { } // compile-time error
```

8. 8. Handle frames without attributes

If a frame has no id/name/src or other useful attribute, locate it by a stable relationship or by its position among frames, then switch to the WebDriver frame context. Prefer a stable parent relationship or a unique frame index only when the page structure is reliable. If it is an iframe, inspect the DOM and count frames. Once inside, switch back with `defaultContent()`.

```
List<WebElement> frames = driver.findElements(By.tagName("iframe"));
System.out.println("Frame count: " + frames.size());

// If the frame order is stable and the intended frame is the second:
driver.switchTo().frame(1); // zero-based index
driver.findElement(By.id("insideFrame")).click();
driver.switchTo().defaultContent();
```

9. 9. Smoke vs sanity testing

Smoke testing is a broad, shallow check that the build's critical functions work and the build is testable. Sanity testing is a narrower, focused check around a particular change or fix. Example: smoke checks login, navigation, and a key transaction; sanity checks the specific fee calculation that was just changed. Terminology can vary by team, so describe your team's definition.

10. 10. QA process you follow

A practical QA process is: requirement/acceptance-criteria review → risk and test planning → test design and data preparation → environment/build validation → execution → defect logging/triage → fix retest and regression → status reporting and sign-off/closure. In Agile, this repeats each sprint. Include traceability, entry/exit criteria, automation suitability, test evidence, and risk communication.

11. 11. Adapter design pattern in Java

The Adapter pattern lets incompatible interfaces work together by wrapping one class with another interface expected by the client. Example: a framework expects a `PaymentProcessor`, but a third-party SDK exposes a different method. An adapter implements the expected interface and delegates to the SDK. In test frameworks, adapters can also translate a legacy API into a common interface. Do not confuse the Adapter design pattern with TestNG's `IAnnotationTransformer` or listener interfaces.

```
interface PaymentProcessor {
    void pay(double amount);
}
class LegacyGateway {
    void makePayment(double value) {
        System.out.println("Paid: " + value);
    }
}
class GatewayAdapter implements PaymentProcessor {
    private final LegacyGateway gateway = new LegacyGateway();
    public void pay(double amount) {
        gateway.makePayment(amount);
    }
}
class Demo {
    public static void main(String[] args) {
        PaymentProcessor processor = new GatewayAdapter();
        processor.pay(250.0);
    }
}
```

12. 12. Where have you used polymorphism?

A strong Selenium example is `WebDriver driver = new ChromeDriver();` or `WebDriver driver = new FirefoxDriver();`: the same interface reference can point to different driver implementations. Another example is method overriding in a base page or test utility. Explain how it reduces coupling and enables browser selection by configuration, if that is how your framework works.

```
WebDriver driver;
if (browser.equalsIgnoreCase("chrome")) {
    driver = new ChromeDriver();
} else {
    driver = new FirefoxDriver();
}
driver.get("https://example.com");
```

13. 13. Find memory leaks in an application

Symptoms include steadily increasing heap use, frequent garbage collection, slow response, or OutOfMemoryError. Reproduce under a controlled workload, capture heap dumps, inspect retained objects and dominator trees, and compare snapshots over time. Tools include Java Flight Recorder (JFR), Java Mission Control, VisualVM, and Eclipse MAT. Look for unbounded caches, listeners not removed, static collections retaining objects, unclosed resources, and thread-local leaks. A high heap alone does not prove a leak; objects must remain reachable unnecessarily.

14. 14. Use of multi-catch

Multi-catch lets one catch block handle multiple unrelated exception types with the same response. The exception alternatives cannot be related by subclassing, and the caught parameter is effectively final. Use it only when the handling logic is genuinely the same; otherwise separate catches improve clarity.

```
try {
    readTestData();
    connectToService();
} catch (IOException | SQLException e) {
    logger.error("Test setup failed", e);
    throw new RuntimeException(e);
}
```

15. 15. When to use Comparator

Use `Comparator` when you need a custom ordering, multiple sort strategies, or cannot modify the class being sorted. `Comparable` defines a type's natural order inside the class; `Comparator` defines an external order. For example, sort employees by salary, then by name.

```
employees.sort(
    Comparator.comparing(Employee::getSalary)
              .thenComparing(Employee::getName)
);
```

16. 16. Pass additional data between multiple web pages

For a web application, data can be passed through URL query parameters/path variables, form submissions, server-side session, cookies, or client-side storage such as sessionStorage/localStorage. Choose based on sensitivity, lifetime, and architecture. Do not put passwords, tokens, or confidential data in URLs. In Selenium tests, verify the mechanism the application actually uses; don't invent application behaviour.

17. 17. Contract between equals() and hashCode()

If two objects are equal according to `equals()`, they must return the same `hashCode()`. Unequal objects may share a hash code. `equals()` should be reflexive, symmetric, transitive, consistent, and false for null. If you override `equals()`, override `hashCode()` consistently; otherwise HashMap/HashSet behaviour can be incorrect.

```
final class UserKey {
    private final String id;
    UserKey(String id) { this.id = id; }

    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof UserKey)) return false;
        UserKey other = (UserKey) o;
        return Objects.equals(id, other.id);
    }
    @Override public int hashCode() {
        return Objects.hash(id);
    }
}
```

18. 18. Improve application performance

Start with a measurable baseline and define the target: response time, throughput, CPU, memory, database latency, or error rate. Use profiling/APM, logs, metrics, and load tests to locate the bottleneck before changing code. Common areas include inefficient SQL/indexes, N+1 queries, excessive network calls, large payloads, unnecessary synchronous work, unbounded caches, and thread contention. Optimise the measured bottleneck and compare results under the same workload. Avoid claiming that caching or adding threads always improves performance.

19. 19. Use of functional interface

A functional interface has exactly one abstract method and can be implemented with a lambda or method reference. It supports concise behaviour passing. Examples in Java include `Runnable`, `Predicate<T>`, `Function<T,R>`, and `Comparator<T>`. Default/static methods do not count as abstract methods.

```
Predicate<String> isBlank = s -> s == null || s.trim().isEmpty();
System.out.println(isBlank.test("")); // true

Runnable task = () -> System.out.println("Running test task");
task.run();
```

20. 20. SQL query to find duplicate values in a column

Use GROUP BY and HAVING COUNT(*) > 1. Use the email and users columns or tables defined by your database. If you need duplicate rows, join the grouped result back to the source table.

```
SELECT email, COUNT(*) AS duplicate_count
FROM users
GROUP BY email
HAVING COUNT(*) > 1;
```

21. 21. Grant permissions on a table

Syntax depends on the database. Common SQL syntax is `GRANT SELECT, INSERT ON table_name TO user_name;`; some databases require schema-qualified names or quote rules. Grant only the permissions needed. Oracle/PostgreSQL/MySQL differ in role and privilege details.

```
GRANT SELECT, INSERT ON schema_name.table_name TO app_user;
-- To grant read-only access:
GRANT SELECT ON schema_name.table_name TO reporting_user;
```

22. 22. Primary key vs unique key

A primary key uniquely identifies each row and cannot be NULL; a table has one primary-key constraint, though it can contain multiple columns. A table can have multiple UNIQUE constraints. NULL handling in UNIQUE constraints differs by database, so check the database's rules. Both typically create or use a unique index depending on the database.

23. 23. Immutable class without declaring the class final

A class need not be declared `final` to be effectively immutable, but it must prevent mutation and unsafe subclassing. A common approach is a private constructor plus a factory that only returns the class's own instances, or a sealed hierarchy where appropriate. For a simple interview example, use private final fields, no setters, defensive copies of mutable inputs/outputs, and make the constructor private with a static factory; because the constructor is private, ordinary external subclassing is blocked.

```
public class ImmutablePerson {
    private final String name;
    private final List<String> tags;

    private ImmutablePerson(String name, List<String> tags) {
        this.name = name;
        this.tags = List.copyOf(tags); // defensive immutable copy
    }
    public static ImmutablePerson of(String name, List<String> tags) {
        return new ImmutablePerson(name, tags);
    }
    public String getName() { return name; }
    public List<String> getTags() { return tags; }
}
```

24. 24. Program demonstrating OOP concepts

This example demonstrates encapsulation (private fields), abstraction (abstract class), inheritance (`Dog` extends `Animal`), and polymorphism (an `Animal` reference points to a `Dog`).

```
abstract class Animal {
    private final String name;             // encapsulation
    Animal(String name) { this.name = name; }
    public String getName() { return name; }
    abstract void sound();                 // abstraction
}
class Dog extends Animal {                 // inheritance
    Dog(String name) { super(name); }
    @Override void sound() { System.out.println("Woof"); }
}
public class OOPDemo {
    public static void main(String[] args) {
        Animal animal = new Dog("Max");     // polymorphism
        System.out.println(animal.getName());
        animal.sound();
    }
}
```

25. 25. Store a custom object in TreeSet or TreeMap

TreeSet elements and TreeMap keys must be orderable, either through `Comparable` natural ordering or a supplied `Comparator`. Without either, inserting a non-comparable custom object generally causes `ClassCastException`. The comparison must be consistent enough for the intended ordering; if compare returns zero for distinct objects, a TreeSet treats them as the same element and a TreeMap treats the key as equivalent.

```
Set<Employee> employees = new TreeSet<>(
    Comparator.comparing(Employee::getName)
);
// Employee must provide getName(); a comparator defines ordering.
```

26. 26. Why String is immutable but StringBuffer is mutable

String immutability supports safe sharing, string-pool reuse, stable hash codes for map keys, and security/reliability. StringBuffer is designed for a mutable sequence of characters and synchronises operations. If every string operation mutated the original, sharing a string between components could unexpectedly change data. Use StringBuilder for most local string construction.

27. 27. Pull mechanism vs push mechanism

In pull, the consumer requests data when it needs it (for example, polling an API or reading a queue). In push, the producer sends updates to consumers as events occur (for example, webhook callbacks or server-sent events). Pull is simple and consumer-controlled but can add polling delay/load; push can be timely but needs connection/retry/back-pressure design.

28. 28. Synchronous vs asynchronous communication

Synchronous communication waits for the response before the caller continues, such as a blocking HTTP request. Asynchronous communication lets the caller continue and handles the result later through a callback, future, event, or queue. Asynchronous does not necessarily mean parallel or faster; it changes how work is coordinated and how errors/timeouts are handled.

29. 29. Absolute path vs relative path (XPath)

Absolute XPath starts from the document root, such as `/html/body/div[1]/input`, and is often brittle when the page structure changes. Relative XPath starts from a matching node using `//`, for example `//input[@id='username']`, and is usually easier to maintain. Use stable attributes and relationships; relative XPath is not automatically stable if its attributes are unstable.

```
// Relative XPath
//input[@id='username']
 //button[normalize-space()='Submit']

/* Absolute XPath example (usually brittle) */
/html/body/div[1]/form/input[1]
```

30. 30. Resolve Git conflicts

Fetch/pull the latest changes, check out the intended branch, merge or rebase according to team policy, identify conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`), edit the file to the correct final version, run tests, stage the resolved files, and commit/continue the merge or rebase. Communicate with the other author when the intended behaviour is unclear. Never blindly accept “ours” or “theirs” without reviewing the code.

```
git status
git fetch origin
git switch feature/my-change
git merge origin/main
# Edit conflicted files and remove conflict markers
git add path/to/resolved-file
git commit
git push
```

31. 31. Build a WAR file and deploy it

For a Maven web application, set `<packaging>war</packaging>` in `pom.xml`, compile/test/package with `mvn clean package`, and obtain the WAR from `target/`. Deployment may involve an approved application-server pipeline (Tomcat, WebLogic, etc.), container image, or platform-specific deployment tooling. Confirm configuration, secrets, database migrations, health checks, and environment-specific properties. Use a controlled pipeline rather than manually copying artifacts where possible.

```
<!-- pom.xml -->
<packaging>war</packaging>

<!-- Build -->
mvn clean test package
<!-- Typical output: target/application-name.war -->
```

32. 32. Rollback policy if deployment fails

Use a documented release strategy: preserve the last known-good artifact/configuration, define health checks and rollback triggers, monitor logs/metrics, and have an authorised rollback owner. Depending on the platform, rollback may mean redeploying the previous immutable artifact, switching traffic to the previous version, or rolling back a release/feature flag. Database changes need backward-compatible expand/contract migrations; blindly rolling back a destructive schema change can cause data loss. Validate service health after rollback and record the incident.

33. 33. Steps to resolve a production issue

1) Acknowledge and assess severity/customer impact. 2) Follow incident process and assign an incident lead. 3) Gather timestamps, correlation IDs, logs, metrics, traces, recent releases, and reproduction steps. 4) Stabilise service safely—rollback, disable a feature flag, or apply an approved mitigation. 5) Verify recovery with health checks and business transactions. 6) Communicate status and impact to stakeholders. 7) Preserve evidence and perform root-cause analysis. 8) Create corrective/preventive actions, tests, monitoring, and a blameless post-incident review. Do not make unreviewed production changes or expose customer data in logs.

Useful imports for code samples

```
import java.io.*;
import java.nio.file.*;
import java.time.Duration;
import java.util.*;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.*;
import org.testng.annotations.Test;
import org.testng.Assert;
import org.apache.poi.ss.usermodel.*;
```

Final interview preparation tips

- Explain the concept first, then give a project example and one limitation or best practice.
- For Selenium scenarios, describe the locator, wait condition, action, and assertion.
- For production/deployment questions, emphasise approved procedures, monitoring, rollback safety, and communication.
- Adapt examples to the actual framework, database, CI tool, and responsibilities you have used.