Selenium Interview Questions and Answers (2026) | Saravana Tech

114 Questions with Answers

Selenium · Java · TestNG · Cucumber · Maven · Git · Jenkins

Saravana Tech

Contents

1. How to get the count of all the links on a web page? . . . 5
2. How to read data from Notepad? . . . 5
3. How to read a file which is present in Network, like SharePoint? . . . 6
4. What are Maven, Git and Jenkins? . . . 7
5. Limitations of Selenium . . . 8
6. Write code for taking a screenshot . . . 10
7. How to handle Alert? . . . 10
8. How to handle multiple windows? . . . 10
9. Annotations in TestNG . . . 11
10. You have 100 scenarios containing Smoke and Sanity. You want to run Smoke alone. How? . . . 12
11. What reports are you maintaining? Extent Report / Allure Report . . . 14
12. How to read data from Excel and write data to Excel? . . . 14
13. What challenges have you faced in Selenium? . . . 15
14. Explain BDD, TDD and DDD . . . 16
15. Explain Feature File . . . 17
16. What are the keywords in a Feature File? . . . 18
17. How would you pass parameters from outside a class? . . . 18
18. Difference between findElement() and findElements() . . . 19
19. What is Relative XPath and Absolute XPath? . . . 19
20. Explain XPath Axes which you have used in your project . . . 19
21. How many locators are present in Selenium? Explain each with example. . . . 21
22. How would you do Drag and Drop? . . . 22
23. How will you find the total number of URLs in a page? . . . 23
24. How would you create a Jenkins build? . . . 23
25. Explain the version controlling software you use in your projects. . . . 25
26. How to open a browser with customized settings? . . . 26
27. How do you go to the homepage directly if you logged in during the previous session? . . . 27
28. Maven build JAR file to push to Git — how do you secure it from viruses or hackers? . . . 27
29. Explain Access Modifiers in Java . . . 29
30. Explain Set and List in Java . . . 30
31. StringBuffer and StringBuilder . . . 31
32. Interface and Abstract Class . . . 32
33. Sort only the first 4 elements of int[] a = {1,4,8,3,7,6,2} . . . 33
34. Java program to find whether a given number is prime or not . . . 33
35. Java program to change lowercase letters to uppercase and uppercase to lowercase . . . 34
36. Java program to print a pyramid . . . 34
37. Java program to reverse a number . . . 35
38. Differences between HashSet, HashMap and Hashtable . . . 35
39. Mouse hover, open a browser, go to File and save it . . . 36

40. How to handle a tooltip in Selenium? . . . 37
41. Write a program to read from PDF . . . 38
42. Write a sorting program for String without using String inbuilt sorting methods . . . 39
43. Write a Singleton program in Java with Double-Check Locking . . . 39
44. What is Garbage Collector in Java? . . . 40
45. How do you achieve multiple inheritance in Java? . . . 41
46. Write a Binary Search program in Java using recursion . . . 42
47. Implement a singly linked list, add a new node, and print the list in reverse . . . 42
48. Write a Java program for Serialization and Deserialization . . . 44
49. Multithreading program to print even and odd numbers using wait() and notify() . . . 45
50. Write an email validation program in Java . . . 46
51. Write a Java program to find the maximum number in an array . . . 46
52. What is a user-defined exception? Create your own exception . . . 47
53. Find how many times values are duplicated using HashMap . . . 47
54. TestNG XML structure with parallel="tests" and parallel="classes" . . . 48
55. How do you handle dependent test cases in TestNG? . . . 49
56. Programs: String palindrome, integer palindrome, HashMap/Hashtable, threads, duplicates/non-duplicates . . . 50
57. What framework are you currently using? Cucumber, POM Hybrid . . . 51
58. How do you handle wait for a particular element on a webpage? . . . 52
59. How do you fetch values from a web table? . . . 52
60. Difference between JVM and JRE . . . 53
61. Difference between ArrayList and Array . . . 53
62. Use of List, Set and Map in Collections . . . 54
63. OOP concepts with example and constructor . . . 54
64. Method overloading and method overriding . . . 55
65. Assert, Verify and waitFor . . . 56
66. super and final keyword . . . 56
67. Is String immutable? . . . 57
68. Difference between == and .equals() . . . 57
69. What is 'this' in Java? . . . 58
70. XPath: preceding, following-sibling and axes – when do we use them? . . . 58
71. PageFactory: use, where to use it, and advantages . . . 59
72. What is TestNG? Why do you use it? Advantages . . . 60
73. What is Maven? What is the purpose of pom.xml? . . . 60
74. Have you used Jenkins? Why do you use it? . . . 61
75. Two Submit buttons: click the second one when it is not visible and scroll down . . . 61
76. When should you use Robot, Actions and JavascriptExecutor? . . . 62
77. What is a Marker Interface? . . . 62
78. How do you find broken links? . . . 63
79. Listeners concept in TestNG . . . 63
80. Exception concept in Java . . . 64
81. How to handle a secured HTTPS site – Firefox profile . . . 65

82. Write code for connecting to a database . . . 66
83. What challenges did you face while creating the automation framework? . . . 66
84. Separate alphabets, numbers and special characters from a string . . . 67
85. Copy all contents from one Excel sheet to another sheet . . . 67
86. Frames in Selenium – click an element in a nested frame . . . 69
87. Print numbers in reverse order and omit odd numbers . . . 69
88. How do you know whether an element is visible on the page? . . . 70
89. How will you run failed cases in TestNG? . . . 70
90. TestNG priority and dependency order – Priority 1, dependsOn A priority 2, and C priority -34 . . . 71
91. Run entire package and exclude only two methods in a class without changing code . . . 71
92. Static keyword in Java . . . 72
93. Run some testcases in parallel and some sequentially . . . 72
94. Verify checkbox is selected; if not, select it . . . 73
95. CSV and XML file format reader . . . 73
96. Handle dropdown in Selenium . . . 74
97. What is StaleElementReferenceException? . . . 75
98. Code to extract HTML source . . . 75
99. quit and close – parameters and return value . . . 75
100. Difference between implicit wait, WebDriverWait and FluentWait . . . 76
101. Actions class import – Selenium or Java? . . . 76
102. Access modifiers and their use . . . 76
103. How to check whether a radio button is clicked/selected . . . 77
104. Find the nearest element to a given element using XPath . . . 77
105. How will you trigger an event in Selenium? . . . 77
106. Count vowels, digits, consonants and spaces in a string . . . 78
107. Check whether a number is an Armstrong number . . . 78
108. Java code to scroll a web page . . . 79
109. Check whether web pages are loaded or not . . . 79
110. Create a method to get specified tag elements and store them in a List . . . 80
111. Upload an image/file when the upload button uses a file input . . . 80
112. Find the Nth largest number in an array . . . 81
113. Provide a unique email every time to a textbox . . . 81
114. Types of reports – Extent Report and Allure Report . . . 82

Q1 How to get the count of all the links on a web page?

In Selenium, links are generally represented by the <a> HTML tag.

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

Complete example

```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class LinkCount {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.google.com");
        List<WebElement> links =
                driver.findElements(By.tagName("a"));
        System.out.println("Total links: " + links.size());
        driver.quit();
    }
}
```

If interviewer asks: "How will you print all link names?"

```
List<WebElement> links = driver.findElements(By.tagName("a"));
for (WebElement link : links) {
    System.out.println(link.getText());
}
```

If they ask for URL also:

```
for (WebElement link : links) {
    System.out.println(
            "Text: " + link.getText() +
            " | URL: " + link.getAttribute("href")
    );
}
```

Important interview point

findElement() returns **one WebElement**.

findElements() returns a **List<WebElement>**.

Q2 How to read data from Notepad?

A Notepad file is normally a .txt file.

In Java, you can use BufferedReader, FileReader, or modern Java Files.

Simple approach using BufferedReader

Suppose data.txt contains:

```
Selenium
Java
TestNG
Maven
Jenkins
```

Code:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadNotepad {
    public static void main(String[] args) throws IOException {
        BufferedReader br =
                new BufferedReader(new FileReader("C:\\TestData\\data.txt"));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        br.close();
    }
}
```

Better approach with try-with-resources

```
try (BufferedReader br =
        new BufferedReader(
        new FileReader("C:\\TestData\\data.txt"))) {
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
}
```

Q3 How to read a file which is present in Network, like SharePoint?

This question has an important distinction.

If the file is on **SharePoint/network storage**, Selenium is **not normally used to read the file directly**.

Selenium is primarily for browser automation.

You have several approaches depending on how the file is exposed.

Scenario 1: Network shared folder

For example:

\\server\sharedfolder\TestData.xlsx

```
File file = new File("\\\\server\\sharedfolder\\TestData.xlsx");
System.out.println(file.exists());
```

Scenario 2: SharePoint URL

For example:

https://company.sharepoint.com/sites/project/Shared Documents/TestData.xlsx

If authentication is required, you generally shouldn't simply treat it as a local file.

Possible approaches are:

Approach A — Sync SharePoint folder locally

If SharePoint/OneDrive synchronizes the folder to the machine:

```
C:\Users\User\Company\Project\TestData.xlsx
```

Then Java can read it like a normal file.

Q4 What are Maven, Git and Jenkins?

These three are often asked together because they are commonly used in automation frameworks and CI/CD.

Maven

Maven is a build and dependency management tool for Java projects.

It helps us:

- Manage dependencies
- Compile code
- Run tests
- Create reports
- Package applications
- Manage plugins

The main configuration file is: pom.xml

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>...</version>
</dependency>
```

Common Maven command

```
mvn clean test
```

Meaning:

clean → removes previous build test → runs tests

Git

Git is a distributed version control system.

It is used to:

- Track code changes
- Maintain different versions
- Create branches
- Merge code
- Collaborate with team members
- Revert changes

Common commands:

```
git clone
git status
git add
git commit
git push
git pull
git checkout
git branch
git merge
```

Example workflow:

```
     Developer
         ↓
     git clone
         ↓
  Create/change code
         ↓
      git add
         ↓
     git commit
         ↓
      git push
         ↓
  Remote repository
```

Jenkins

Jenkins is a CI/CD automation server.

It can automatically execute your automation tests.

For example:

```
  Developer pushes code
            ↓
           Git
            ↓
         Jenkins
            ↓
          Maven
            ↓
       TestNG Tests
            ↓
       Test Results
            ↓
          Report
```

Suppose your framework has:

```
  mvn clean test
```

Jenkins can execute this command automatically.

Interview Answer :

"Maven manages the Java project and dependencies, Git manages source code and version control, and Jenkins is used for CI/CD to automatically build the project and execute the test suite."

Q5 Limitations of Selenium

This is a **very important interview question**.

Selenium is powerful, but it has limitations.

1. Cannot automate CAPTCHA reliably

For example:

I'm not a robot

CAPTCHA is specifically designed to prevent automation.

You should not try to automate CAPTCHA in a real test.

Usually:

CAPTCHA → bypass/mock/disable in test environment

2. Cannot directly automate desktop applications

Selenium is designed for web browsers.

For example:

Chrome Edge Firefox

But not:

- Windows Calculator
- Notepad
- Desktop application

3. Limited handling of OTP

Selenium cannot magically read an OTP sent to:

Mobile

Email

You need another mechanism/API/test hook to retrieve it.

4. Browser dependency

Selenium interacts with browsers, so browser versions, drivers, and compatibility can sometimes become issues.

Modern Selenium has Selenium Manager to simplify driver management, but browser/environment compatibility can still matter.

5. Cannot directly test API endpoints

Selenium is not an API testing tool.

For API automation, we can use:

- Rest Assured
- Postman
- REST clients

6. File handling is limited

Selenium can interact with browser file upload/download flows, but Java libraries are generally better for actually reading/manipulating files.

For example:

Excel → Apache POI

PDF → PDF libraries

JSON → Jackson/Gson

CSV → CSV libraries

7. Cannot directly automate database operations

For database testing, we generally use:

- JDBC
- SQL
- Database client/tools

Q6 Write code for taking a screenshot

Selenium provides the TakesScreenshot interface.

```
import java.io.File;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;

TakesScreenshot ts = (TakesScreenshot) driver;
File source = ts.getScreenshotAs(OutputType.FILE);
File destination =
        new File("C:\\Screenshots\\test.png");
FileUtils.copyFile(source, destination);
```

Q7 How to handle Alert?

Selenium provides:

```
driver.switchTo().alert();
```

Suppose the browser shows:

Are you sure you want to delete? OK Cancel

Accept alert

```
Alert alert = driver.switchTo().alert();
alert.accept();
```

Dismiss alert

```
Alert alert = driver.switchTo().alert();
alert.dismiss();
```

Get alert text

```
String message =  driver.switchTo().alert().getText();
System.out.println(message);
```

Enter text into prompt

If it's a JavaScript prompt:

```
Alert alert = driver.switchTo().alert();
alert.sendKeys("Saravana Tech");
alert.accept();
```

Q8 How to handle multiple windows?

Selenium provides:

```
getWindowHandle()
getWindowHandles()
```

getWindowHandle()

Returns the current window ID.

```
String parentWindow = driver.getWindowHandle();
```

getWindowHandles()

Returns all currently opened window IDs.

```
Set<String> windows = driver.getWindowHandles();
```

Example

Suppose:

```
Parent Window
      ↓
    Click
      ↓
 Child Window
```

```
String parentWindow = driver.getWindowHandle();
driver.findElement(By.id("openWindow")).click();
Set<String> windows = driver.getWindowHandles();
for (String window : windows) {
    if (!window.equals(parentWindow)) {
        driver.switchTo().window(window);
        System.out.println(driver.getTitle());
        driver.close();
    }
}

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

Q9 Annotations in TestNG

TestNG annotations tell TestNG **when a method should execute**.

The most commonly used annotations are:

Execution order

A simplified execution order is:

```
@BeforeSuite
@BeforeTest
@BeforeClass
@BeforeMethod
@Test
@AfterMethod
@BeforeMethod
@Test
@AfterMethod
@AfterClass
@AfterTest
@AfterSuite
@BeforeMethod
public void setUp() {
    System.out.println("Before Test");
}

@Test
public void loginTest() {
    System.out.println("Login Test");
}

@AfterMethod
public void tearDown() {
    System.out.println("After Test");
}
```

If you have three @Test methods:

```
@BeforeMethod
```

Test 1

```
@AfterMethod
@BeforeMethod
```

Test 2

```
@AfterMethod
@BeforeMethod
```

Test 3

```
@AfterMethod
```

Important TestNG annotations

@Test

Marks a method as a test.

```
@Test
public void loginTest() {
    System.out.println("Login");
}
```

@BeforeMethod

Runs before every test method.

@AfterMethod

Runs after every test method.

@BeforeClass

Runs before the first test method in the class.

@AfterClass

Runs after all test methods in the class.

@BeforeSuite

Runs once before the entire TestNG suite.

@AfterSuite

Runs once after the entire TestNG suite.

@BeforeTest

Runs before the <test> section in testng.xml.

@AfterTest

Runs after the <test> section.

Q10 You have 100 scenarios containing Smoke and Sanity. You want to run Smoke alone. How?

This is probably the **most important TestNG question in your list**.

The standard solution is **TestNG Groups**.

Suppose you have
TC001 → Smoke
TC002 → Sanity
TC003 → Smoke

TC004 → Sanity
TC005 → Smoke

We can assign groups.

```
@Test(groups = "smoke")
public void loginTest() {
    System.out.println("Login");
}

@Test(groups = "sanity")
public void paymentTest() {
    System.out.println("Payment");
}

@Test(groups = "smoke")
public void searchTest() {
    System.out.println("Search");
}
```

Example:

```
import org.testng.annotations.Test;

public class LoginTest {
    @Test(groups = "smoke")
    public void validLogin() {
        System.out.println("Valid Login");
    }
    @Test(groups = "sanity")
    public void invalidLogin() {
        System.out.println("Invalid Login");
    }
    @Test(groups = "smoke")
    public void logout() {
        System.out.println("Logout");
    }
}
```

How do you execute Smoke only?

Use testng.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM
    "https://testng.org/testng-1.0.dtd">
<suite name="Smoke Suite">
    <test name="Smoke Tests">
        <groups>
            <run>
                <include name="smoke"/>
            </run>
        </groups>
        <classes>
            <class name="tests.LoginTest"/>
        </classes>
    </test>
</suite>
```

Now TestNG will execute:

```
  Smoke
    ↓
validLogin
```

logout

and skip:

```
    Sanity
```

```
     ↓
invalidLogin
```

Q11 What reports are you maintaining? Extent Report / Allure Report

Interview answer

In my automation framework, we maintain automation execution reports to track passed, failed and skipped test cases. I have worked with reports such as **Extent Reports and Allure Reports**.

The report generally contains the test case name, execution status, steps, screenshots for failures, exception details and execution time.

For failed test cases, we capture a screenshot and attach it to the report. The reports are generated as part of the CI/CD execution through Jenkins.

Extent Report example

```
ExtentReports extent = new ExtentReports();
ExtentTest test = extent.createTest("Login Test");
test.pass("Login page opened");
test.pass("Username entered");
test.fail("Login failed");
extent.flush();
```

Screenshot attachment

```
String screenshotPath = "screenshots/login.png";
test.addScreenCaptureFromPath(screenshotPath);
```

Allure

With Allure, we can attach screenshots and other information:

```
Allure.addAttachment(
    "Screenshot",
    new ByteArrayInputStream(
        ((TakesScreenshot) driver).getScreenshotAs(OutputType.BYTES)
    )
);
```

Extent Report	Allure
HTML-based reporting	Rich interactive reporting
Easy to customize	Excellent visualization
Easy screenshot attachment	Supports attachments, steps, categories
Common with TestNG	Works with TestNG, JUnit, Cucumber
Simple setup	Very useful for CI/CD

Q12 How to read data from Excel and write data to Excel?

```
import java.io.FileInputStream;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.apache.poi.ss.usermodel.*;

public class ExcelRead {
    public static void main(String[] args) throws Exception {
        FileInputStream file =
```

```
                new FileInputStream("TestData.xlsx");
            Workbook workbook = new XSSFWorkbook(file);
            Sheet sheet = workbook.getSheet("Login");
            Row row = sheet.getRow(1);
            String username =
                    row.getCell(0).getStringCellValue();
            String password =
                    row.getCell(1).getStringCellValue();
            System.out.println(username);
            System.out.println(password);
            workbook.close();
            file.close();
        }
    }
```

Excel example

Username	Password
user1	pass123
user2	pass456

Write data to Excel

```
FileInputStream file =
    new FileInputStream("TestData.xlsx");
Workbook workbook = new XSSFWorkbook(file);
Sheet sheet = workbook.getSheet("Login")
Row row = sheet.createRow(3);
row.createCell(0).setCellValue("user3");
row.createCell(1).setCellValue("pass789");
FileOutputStream output =
    new FileOutputStream("TestData.xlsx");
workbook.write(output);
workbook.close();
file.close();
output.close();
```

Q13 What challenges have you faced in Selenium?

This is an important **real-project interview question**.

Don't simply say "I faced synchronization issues." Give multiple practical examples.

Interview answer

I have faced several challenges while working with Selenium automation.

1. Dynamic elements

Some elements have dynamic IDs or attributes.

```
<input id="user_12345">
```

The ID changes every execution.

I use stable attributes or XPath:

```
//input[contains(@id,'user_')]
```

2. Synchronization issues

Sometimes Selenium executes faster than the application.

I use explicit waits:

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(20));
wait.until(
    ExpectedConditions.elementToBeClickable(loginButton)
).click();
```

3. StaleElementReferenceException

This happens when the DOM is refreshed or the element is recreated.

Solution:

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(20));
wait.until(
    ExpectedConditions.refreshed(
        ExpectedConditions.elementToBeClickable(locator)
    )
);
```

Or locate the element again before interacting.

4. Element intercepted / overlay issues

Sometimes a popup, loader or overlay is covering the element.

I wait for the overlay to disappear:

```
wait.until(
    ExpectedConditions.invisibilityOfElementLocated(
        By.id("loader")
    )
);
```

5. Multiple windows

I use:

```
Set<String> windows = driver.getWindowHandles();
for (String window : windows) {
    driver.switchTo().window(window);
}
```

6.Dynamic tables

I use relative XPath and iterate through rows.

```
List<WebElement> rows =
    driver.findElements(By.xpath("//table//tbody/tr"));
System.out.println(rows.size());
```

Q14 Explain BDD, TDD and DDD

BDD – Behavior Driven Development

BDD focuses on **application behavior from the user's/business perspective**.

BDD commonly uses:

Given When Then

Example:

```
Feature: Login
  Scenario: Successful login
    Given user is on the login page
    When user enters valid username and password
    And clicks login
    Then user should be successfully logged in
```

TDD – Test Driven Development

TDD means writing the test **before writing the implementation code**.

The typical cycle is:

```
   Red
    ↓
  Green
    ↓
Refactor
```

Example

First write:

```
@Test
public void additionTest() {
    assertEquals(10, calculator.add(5, 5));
}
```

Initially it may fail.

Then implement the functionality.

Finally refactor the code.

Simple difference

TDD → Developer/code-focused

BDD → Business behavior-focused

DDD – Domain Driven Design

DDD stands for **Domain Driven Design**.

It focuses on designing software around the **business domain and business rules**.

For example, in a banking application:

Customer
Account
Transaction
Loan
Payment

These business concepts become part of the domain model.

TDD	BDD	DDD
Test-focused	Behavior-focused	Business-domain-focused
Usually developer-oriented	Developer + QA + Business	Architecture/design
Tests drive implementation	Scenarios describe behavior	Domain drives system design
Red → Green → Refactor	Given → When → Then	Domain model

Q15 Explain Feature File

A **feature file** is a Cucumber file that describes application behavior in a business-readable format.

The extension is:

.feature

Example:

```
Feature: Login functionality
```

Background:

```
  Given user is on the login page

Scenario: Successful login
  When user enters valid username
  And user enters valid password
  And user clicks login button
  Then user should see the home page
```

Structure

```
 Feature
    ↓
Background
    ↓
 Scenario
    ↓
  Steps
```

Q16 What are the keywords in a Feature File?

The important Gherkin keywords are:

Feature
Background
Scenario
Scenario Outline
Examples
Given
When
Then
And
But

Q17 How would you pass parameters from outside a class?

There are several ways depending on the requirement.

Method 1 – Constructor

```
public class Login {
    String username;
    public Login(String username) {
        this.username = username;
    }
    public void login() {
        System.out.println(username);
    }
}
```

Calling :

```
Login obj = new Login("Vajravel");
obj.login();
```

Method 2 – Method parameter

```
public void login(String username, String password)
{
    System.out.println(username);
    System.out.println(password);
}
```

Q18 Difference between findElement() and findElements()

This is a very common Selenium question.

findElement()	findElements()
Returns a single WebElement	Returns List<WebElement>
Finds first matching element	Finds all matching elements
If not found → NoSuchElementException	If not found → empty list
Used for one element	Used for multiple elements

Q19 What is Relative XPath and Absolute XPath?

Absolute XPath

Absolute XPath starts from the root of the HTML document.

Example:

```
/html/body/div[1]/div[2]/form/input[1]
```

Disadvantage:

If one div is added or removed, the XPath can break.

Relative XPath

Relative XPath starts with: //

```
//input[@id='username']
```

Relative XPath is preferred because:

- More maintainable
- Less dependent on DOM structure
- Easier to understand
- More robust against UI changes

Q20 Explain XPath Axes which you have used in your project

XPath axes are used to navigate between elements based on their relationship in the DOM.

The important ones are:

parent
child
ancestor
descendant

following
following-sibling
preceding
preceding-sibling
self

1. Parent

Suppose:

```
<div>
    <input id="username">
</div>
```

XPath:

```
//input[@id='username']/parent::div
```

2. Child

```
//div[@class='login']/child::input
```

Finds direct child input elements.

3. Ancestor

```
//input[@id='username']/ancestor::form
```

Finds the parent/ancestor form.

4. Descendant

```
//div[@class='login']/descendant::input
```

Finds input elements anywhere below the div.

5. Following-sibling

Example:

```
<div>
    <label>Username</label>
    <input>
</div>
```

XPath:

```
//label[text()='Username']/following-sibling::input
```

This is very useful when the label and input are siblings.

6. Preceding-sibling

```
//input[@id='username']/preceding-sibling::label
```

Gets the label before the input.

7. Following

following finds elements appearing later in the DOM.

```
//label[text()='Username']/following::input
```

8. Preceding

preceding finds elements appearing before the current element.

```
//input[@id='password']/preceding::label
```

Q21 How many locators are present in Selenium? Explain each with example.

Selenium provides **8 traditional locator strategies** in Selenium WebDriver:

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

1. ID

If the element has a unique id:

```
<input id="username" type="text">
driver.findElement(By.id("username")).sendKeys("Vajravel");
```

Preferred when available because ID is generally fast and reliable.

2. Name

```
<input name="username" type="text">
driver.findElement(By.name("username"))
      .sendKeys("Vajravel");
```

3. Class Name

```
<input class="form-control" type="text">
driver.findElement(By.className("form-control"))
      .sendKeys("Vajravel");
```

⚠ If multiple classes exist:

```
class="form-control login-input"
```

Don't pass:

```
By.className("form-control login-input")
```

Use CSS instead:

```
By.cssSelector(".form-control.login-input")
```

4. Tag Name

```
<input type="text">
List<WebElement> inputs =
    driver.findElements(By.tagName("input"));
System.out.println(inputs.size());
```

Useful for finding multiple elements of the same type.

5. Link Text

For:

```
<a href="/login">Login</a>
driver.findElement(By.linkText("Login")).click();
```

Works with complete visible link text.

6. Partial Link Text

```
driver.findElement(
    By.partialLinkText("Log")
).click();
```

This can locate:

Login
Logout
Login Here

depending on matching.

7. CSS Selector

```
driver.findElement(
    By.cssSelector("#username")
).sendKeys("Vajravel");
```

Another example:

```
driver.findElement(
    By.cssSelector("input[name='username']")
);
```

8. XPath

```
driver.findElement(
    By.xpath("//input[@id='username']")
).sendKeys("Vajravel");
```

Example using text:

```
driver.findElement(
    By.xpath("//button[normalize-space()='Login']")
).click();
```

Q22 How would you do Drag and Drop?

Selenium provides the **Actions class** for mouse interactions such as drag and drop.

Method 1 – dragAndDrop()

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

Method 2 – Click, hold, move and release

```
Actions actions = new Actions(driver);
actions.clickAndHold(source)
       .moveToElement(target)
       .release()
       .perform();
```

Interview answer

I use Selenium's Actions class for drag-and-drop operations. The simplest approach is dragAndDrop(source, target).perform(). If the application doesn't support the standard drag-and-drop operation properly, I use clickAndHold, moveToElement and release.

Q23 How will you find the total number of URLs in a page?

Usually, URLs are represented by <a> tags.

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

If I specifically want the URLs:

```
for (WebElement link : links) {
    String url = link.getAttribute("href");
    System.out.println(url);
}
```

Count only non-empty URLs

```
int count = 0;
for (WebElement link : links) {
    String url = link.getAttribute("href");
    if (url != null && !url.isEmpty()) {
        count++;
    }
}

System.out.println("Total URLs: " + count);
```

Interview answer

I locate all anchor tags using findElements(By.tagName("a")). Then I use getAttribute("href") to retrieve the URL. If required, I filter out null or empty href values and count the remaining URLs.

Q24 How would you create a Jenkins build?

Interview answer

I integrate the automation project with Jenkins so that the tests can be executed automatically. Jenkins gets the latest code from Git, builds the Maven project, executes the test cases and publishes the reports.

Typical flow

```
       Git
        ↓
    Jenkins
        ↓
 Checkout code
        ↓
  Maven build
        ↓
   Run tests
        ↓
Generate report
        ↓
 Publish report
```

Step 1 – Create Jenkins job

In Jenkins:

```
           New Item
              ↓
```

```
Freestyle Project / Pipeline
```

Configure the Git repository.

For example:

```
Source Code Management
          ↓
         Git
          ↓
    Repository URL
          ↓
     Credentials
```

Step 2 – Add Maven command

For a Maven project:

```
mvn clean test
```

Or:

```
mvn clean install
```

If you have a specific suite:

```
mvn clean test -DsuiteXmlFile=testng.xml
```

Step 3 – Configure report

For example, TestNG/Extent/Allure reports can be published depending on the Jenkins plugins and project configuration.

Pipeline example

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/example/project.git'
            }
        }
        stage('Build') {
            steps {
                sh 'mvn clean test'
            }
        }
        stage('Report') {
            steps {
                echo 'Publishing test report'
            }
        }
    }
}
```

Important interview point

If using Windows Jenkins agent:

bat 'mvn clean test'

instead of:

sh 'mvn clean test'

Q25 Explain the version controlling software you use in your projects.

If you're using Git, an interview answer can be:

I use Git for version control. It allows the team to maintain source code, create branches, merge changes, review code and maintain the project history.

Common Git workflow

```
Working Directory
        ↓
     git add
        ↓
   Staging Area
        ↓
    git commit
        ↓
 Local Repository
        ↓
     git push
        ↓
Remote Repository
```

Common commands

Clone

```
git clone <repository>
```

Check status

```
git status
```

Create branch

```
git checkout -b feature/login
```

or:

```
git switch -c feature/login
```

Add changes

```
git add .
```

Commit

```
git commit -m "Added login automation"
```

Push

```
git push origin feature/login
```

Pull latest changes

```
git pull origin main
```

Merge

```
git checkout main
git merge feature/login
```

Interview follow-up: What is a merge conflict?

A merge conflict occurs when Git cannot automatically combine changes, usually because different developers modified the same part of a file. I review the conflicting code, keep the required changes, remove the conflict markers and then commit the resolved version.

Q26 How to open a browser with customized settings?

We use browser-specific **Options classes**.

For Chrome:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--start-maximized");
WebDriver driver =
    new ChromeDriver(options);
```

Incognito

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--incognito");
WebDriver driver =
    new ChromeDriver(options);
```

Headless

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless");
WebDriver driver =
    new ChromeDriver(options);
```

Disable notifications

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--disable-notifications");
WebDriver driver =
    new ChromeDriver(options);
```

Multiple settings

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--start-maximized");
options.addArguments("--incognito");
options.addArguments("--disable-notifications");
WebDriver driver =
    new ChromeDriver(options);
```

Interview answer

I use browser Options classes such as ChromeOptions, FirefoxOptions and EdgeOptions to customize browser behavior. For example, I can run Chrome in headless mode, incognito mode, disable notifications or configure other browser preferences.

Q27 How do you go to the homepage directly if you logged in during the previous session?

There are two concepts here.

If the application maintains login using **cookies/session**, we can save and reuse the cookies.

Save cookies

```
Set<Cookie> cookies = driver.manage().getCookies();
```

We can persist the cookies.

During the next session:

```
for (Cookie cookie : cookies) {
    driver.manage().addCookie(cookie);
}

driver.navigate().refresh();
```

Then navigate directly:

```
driver.get("https://example.com/home");
```

Important point

We normally need to first open the domain before adding cookies:

```
driver.get("https://example.com");
for (Cookie cookie : cookies) {
    driver.manage().addCookie(cookie);
}

driver.navigate().refresh();
```

Interview answer

If the application uses cookie-based authentication, I can preserve the authentication cookies and reuse them in the next browser session. This avoids performing the login steps repeatedly. The exact approach depends on how the application's authentication mechanism works.

Selenium 4 also provides cookie APIs

```
driver.manage().addCookie(cookie);
driver.manage().deleteAllCookies();
driver.manage().getCookies();
```

Q28 Maven build JAR file to push to Git — how do you secure it from viruses or hackers?

This question is slightly tricky.

First important point

Git is not a security mechanism for JAR files.

A JAR generated by Maven is a build artifact. Generally, we should **not commit generated JAR files into Git** unless there is a specific reason.

Instead:

```
Source code
     ↓
    Git
```

```
          ↓
       Jenkins
          ↓
    Maven build
          ↓
         JAR
          ↓
Artifact Repository
```

For example, organizations commonly use an artifact repository such as Nexus, JFrog Artifactory, or a cloud artifact service.

Security practices

1. Don't commit passwords/secrets

Never put:

```
username=admin
password=MyPassword123
```

into Git.

Use:

- Jenkins Credentials
- Environment variables
- Secret management systems
- .env/configuration mechanisms appropriate to the deployment

2. Use .gitignore

For example:

```
target/
*.jar
*.class
```

if generated JARs should not be version controlled.

3. Dependency scanning

Maven dependencies should be scanned for known vulnerabilities.

For example, organizations may use tools such as:

```
OWASP Dependency-Check
Snyk
JFrog Xray
```

4. Malware/virus scanning

Build artifacts can be scanned using the organization's security/antivirus tooling before distribution.

5. Access control

The Git repository should have:

- Authentication
- Role-based access
- Branch protection
- Pull request reviews
- MFA/SSO where applicable

Strong interview answer

I wouldn't normally push Maven-generated JAR files directly into Git. I keep source code in Git and generate the JAR through Jenkins/Maven. The generated artifact is stored in an artifact repository. For security, we avoid committing credentials, use secret management, scan dependencies for vulnerabilities, scan artifacts where required, and control repository access.

This is a much stronger answer than saying "Git protects the JAR from hackers."

Q29 Explain Access Modifiers in Java

Java has **four access levels**:

- public
- protected
- default
- private

1. Public

Accessible from anywhere.

```
public class Login {
    public void login() {
        System.out.println("Login");
    }
}
```

Can be accessed from another package:

```
Login obj = new Login();
obj.login();
```

2. Private

Accessible only inside the same class.

```
public class Login {
    private String password;
    private void validatePassword() {
        System.out.println("Validating");
    }
}
```

Cannot directly access from another class.

3. Protected

Accessible:

- Within the same package
- From subclasses in another package

```
protected void login() {
    System.out.println("Login");
}
```

4. Default

If no access modifier is specified:

```
void login() {
    System.out.println("Login");
}
```

It is accessible only within the **same package**.

Quick table

Modifier	Same Class	Same Package	Subclass	Other Package
public	✔	✔	✔	✔
protected	✔	✔	✔	✘*
default	✔	✔	✘	✘
private	✔	✘	✘	✘

* protected can be accessed from another package through inheritance.

Easy way to remember

```
public     → Everywhere
protected  → Package + Child
```

default → Package

```
private    → Class only
```

Q30 Explain Set and List in Java

Both List and Set are interfaces in the Java Collections Framework.

List

A List:

- Allows duplicates
- Maintains insertion order
- Supports index-based access

Example:

```
List<String> names = new ArrayList<>();
names.add("Vajravel");
names.add("John");
names.add("Vajravel");
System.out.println(names);
```

Output:

[Vajravel, John, Vajravel]

Access by index:

```
System.out.println(names.get(0));
```

Output:

Vajravel

Set

A Set:

- Does not allow duplicate elements
- Generally used when uniqueness is required
- HashSet does not guarantee insertion order

Example:

```
Set<String> names = new HashSet<>();
names.add("Vajravel");
names.add("John");
names.add("Vajravel");
System.out.println(names);
```

The duplicate "Vajravel" will not be stored twice.

Selenium example

This is particularly useful for interview questions.

List with WebElements

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

Because there can be multiple elements, Selenium returns a List<WebElement>.

Set with window handles

This is a **very common Selenium example**:

```
Set<String> windowHandles =
    driver.getWindowHandles();
```

Why Set?

Because window handles must be **unique**.

```
for (String handle : windowHandles) {
    driver.switchTo().window(handle);
}
```

List vs Set

List	Set
Allows duplicates	Does not allow duplicates
Maintains order	Depends on implementation
Index access available	No index-based access
get(index) available	No get(index)
Example: ArrayList	Example: HashSet
Selenium: findElements()	Selenium: getWindowHandles()

Q31 StringBuffer and StringBuilder

What are StringBuffer and StringBuilder?

StringBuffer and StringBuilder are mutable classes in Java used when a string needs to be changed many times. Unlike String, they do not create a new object for every modification.

StringBuffer

- Mutable: append(), insert(), delete(), reverse(), and replace() modify the same object.
- Synchronized: its methods are thread-safe.
- Because of synchronization, it is generally slower than StringBuilder.

- Useful when the same mutable string object is accessed by multiple threads.

StringBuilder

- Mutable and not synchronized.
- Faster than StringBuffer in single-threaded applications.
- Preferred for most normal string manipulation in test automation and application code.
- Introduced in Java 5.

Example

```
StringBuffer buffer = new StringBuffer("Hello");
buffer.append(" Java");
buffer.reverse();
System.out.println(buffer);

StringBuilder builder = new StringBuilder("Hello");
builder.append(" Selenium");
builder.insert(0, "Java ");
System.out.println(builder);
```

Interview comparison

- String: immutable.
- StringBuffer: mutable + synchronized + thread-safe + relatively slower.
- StringBuilder: mutable + not synchronized + faster + preferred for single-threaded use.

Q32 Interface and Abstract Class

Abstract class

An abstract class is a class declared with the abstract keyword. It can contain abstract methods (without implementation) and concrete methods (with implementation). It can also have constructors, instance variables, static members, and implemented methods.

Interface

An interface defines a contract that implementing classes must follow. A class can implement multiple interfaces. Modern Java interfaces can contain abstract, default, static, and private methods.

Key differences

- A class can extend only one class, but it can implement multiple interfaces.
- An abstract class can have instance variables and constructors; an interface does not have normal instance fields or constructors.
- Abstract classes are useful when related classes share common state or implementation.
- Interfaces are useful when different classes need to follow the same capability/contract.
- An abstract class is extended using extends; an interface is implemented using implements.

Example

```
abstract class Vehicle {
    abstract void start();

    void stop() {
        System.out.println("Vehicle stopped");
    }
}
```

```
interface Testable {
    void test();
}

class Car extends Vehicle implements Testable {
    public void start() {
        System.out.println("Car started");
    }

    public void test() {
        System.out.println("Car testing");
    }
}
```

Q33 Sort only the first 4 elements of int[] a = {1,4,8,3,7,6,2}

The requirement is to sort only indexes 0 through 3. Arrays.sort(array, fromIndex, toIndex) sorts the range from fromIndex (inclusive) to toIndex (exclusive). Therefore, use 0 and 4.

```
import java.util.Arrays;

public class SortFirstFour {
    public static void main(String[] args) {
        int[] a = {1, 4, 8, 3, 7, 6, 2};

        Arrays.sort(a, 0, 4);

        System.out.println(Arrays.toString(a));
    }
}

// Output:
// [1, 3, 4, 8, 7, 6, 2]
```

Important interview point

The second argument is exclusive. Arrays.sort(a, 0, 4) sorts positions 0, 1, 2 and 3 only; positions 4 onward remain unchanged.

Q34 Java program to find whether a given number is prime or not

A prime number is a number greater than 1 that has exactly two factors: 1 and itself. For efficiency, we only need to check divisors up to the square root of the number.

```
public class PrimeNumber {
    public static void main(String[] args) {
        int number = 29;
        boolean isPrime = true;

        if (number <= 1) {
            isPrime = false;
        } else {
            for (int i = 2; i * i <= number; i++) {
                if (number % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        if (isPrime) {
            System.out.println(number + " is a prime number");
```

```
        } else {
            System.out.println(number + " is not a prime number");
        }
    }
}
```

Complexity

The optimized approach takes O(√n) time and O(1) extra space.

Q35 Java program to change lowercase letters to uppercase and uppercase to lowercase

Approach

Character.isLowerCase() and Character.isUpperCase() can be used to identify the case. Character.toUpperCase() and Character.toLowerCase() perform the conversion.

```
public class ToggleCase {
    public static void main(String[] args) {
        String input = "Java Selenium 123";
        StringBuilder result = new StringBuilder();

        for (char ch : input.toCharArray()) {
            if (Character.isLowerCase(ch)) {
                result.append(Character.toUpperCase(ch));
            } else if (Character.isUpperCase(ch)) {
                result.append(Character.toLowerCase(ch));
            } else {
                result.append(ch);
            }
        }

        System.out.println(result);
    }
}

// Output:
// jAVA sELENIUM 123
```

Q36 Java program to print a pyramid

Star pyramid

```
public class Pyramid {
    public static void main(String[] args) {
        int rows = 5;

        for (int i = 1; i <= rows; i++) {

            // Print spaces
            for (int j = i; j < rows; j++) {
                System.out.print(" ");
            }

            // Print stars
            for (int j = 1; j <= (2 * i - 1); j++) {
                System.out.print("*");
            }

            System.out.println();
```

```
        }
    }
}

// Output:
//     *
//    ***
//   *****
//  *******
// *********
```

Logic

- The outer loop controls the number of rows.
- The first inner loop prints leading spaces.
- The second inner loop prints 2*i-1 stars.
- After each row, println() moves to the next line.

Q37 Java program to reverse a number

Using modulo and division

```
public class ReverseNumber {
    public static void main(String[] args) {
        int number = 12345;
        int reverse = 0;

        while (number != 0) {
            int digit = number % 10;
            reverse = reverse * 10 + digit;
            number = number / 10;
        }

        System.out.println("Reversed number: " + reverse);
    }
}

// Output:
// Reversed number: 54321
```

Logic

- number % 10 extracts the last digit.
- reverse * 10 + digit adds that digit to the reversed number.
- number / 10 removes the last digit.
- The loop continues until the original number becomes zero.

Q38 Differences between HashSet, HashMap and Hashtable

HashSet

- Stores unique elements.
- Does not store key-value pairs.
- Does not allow duplicate values.
- Allows one null element.
- Not synchronized.

HashMap

- Stores data as key-value pairs.
- Keys must be unique; values can be duplicated.
- Allows one null key and multiple null values.
- Not synchronized.
- Generally preferred over Hashtable for modern non-thread-safe use.

Hashtable

- Stores key-value pairs.
- Synchronized and thread-safe for its individual operations.
- Does not allow null keys or null values.
- It is a legacy collection class.
- Often slower than HashMap because of synchronization.

Comparison

Feature	HashSet	HashMap	Hashtable
Data	Values	Key + Value	Key + Value
Duplicates	Not allowed	Keys: no; Values: yes	Keys: no; Values: yes
Null	One null	One null key; null values	No null key/value
Synchronized	No	No	Yes
Type	Collection	Map	Legacy Map

Q39 Mouse hover, open a browser, go to File and save it

Mouse hover using Selenium

Selenium provides the Actions class for mouse operations such as hover, right-click, drag and drop, and click-and-hold.

```
WebDriver driver = new ChromeDriver();
driver.get("https://example.com");

WebElement menu = driver.findElement(By.id("menu"));
Actions actions = new Actions(driver);

actions.moveToElement(menu).perform();
```

Open a browser and navigate

```
WebDriver driver = new ChromeDriver();
driver.manage().window().maximize();

driver.get("https://example.com");

// Alternative:
driver.navigate().to("https://example.com");
```

Important clarification about File → Save

If the requirement is to open the browser's native File menu and choose Save, Selenium normally cannot directly automate browser chrome/native OS menus because WebDriver controls the web page DOM, not the browser chrome. For downloads, it is better to configure the browser's download preferences or use the application download flow.

Example: configure Chrome download directory

```
ChromeOptions options = new ChromeOptions();

Map<String, Object> prefs = new HashMap<>();
prefs.put("download.default_directory", "C:\\Downloads");
prefs.put("download.prompt_for_download", false);

options.setExperimentalOption("prefs", prefs);

WebDriver driver = new ChromeDriver(options);
driver.get("https://example.com");
```

Interview answer

I would not use Selenium to click the browser's native File → Save menu. Selenium is designed to automate web elements. If the web application provides a download button, I automate that button and configure the browser download preferences. For OS-level native dialogs, a separate desktop automation approach may be required.

Q40 How to handle a tooltip in Selenium?

What is a tooltip?

A tooltip is a small message that appears when the mouse pointer is placed over an element, or sometimes when the element receives focus. Tooltips can be implemented using a title attribute, an ARIA attribute, or a dynamically generated HTML element.

Case 1: Tooltip using title attribute

```
WebElement element = driver.findElement(By.id("helpIcon"));

String tooltip = element.getAttribute("title");

System.out.println("Tooltip: " + tooltip);
```

Case 2: Tooltip displayed in a separate HTML element

```
WebElement icon = driver.findElement(By.id("helpIcon"));

Actions actions = new Actions(driver);
actions.moveToElement(icon).perform();

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement tooltip = wait.until(
    ExpectedConditions.visibilityOfElementLocated(
        By.cssSelector(".tooltip")
    )
);

System.out.println("Tooltip: " + tooltip.getText());
```

Case 3: Angular/Material-style tooltip

Frameworks such as Angular Material may create the tooltip dynamically in an overlay container. Therefore, do not assume the tooltip is a child of the original button. Hover over the element first, then locate the visible tooltip.

```
WebElement icon = driver.findElement(By.cssSelector("[matTooltip]"));

new Actions(driver)
        .moveToElement(icon)
        .perform();

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement tooltip = wait.until(
    ExpectedConditions.visibilityOfElementLocated(
        By.cssSelector(".mat-tooltip")
    )
);
```

Q41 Write a program to read from PDF

Interview explanation

In Java automation, Apache PDFBox is commonly used to read text from PDF files. The basic flow is to load the PDF, create a PDFTextStripper, extract the text, and close the document. In a Maven project, the PDFBox dependency should be added to pom.xml.

Maven dependency

```
<dependency>
    <groupId>org.apache.pdfbox</groupId>
    <artifactId>pdfbox</artifactId>
    <version>3.0.6</version>
</dependency>
```

Java program

```
import java.io.File;
import java.io.IOException;
import org.apache.pdfbox.Loader;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.text.PDFTextStripper;

public class ReadPdf {
    public static void main(String[] args) {
        File file = new File("C:\\TestData\\sample.pdf");

        try (PDDocument document = Loader.loadPDF(file)) {
            PDFTextStripper stripper = new PDFTextStripper();
            String text = stripper.getText(document);

            System.out.println(text);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Interview points

PDFBox can extract text from text-based PDFs.

For scanned/image-only PDFs, OCR may be required because there may be no actual text layer.

try-with-resources automatically closes the PDF document.

In test automation, extracted PDF text can be validated using assertions.

Q42 Write a sorting program for String without using String inbuilt sorting methods

Approach

One simple interview solution is bubble sort. Compare adjacent characters and swap them when they are in the wrong order. This avoids using Arrays.sort(), String sorting utilities, or other built-in sorting methods.

```
public class StringSort {
    public static void main(String[] args) {
        String input = "selenium";
        char[] chars = input.toCharArray();

        for (int i = 0; i < chars.length - 1; i++) {
            for (int j = 0; j < chars.length - 1 - i; j++) {
                if (chars[j] > chars[j + 1]) {
                    char temp = chars[j];
                    chars[j] = chars[j + 1];
                    chars[j + 1] = temp;
                }
            }
        }

        System.out.println("Sorted String: " + new String(chars));
    }
}

// Output:
// eeilmnos
```

Important point

The comparison chars[j] > chars[j + 1] compares Unicode/ASCII character values. This program sorts characters in ascending order. Bubble sort has $O(n^2)$ time complexity.

Q43 Write a Singleton program in Java with Double-Check Locking

What is Singleton?

Singleton is a design pattern that ensures only one instance of a class is created and provides a global access point to that instance.

Double-checked locking

Double-checked locking reduces synchronization overhead. The first null check avoids synchronization after the instance has already been created. The synchronized block performs a second null check before creating the object.

```
public class Singleton {

    private static volatile Singleton instance;

    private Singleton() {
        // Private constructor prevents direct object creation
    }

    public static Singleton getInstance() {
        if (instance == null) {
```

```
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

Why volatile?

volatile prevents problematic instruction reordering and ensures visibility of the initialized reference between threads.

The second null check prevents multiple threads from creating multiple instances.

The constructor is private so external classes cannot use new Singleton().

Usage

```
Singleton obj1 = Singleton.getInstance();
Singleton obj2 = Singleton.getInstance();

System.out.println(obj1 == obj2); // true
```

Q44 What is Garbage Collector in Java?

Definition

Garbage Collection (GC) is Java's automatic memory-management mechanism. It identifies objects that are no longer reachable by the running application and reclaims their heap memory.

How it works

Objects are normally created in the Java heap.

When an object can no longer be reached through live references, it becomes eligible for garbage collection.

The JVM's garbage collector can reclaim the memory occupied by such objects.

The programmer does not explicitly free objects like in C/C++.

System.gc() can request garbage collection, but it does not guarantee that GC will run immediately.

Example

```
public class GarbageDemo {
    public static void main(String[] args) {
        GarbageDemo obj = new GarbageDemo();

        obj = null; // Object becomes eligible for GC

        System.gc(); // Request to JVM; not guaranteed
    }

    @Override
    protected void finalize() throws Throwable {
        System.out.println("Finalize called");
    }
}
```

Interview note: finalize() is deprecated and should not be used in modern Java applications. Use try-with-resources and AutoCloseable for deterministic cleanup of external resources.

Common GC terms

Young generation: commonly contains newly created objects.

Old generation: commonly contains objects that survive longer.

Minor/young collections generally deal with young-generation objects.

Major/full collections can involve older regions and may have higher pause impact depending on the collector.

Q45 How do you achieve multiple inheritance in Java?

Answer

Java does not support multiple inheritance of classes. A class cannot extend two classes because this can create ambiguity, commonly illustrated by the diamond problem. Java achieves multiple inheritance of type/behavior mainly through interfaces.

```
interface Printable {
    void print();
}

interface Showable {
    void show();
}

class Demo implements Printable, Showable {

    public void print() {
        System.out.println("Print method");
    }

    public void show() {
        System.out.println("Show method");
    }
}

public class MultipleInheritanceDemo {
    public static void main(String[] args) {
        Demo obj = new Demo();
        obj.print();
        obj.show();
    }
}
```

Default method conflict

If two interfaces provide conflicting default methods with the same signature, the implementing class must resolve the conflict by overriding the method.

```
interface A {
    default void display() {
        System.out.println("A");
    }
}

interface B {
    default void display() {
        System.out.println("B");
    }
}

class C implements A, B {
    @Override
    public void display() {
        A.super.display(); // Explicitly choose A
```

```
    }
}
```

Q46 Write a Binary Search program in Java using recursion

Prerequisite

Binary search requires the array to be sorted. It repeatedly compares the target with the middle element and searches only the half that can contain the target.

```
public class RecursiveBinarySearch {

    static int binarySearch(int[] arr, int left, int right, int target) {

        if (left > right) {
            return -1;
        }

        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            return mid;
        }

        if (target < arr[mid]) {
            return binarySearch(arr, left, mid - 1, target);
        }

        return binarySearch(arr, mid + 1, right, target);
    }

    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50, 60};
        int target = 40;

        int result = binarySearch(arr, 0, arr.length - 1, target);

        if (result != -1) {
            System.out.println("Element found at index: " + result);
        } else {
            System.out.println("Element not found");
        }
    }
}
```

Complexity

Time complexity: O(log n). Recursive stack space: O(log n).

Q47 Implement a singly linked list, add a new node, and print the list in reverse

Concept

A singly linked list consists of nodes. Each node contains data and a reference to the next node. The last node points to null.

```
public class SinglyLinkedList {

    static class Node {
        int data;
        Node next;
```

```
        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    Node head;

    // Add node at the end
    void add(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            return;
        }

        Node current = head;

        while (current.next != null) {
            current = current.next;
        }

        current.next = newNode;
    }

    // Print normally
    void printList() {
        Node current = head;

        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }

        System.out.println();
    }

    // Print in reverse using recursion
    void printReverse(Node node) {
        if (node == null) {
            return;
        }

        printReverse(node.next);
        System.out.print(node.data + " ");
    }

    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();

        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40); // New node

        System.out.print("Original list: ");
        list.printList();

        System.out.print("Reverse list: ");
        list.printReverse(list.head);
    }
}

// Output:
// Original list: 10 20 30 40
// Reverse list: 40 30 20 10
```

Interview point

The reverse-print method does not physically reverse the links. It uses recursion to visit the nodes to the end first, then prints them while the recursive calls return.

Q48 Write a Java program for Serialization and Deserialization

Definitions

Serialization converts an object's state into a byte stream so it can be stored or transmitted.

Deserialization reconstructs the object from the byte stream.

A class normally implements Serializable to participate in Java's built-in object serialization.

serialVersionUID is recommended to control class-version compatibility.

```
import java.io.*;

class Employee implements Serializable {

    private static final long serialVersionUID = 1L;

    private int id;
    private String name;

    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Employee{id=" + id + ", name='" + name + "'}";
    }
}

public class SerializationDemo {

    public static void main(String[] args) {

        Employee employee = new Employee(101, "Ravi");

        // Serialization
        try (ObjectOutputStream out =
                new ObjectOutputStream(new FileOutputStream("employee.ser"))) {

            out.writeObject(employee);
            System.out.println("Object serialized");

        } catch (IOException e) {
            e.printStackTrace();
        }

        // Deserialization
        try (ObjectInputStream in =
                new ObjectInputStream(new FileInputStream("employee.ser"))) {

            Employee restored = (Employee) in.readObject();
            System.out.println("Object deserialized: " + restored);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

transient keyword

A transient field is not serialized. For example, a password field declared as transient will not be included in the serialized object state.

Q49 Multithreading program to print even and odd numbers using wait() and notify()

Concept

wait() and notify() are methods of Object used for thread coordination. A thread calls wait() while holding the object's monitor, and another thread can call notify() on the same monitor to wake a waiting thread.

```
public class EvenOddUsingWaitNotify {

    private final Object lock = new Object();
    private int number = 1;
    private final int limit = 10;

    public void printOdd() {
        synchronized (lock) {
            while (number <= limit) {

                while (number % 2 == 0) {
                    try {
                        lock.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }

                if (number <= limit) {
                    System.out.println("Odd : " + number);
                    number++;
                    lock.notify();
                }
            }
        }
    }

    public void printEven() {
        synchronized (lock) {
            while (number <= limit) {

                while (number % 2 != 0) {
                    try {
                        lock.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }

                if (number <= limit) {
                    System.out.println("Even: " + number);
                    number++;
                    lock.notify();
                }
            }
        }
    }

    public static void main(String[] args) {
        EvenOddUsingWaitNotify obj = new EvenOddUsingWaitNotify();
```

```
        Thread oddThread = new Thread(obj::printOdd);
        Thread evenThread = new Thread(obj::printEven);

        oddThread.start();
        evenThread.start();
    }
}
```

Important interview points

wait() releases the monitor while the thread waits.

notify() wakes one waiting thread associated with the same monitor.

wait() and notify() must be called while owning the object's monitor, normally inside synchronized code.

Use while rather than if around wait() because a thread should re-check the condition after waking.

InterruptedException should be handled properly by restoring the interrupted status when appropriate.

Q50 Write an email validation program in Java

Approach

A regular expression can be used for basic email-format validation. For production systems, validation should not rely only on a regex because complete email syntax and actual deliverability are more complex.

```
import java.util.regex.Pattern;

public class EmailValidation {

    private static final String EMAIL_REGEX =
            "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";

    public static boolean isValidEmail(String email) {
        return email != null
                && Pattern.matches(EMAIL_REGEX, email);
    }

    public static void main(String[] args) {

        String email1 = "test@example.com";
        String email2 = "invalid-email";

        System.out.println(email1 + " : " + isValidEmail(email1));
        System.out.println(email2 + " : " + isValidEmail(email2));
    }
}

// Output:
// test@example.com : true
// invalid-email : false
```

Q51 Write a Java program to find the maximum number in an array

Traverse the array once and maintain a variable containing the largest value found so far.

```
public class MaximumInArray {
    public static void main(String[] args) {
        int[] numbers = {10, 45, 7, 89, 32, 66};

        int max = numbers[0];

        for (int i = 1; i < numbers.length; i++) {
```

```
            if (numbers[i] > max) {
                max = numbers[i];
            }
        }

        System.out.println("Maximum number: " + max);
    }
}
```

Complexity

Time: O(n). Extra space: O(1). In an interview, mention that the array must contain at least one element for the numbers[0] approach.

Q52 What is a user-defined exception? Create your own exception

A user-defined exception is a custom exception class created by the programmer to represent an application-specific error condition. It normally extends Exception for a checked exception or RuntimeException for an unchecked exception.

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class CustomExceptionDemo {

    static void validateAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or above");
        }
        System.out.println("Eligible");
    }

    public static void main(String[] args) {
        try {
            validateAge(16);
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Interview points

Use throw to explicitly create/throw an exception.

Use throws to declare that a method may pass an exception to its caller.

Extend Exception when you want a checked custom exception; RuntimeException for an unchecked one.

Q53 Find how many times values are duplicated using HashMap

```
import java.util.HashMap;
import java.util.Map;

public class DuplicateCount {
    public static void main(String[] args) {
        String[] values = {
            "111", "222", "333", "444", "555",
            "222", "111", "333", "111", "222"
        };
```

```
        Map<String, Integer> countMap = new HashMap<>();

        for (String value : values) {
            countMap.put(value, countMap.getOrDefault(value, 0) + 1);
        }

        for (Map.Entry<String, Integer> entry : countMap.entrySet()) {
            System.out.println(entry.getKey() + " -> " + entry.getValue());
        }
    }
}

// Expected counts:
// 111 -> 3
// 222 -> 3
// 333 -> 2
// 444 -> 1
// 555 -> 1
```

How to print only duplicates

```
for (Map.Entry<String, Integer> entry : countMap.entrySet()) {
    if (entry.getValue() > 1) {
        System.out.println(entry.getKey() + " duplicated "
                + entry.getValue() + " times");
    }
}
```

Q54 TestNG XML structure with parallel="tests" and parallel="classes"

TestNG uses testng.xml to define suites, tests, classes, groups, parameters and parallel execution. The parallel value controls what TestNG executes concurrently.

parallel="tests"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM
  "https://testng.org/testng-1.0.dtd">

<suite name="RegressionSuite" parallel="tests" thread-count="3">

    <test name="LoginTests">
        <classes>
            <class name="tests.LoginTest"/>
        </classes>
    </test>

    <test name="SearchTests">
        <classes>
            <class name="tests.SearchTest"/>
        </classes>
    </test>

    <test name="CheckoutTests">
        <classes>
            <class name="tests.CheckoutTest"/>
        </classes>
    </test>

</suite>
```

parallel="tests" runs separate <test> sections in parallel, subject to thread-count and other TestNG constraints.

parallel="classes"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM
  "https://testng.org/testng-1.0.dtd">

<suite name="RegressionSuite" parallel="classes" thread-count="3">

    <test name="Regression">
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.SearchTest"/>
            <class name="tests.CheckoutTest"/>
        </classes>
    </test>

</suite>
```

parallel="classes" runs classes in parallel. Methods inside the same class are not automatically made parallel just because classes are parallel.

Q55 How do you handle dependent test cases in TestNG?

TestNG supports dependencies using dependsOnMethods and dependsOnGroups. A dependent test runs only when its dependency has successfully completed.

```
import org.testng.annotations.Test;

public class DependencyTest {

    @Test
    public void login() {
        System.out.println("Login successful");
    }

    @Test(dependsOnMethods = "login")
    public void search() {
        System.out.println("Search executed");
    }

    @Test(dependsOnMethods = "search")
    public void checkout() {
        System.out.println("Checkout executed");
    }
}
```

Group dependency

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

@Test(dependsOnGroups = "login")
public void search() {}
```

If a prerequisite test fails, dependent tests are generally skipped. In good automation design, avoid unnecessary test dependencies because tests should ideally be independent and repeatable.

Q56 Programs: String palindrome, integer palindrome, HashMap/Hashtable, threads, duplicates/non-duplicates

1. String palindrome

```
public class StringPalindrome {
    public static void main(String[] args) {
        String input = "madam";
        String reverse = "";

        for (int i = input.length() - 1; i >= 0; i--) {
            reverse += input.charAt(i);
        }

        System.out.println(input.equals(reverse)
                ? "Palindrome" : "Not Palindrome");
    }
}
```

2. Integer palindrome

```
public class NumberPalindrome {
    public static void main(String[] args) {
        int number = 1221;
        int original = number;
        int reverse = 0;

        while (number != 0) {
            int digit = number % 10;
            reverse = reverse * 10 + digit;
            number /= 10;
        }

        System.out.println(original == reverse
                ? "Palindrome" : "Not Palindrome");
    }
}
```

3. HashMap and Hashtable

```
import java.util.HashMap;
import java.util.Hashtable;

public class MapExample {
    public static void main(String[] args) {

        HashMap<Integer, String> map = new HashMap<>();
        map.put(1, "Java");
        map.put(2, "Selenium");
        map.put(null, "Allowed");
        System.out.println(map);

        Hashtable<Integer, String> table = new Hashtable<>();
        table.put(1, "Java");
        table.put(2, "Selenium");
        System.out.println(table);

        // Hashtable does not permit null keys or null values.
    }
}
```

4. Sample thread code

```
class MyTask extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName()
                    + " : " + i);
        }
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        MyTask t1 = new MyTask();
        MyTask t2 = new MyTask();

        t1.start();
        t2.start();
    }
}
```

5. Store duplicates and non-duplicates separately

```
import java.util.*;

public class DuplicateSeparate {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 4, 5, 1, 6};

        Map<Integer, Integer> counts = new LinkedHashMap<>();

        for (int n : arr) {
            counts.put(n, counts.getOrDefault(n, 0) + 1);
        }

        List<Integer> duplicates = new ArrayList<>();
        List<Integer> nonDuplicates = new ArrayList<>();

        for (Map.Entry<Integer, Integer> entry : counts.entrySet()) {
            if (entry.getValue() > 1) {
                duplicates.add(entry.getKey());
            } else {
                nonDuplicates.add(entry.getKey());
            }
        }

        System.out.println("Duplicates: " + duplicates);
        System.out.println("Non-duplicates: " + nonDuplicates);
    }
}
```

Q57 What framework are you currently using? Cucumber, POM Hybrid

Interview-ready answer:

I am currently using a hybrid automation framework based on Selenium with Java, Cucumber and Page Object Model. Cucumber provides BDD-style feature files and step definitions. POM keeps page locators and page actions separate from test steps. The framework can also include reusable utilities, configuration management, test data handling, reporting, hooks, and Maven for dependency/build management.

Typical framework structure

src/test/java

```
features/
stepDefinitions/
pages/
runners/
utilities/
hooks/
testData/
```

pom.xml
testng.xml / cucumber configuration
reports/

Benefits

Readable Gherkin scenarios.

Reusable page objects.

Centralized locators and actions.

Reusable utilities for waits, screenshots, files and test data.

Easy CI execution through Maven/Jenkins.

Better maintainability and separation of responsibilities.

Q58 How do you handle wait for a particular element on a webpage?

Prefer explicit waits when waiting for a particular element or condition. Explicit waits poll for a condition until it is satisfied or the timeout is reached.

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement element = wait.until(
    ExpectedConditions.elementToBeClickable(
        By.id("submit")
    )
);

element.click();
```

Common conditions

visibilityOfElementLocated – element is present and visible.

elementToBeClickable – element is visible and enabled.

presenceOfElementLocated – element exists in the DOM.

invisibilityOfElementLocated – useful for loaders/spinners.

textToBePresentInElement – waits for expected text.

Avoid excessive Thread.sleep(). It always waits the full duration and can make tests slower and flaky.

Q59 How do you fetch values from a web table?

For an HTML table, locate rows and cells and iterate through them. For dynamic tables, use relative XPath/CSS locators and explicit waits.

```
WebElement table = driver.findElement(By.id("employeeTable"));

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

```
for (WebElement row : rows) {
    List<WebElement> cells =
            row.findElements(By.cssSelector("td"));

    for (WebElement cell : cells) {
        System.out.print(cell.getText() + " | ");
    }
    System.out.println();
}
```

Find a value in a particular column

```
for (WebElement row : rows) {
    String name = row.findElement(
            By.cssSelector("td:nth-child(1)")
    ).getText();

    String status = row.findElement(
            By.cssSelector("td:nth-child(3)")
    ).getText();

    if (name.equals("John")) {
        System.out.println("Status: " + status);
        break;
    }
}
```

For pagination, handle the current page, inspect whether a Next button is enabled, click it, wait for the table to refresh, and continue.

Q60 Difference between JVM and JRE

JDK

```
├── JRE
│    └── JVM
└── Development tools
     (javac, javadoc, jar, etc.)
```

JVM (Java Virtual Machine): executes Java bytecode and provides runtime services such as memory management and garbage collection.

JRE (Java Runtime Environment): the environment needed to run Java applications; conceptually includes the JVM plus runtime libraries.

JDK (Java Development Kit): development environment containing tools for compiling/developing Java applications plus the runtime components.

Interview shortcut: JVM runs bytecode; JRE provides the runtime environment; JDK is for developing Java applications.

Q61 Difference between ArrayList and Array

Feature	Array	ArrayList
Size	Fixed after creation	Dynamic
Data	Primitives or objects	Objects; primitive types use wrappers
Access	array[index]	get(index)

Feature	Array	ArrayList
Methods	Very few built-in operations	add, remove, contains, etc.
Generics	Not applicable	Supported

Use arrays when size is fixed and direct primitive-array performance is useful. Use ArrayList when the collection size changes or collection APIs are useful.

Q62 Use of List, Set and Map in Collections

List: ordered collection; duplicates are allowed. Examples: ArrayList, LinkedList.

Set: unique elements; duplicates are not allowed. Examples: HashSet, LinkedHashSet, TreeSet.

Map: key-value association; keys are unique while values may be duplicated. Examples: HashMap, LinkedHashMap, TreeMap.

```
List<String> list = new ArrayList<>();
list.add("Java");
list.add("Java");

Set<String> set = new HashSet<>();
set.add("Java");
set.add("Java");

Map<Integer, String> map = new HashMap<>();
map.put(1, "Java");
map.put(2, "Selenium");
```

A practical automation example: List for rows/elements, Set for removing duplicate IDs, and Map for test-data key/value pairs.

Q63 OOP concepts with example and constructor

Four major OOP concepts

Encapsulation: keeping data and behavior together and controlling access, commonly with private fields and public methods.

Inheritance: a child class reuses/extends behavior of a parent class.

Polymorphism: the same interface/reference can represent different implementations; includes overloading and overriding.

Abstraction: exposing essential behavior while hiding implementation details using abstract classes/interfaces.

```
class Employee {
    private int id;              // Encapsulation

    Employee(int id) {           // Constructor
        this.id = id;
    }

    public int getId() {
        return id;
    }
}

class Manager extends Employee {  // Inheritance
    Manager(int id) {
        super(id);
```

```
    }
}

interface Report {
    void generate();             // Abstraction
}

class PdfReport implements Report {
    public void generate() {
        System.out.println("PDF report");
    }
}
```

Constructor

A constructor initializes an object. It has the same name as the class and no return type. Constructors can be overloaded. If no constructor is declared, Java may provide a default no-argument constructor.

Q64 Method overloading and method overriding

Overloading

Same method name with different parameter lists in the same class or inheritance context. It is compile-time polymorphism.

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

Overriding

A subclass provides its own implementation of an inherited instance method with a compatible signature. It is runtime polymorphism.

```
class Parent {
    void display() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    @Override
    void display() {
        System.out.println("Child");
    }
}
```

Overloading: different parameters; compile-time decision.

Overriding: same signature in subclass; runtime dispatch.

A static method is hidden rather than overridden.

Q65 Assert, Verify and waitFor

Assert

An assertion checks whether an expected condition is true. In Selenium/TestNG frameworks, a failed assertion can fail the test and, for hard assertions, stop the remaining statements in that test method.

```
Assert.assertEquals(actual, expected);
Assert.assertTrue(condition);
```

Verify

In test automation, 'verify' usually refers to a soft validation where execution continues after a failure and failures are reported later. TestNG SoftAssert is a common implementation.

```
SoftAssert soft = new SoftAssert();
soft.assertEquals(actual, expected);
soft.assertTrue(condition);
soft.assertAll();
```

waitFor

waitFor is not one universal Selenium Java command. It generally means waiting for a condition, often through WebDriverWait/ExpectedConditions. In some frameworks or tools, a method named waitFor may be a custom utility.

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(
    By.id("message")));
```

Q66 super and final keyword

super

super() calls the parent constructor.

super.method() calls a parent-class method.

super.variable accesses a parent field when it is hidden by a child field.

```
class Parent {
    int x = 10;

    Parent() {
        System.out.println("Parent constructor");
    }

    void display() {
        System.out.println("Parent display");
    }
}

class Child extends Parent {
    int x = 20;

    Child() {
        super();
    }

    void test() {
        System.out.println(super.x);
        super.display();
    }
```

```
}
```

final

final variable: cannot be reassigned after initialization.

final method: cannot be overridden.

final class: cannot be extended.

```
final int MAX = 100;

final class Utility {
    // cannot be extended
}

class Parent {
    final void show() {}
}
```

Q67 Is String immutable?

Yes. Java String objects are immutable: once a String object is created, its character content cannot be changed. Operations such as concat() create a new String object instead of modifying the original.

```
String s = "Java";
s.concat(" Selenium");
System.out.println(s); // Java
s = s.concat(" Selenium");
System.out.println(s); // Java Selenium
```

Why immutable?

Security: strings are commonly used for paths, class names and configuration values.

String pool: immutable strings can be safely shared.

Thread-safety: immutable objects can be shared without synchronization for their state.

Hashing: a String's content/hash can remain stable while it is used as a HashMap key.

Q68 Difference between == and .equals()

For objects, == compares whether two references point to the same object. equals() compares logical equality when the class overrides it. String overrides equals() to compare character content.

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

System.out.println(a == b);      // false
System.out.println(a.equals(b)); // true
```

For primitives, == compares values directly. For Strings, use equals() for content comparison rather than ==.

Q69 What is 'this' in Java?

this is a reference to the current object. It is commonly used to distinguish instance variables from parameters, call another constructor in the same class, or pass/return the current object.

```
class Employee {
    private int id;
    private String name;

    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    Employee() {
        this(0, "Unknown");
    }

    Employee getObject() {
        return this;
    }
}
```

this.id means the current object's instance variable.

this(...) invokes another constructor in the same class and must be the first constructor statement.

this cannot be used in a static context to refer to an instance.

Q70 XPath: preceding, following-sibling and axes – when do we use them?

XPath axes allow you to locate elements based on their relationship to the current node. They are useful when stable attributes are unavailable and an element can be located relative to a label, parent, sibling, or nearby structure.

Common axes

Axis	Purpose
parent	Select immediate parent
child	Select child nodes
ancestor	Select ancestors
descendant	Select descendants
following-sibling	Select siblings after the current node
preceding-sibling	Select siblings before the current node
following	Nodes appearing after the current node in document order, outside the descendant relationship
preceding	Nodes appearing before the current node in document order, outside the ancestor relationship
ancestor-or-self	Ancestor nodes plus current node
descendant-or-self	Descendants plus current node

Examples

```
// Input following its label
//label[normalize-space()='Email']/following::input[1]

// Button after a particular sibling
//span[normalize-space()='Submit']/following-sibling::button[1]

// Input before a label
//label[normalize-space()='Email']/preceding-sibling::input[1]

// Element inside the same parent
//div[@class='row']//label[text()='Email']/ancestor::div[@class='row']//input
```

Use relative XPath carefully. Prefer stable IDs, data-testid attributes, accessible roles, or other robust selectors when available.

Q71 PageFactory: use, where to use it, and advantages

PageFactory is a Selenium support pattern that initializes Page Object fields, commonly using @FindBy. It has historically been used to reduce repeated findElement calls and keep page locators organized.

```
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 {

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

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

    @FindBy(id = "login")
    WebElement loginButton;

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

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

Advantages

Centralizes page elements.

Reduces repetitive locator code.

Improves readability of Page Object classes.

Works well with POM for smaller/traditional Selenium frameworks.

Modern note

PageFactory is optional, not required for POM. Many modern Selenium frameworks prefer By locators or Selenium 4-style page objects because explicit element lookup can make timing and stale-element behavior easier to reason about.

Q72 What is TestNG? Why do you use it? Advantages

TestNG is a Java testing framework used to organize, execute, group, parameterize and report tests. It is widely used with Selenium automation.

```
import org.testng.annotations.Test;

public class LoginTest {

    @Test
    public void validLogin() {
        System.out.println("Test executed");
    }
}
```

Advantages

Annotations such as @BeforeMethod, @AfterMethod, @BeforeClass and @AfterClass.

Assertions and SoftAssert.

Groups and dependencies.

DataProvider for data-driven tests.

Parallel execution.

testng.xml suite configuration.

Listeners and reporting integration.

Easy Maven/CI integration.

Q73 What is Maven? What is the purpose of pom.xml?

Maven is a build and dependency-management tool. It can compile code, resolve dependencies, run tests, package applications, and support standardized project lifecycles.

pom.xml

The Project Object Model file is the central Maven configuration file. It defines project coordinates, dependencies, plugins, properties, repositories when needed, and build/test configuration.

```
<project>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>selenium-framework</artifactId>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>YOUR_VERSION</version>
        </dependency>
    </dependencies>
</project>
```

mvn test – commonly runs tests configured for the project.

Maven downloads dependencies from repositories and caches them locally.

Plugins extend Maven's build lifecycle.

Q74 Have you used Jenkins? Why do you use it?

Interview-ready answer:

Yes. I use Jenkins as a CI/CD automation server to execute automated test suites outside the developer's local machine. For a Selenium/Cucumber framework, Jenkins can pull the latest code from Git, install dependencies using Maven, execute the required test tags/suites, archive reports and screenshots, and notify the team when a build fails.

Typical flow

```
          Git checkout
                ↓
        Maven clean/test
                ↓
   Cucumber/TestNG execution
                ↓
     Reports + screenshots
                ↓
    Archive/publish results
                ↓
Email/Teams/Slack notification
```

Scheduled nightly regression.

Execution on code changes.

Parameterized jobs for environment/browser/tag.

Centralized test reports and build history.

Integration with source control and deployment pipelines.

Q75 Two Submit buttons: click the second one when it is not visible and scroll down

First identify the second button, then scroll it into view and click it. Selenium's JavaScript-independent approach is normally to use WebElement.scrollIntoView or Actions/JavaScript when necessary.

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

WebElement secondButton = buttons.get(1);
```

((JavascriptExecutor) driver).executeScript(

```
    "arguments[0].scrollIntoView({block:'center'});",
    secondButton
);

new WebDriverWait(driver, Duration.ofSeconds(10))
    .until(ExpectedConditions.elementToBeClickable(secondButton))
    .click();
```

Alternative

```
WebElement secondButton = driver.findElements(
    By.xpath("//button[normalize-space()='Submit']")
).get(1);

new Actions(driver)
    .scrollToElement(secondButton)
```

```
        .perform();

    secondButton.click();
```

If there are multiple identical buttons, use a better locator based on surrounding section/parent rather than relying only on index.

Q76 When should you use Robot, Actions and JavascriptExecutor?

Tool	Use when	Examples
Actions	Web page mouse/keyboard interactions	hover, drag-drop, right-click, key combinations
Robot	OS-level/native keyboard and mouse events	native file chooser, OS dialog interactions
JavascriptExecutor	Need to execute JavaScript in browser page context	scroll, DOM-related workarounds, inspect/modify page state when appropriate

Examples

```
// Actions - hover
new Actions(driver)
    .moveToElement(menu)
    .perform();

// Robot - native keyboard example
Robot robot = new Robot();
robot.keyPress(KeyEvent.VK_ENTER);
robot.keyRelease(KeyEvent.VK_ENTER);

// JavaScript - scroll
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("arguments[0].scrollIntoView({block:'center'});", element);
```

Prefer normal Selenium APIs first. Use JavaScript as a targeted workaround rather than the default click strategy.

Q77 What is a Marker Interface?

A marker interface is an interface with no methods or fields intended to mark a class as having a particular capability or semantic property. The classic Java example is Serializable.

```
import java.io.Serializable;

class Employee implements Serializable {
    private int id;
    private String name;
}
```

Serializable marks a class as eligible for Java serialization.

Cloneable is another classic marker interface.

Marker interfaces communicate metadata/capability through the type system.

Modern Java also provides annotations for metadata, so marker interfaces are used more selectively than in some older designs.

Q78 How do you find broken links?

A common Selenium approach is to collect href values from anchors and send HTTP requests to verify the response status. The browser itself is not the best tool for validating every link; an HTTP client is more efficient.

```
List<WebElement> links =
        driver.findElements(By.tagName("a"));

for (WebElement link : links) {
    String href = link.getAttribute("href");

    if (href == null || href.isBlank()) {
        continue;
    }

    try {
        HttpURLConnection connection =
                (HttpURLConnection) new URL(href).openConnection();

        connection.setRequestMethod("HEAD");
        connection.setConnectTimeout(5000);
        connection.connect();

        int status = connection.getResponseCode();

        if (status >= 400) {
            System.out.println("Broken: " + href
                    + " -> " + status);
        } else {
            System.out.println("Valid: " + href
                    + " -> " + status);
        }

        connection.disconnect();

    } catch (Exception e) {
        System.out.println("Unable to validate: " + href);
    }
}
```

Interview considerations

Skip javascript:, mailto:, tel: and other non-HTTP links when appropriate.

Some servers block HEAD requests; GET may be required for certain endpoints.

Redirects and authentication can affect status codes.

A 3xx response is not necessarily a broken link; follow redirects according to the validation requirement.

Q79 Listeners concept in TestNG

TestNG listeners allow code to react to test execution events such as test start, success, failure, skip, suite start and finish. They are useful for screenshots, logging, custom reports and failure diagnostics.

```
import org.testng.ITestListener;
import org.testng.ITestResult;

public class TestListener implements ITestListener {

    @Override
    public void onTestStart(ITestResult result) {
        System.out.println("START: "
                + result.getName());
    }
```

```
    @Override
    public void onTestSuccess(ITestResult result) {
        System.out.println("PASS: "
                + result.getName());
    }

    @Override
    public void onTestFailure(ITestResult result) {
        System.out.println("FAIL: "
                + result.getName());
        // Capture screenshot here
    }

    @Override
    public void onTestSkipped(ITestResult result) {
        System.out.println("SKIP: "
                + result.getName());
    }
}
```

Register listener

```
@Listeners(TestListener.class)
public class LoginTest {

    @Test
    public void login() {
        // test
    }
}
```

Listeners are particularly useful in Selenium frameworks for taking a screenshot automatically when a test fails and attaching it to reports.

Q80 Exception concept in Java

An exception is an event that disrupts the normal flow of program execution. Java provides structured exception handling using try, catch, finally, throw and throws.

Hierarchy

Throwable

```
├── Error
└── Exception
    ├── RuntimeException
    └── Other checked exceptions
```

Checked vs unchecked

Checked exceptions are checked by the compiler, such as IOException. They must be caught or declared with throws.

Unchecked exceptions extend RuntimeException, such as NullPointerException, IllegalArgumentException and ArithmeticException.

Errors represent serious JVM/system-level problems and are generally not intended to be handled as normal application exceptions.

```
public class ExceptionDemo {

    public static void main(String[] args) {

        try {
            int result = 10 / 0;
```

```
            System.out.println(result);

        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero");

        } finally {
            System.out.println("Finally block executed");
        }
    }
}
```

throw vs throws

```
// throw: explicitly throws an exception
throw new IllegalArgumentException("Invalid value");

// throws: declares a possible exception
void readFile() throws IOException {
    // file operation
}
```

Custom exception

```
class InvalidUserException extends RuntimeException {
    InvalidUserException(String message) {
        super(message);
    }
}
```

Automation interview examples

NoSuchElementException: locator/element was not found at the time of lookup.

StaleElementReferenceException: previously located element is no longer attached to the current DOM.

TimeoutException: an explicit wait condition was not satisfied within the timeout.

ElementClickInterceptedException: another element/overlay prevents the click.

WebDriverException: broader WebDriver-level failure.

Good automation code handles expected transient conditions with appropriate waits and recovery, while unexpected failures should be reported rather than hidden.

Q81 How to handle a secured HTTPS site – Firefox profile

If a test environment uses HTTPS with an untrusted/self-signed certificate, Firefox can be configured through FirefoxOptions. In Selenium, accepting insecure certificates is generally preferred for test environments rather than manually interacting with the browser certificate page.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.firefox.FirefoxProfile;
import org.openqa.selenium.firefox.FirefoxDriver;

FirefoxProfile profile = new FirefoxProfile();
profile.setAcceptUntrustedCertificates(true);
profile.setAssumeUntrustedCertificateIssuer(false);

FirefoxOptions options = new FirefoxOptions();
options.setProfile(profile);
options.setAcceptInsecureCerts(true);

WebDriver driver = new FirefoxDriver(options);
driver.get("https://test-environment.example.com");
```

Interview points

Use this mainly in controlled test/UAT environments.

Do not disable certificate validation in production as a workaround.

Selenium 4 also supports options.setAcceptInsecureCerts(true).

If authentication is HTTP Basic rather than a certificate issue, handle authentication separately.

Q82 Write code for connecting to a database

Java Database Connectivity (JDBC) is used to connect Java code to relational databases. The general sequence is: load/configure driver, create Connection, create PreparedStatement/Statement, execute SQL, read ResultSet, and close resources.

```
import java.sql.*;

public class DatabaseConnection {
    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/testdb";
        String user = "root";
        String password = "password";

        String sql = "SELECT id, name FROM employee WHERE id = ?";

        try (Connection con =
                DriverManager.getConnection(url, user, password);
                PreparedStatement ps = con.prepareStatement(sql)) {

            ps.setInt(1, 101);

            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    System.out.println(
                            rs.getInt("id") + " - " +
                            rs.getString("name")
                    );
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Automation usage

In UI/API automation, database validation can be used to compare UI values against backend records, verify test-data creation, or clean up data after tests. Use configuration/secret management instead of hard-coding real credentials.

Q83 What challenges did you face while creating the automation framework?

Interview-ready answer:

While creating a Selenium + Java + Cucumber + POM hybrid framework, the main challenges I faced were maintainability, synchronization, test-data management, reusable components, parallel execution, reporting and CI stability.

Challenges and solutions

Dynamic elements: used stable locators, relative XPath/CSS and reusable locator strategies.

Synchronization: introduced explicit waits and reusable wait utilities instead of unnecessary sleeps.

Page maintenance: centralized page locators/actions through Page Objects.

Test data: separated test data from step definitions and supported external files/configuration.

Parallel execution: isolated WebDriver instances per test/thread and avoided shared mutable state.

Failure debugging: added screenshots, logs and reports through hooks/listeners.

Environment management: externalized URLs and environment-specific configuration.

Flaky tests: identified timing, stale elements, overlays and asynchronous UI behavior and handled them explicitly.

CI execution: integrated Maven test execution and reports into Jenkins.

Cucumber step duplication: created reusable generic steps/utilities and kept business steps meaningful.

Strong interview closing

The biggest lesson was to design the framework for maintainability first: reusable utilities, clean page objects, independent tests, good reporting and minimal duplication.

Q84 Separate alphabets, numbers and special characters from a string

```
public class SeparateCharacters {
    public static void main(String[] args) {

        String input = "Java123@#Selenium45!";
        StringBuilder alphabets = new StringBuilder();
        StringBuilder numbers = new StringBuilder();
        StringBuilder special = new StringBuilder();

        for (char ch : input.toCharArray()) {
            if (Character.isLetter(ch)) {
                alphabets.append(ch);
            } else if (Character.isDigit(ch)) {
                numbers.append(ch);
            } else {
                special.append(ch);
            }
        }

        System.out.println("Alphabets: " + alphabets);
        System.out.println("Numbers: " + numbers);
        System.out.println("Special: " + special);
    }
}
```

Q85 Copy all contents from one Excel sheet to another sheet

Apache POI can read and write .xlsx workbooks. The program below copies cells from a source sheet into a destination sheet, preserving common cell types. For a production utility, styles, merged cells, formulas and comments may also need to be copied.

```
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import java.io.*;
```

```
public class CopyExcelSheet {
    public static void main(String[] args) throws Exception {

        FileInputStream input =
            new FileInputStream("source.xlsx");

        Workbook workbook = new XSSFWorkbook(input);

        Sheet source = workbook.getSheet("Sheet1");
        Sheet target = workbook.createSheet("Copy");

        for (Row sourceRow : source) {
            Row targetRow = target.createRow(sourceRow.getRowNum());

            for (Cell sourceCell : sourceRow) {
                Cell targetCell =
                    targetRow.createCell(sourceCell.getColumnIndex());

                switch (sourceCell.getCellType()) {
                    case STRING:
                        targetCell.setCellValue(
                            sourceCell.getStringCellValue());
                        break;

                    case NUMERIC:
                        targetCell.setCellValue(
                            sourceCell.getNumericCellValue());
                        break;

                    case BOOLEAN:
                        targetCell.setCellValue(
                            sourceCell.getBooleanCellValue());
                        break;

                    case FORMULA:
                        targetCell.setCellFormula(
                            sourceCell.getCellFormula());
                        break;

                    default:
                        break;
                }
            }
        }

        input.close();

        try (FileOutputStream output =
                  new FileOutputStream("source.xlsx")) {
            workbook.write(output);
        }

        workbook.close();
    }
}
```

For real projects, avoid overwriting the source workbook unless required; write to a separate output file when possible.

Q86 Frames in Selenium – click an element in a nested frame

An iframe has its own browsing context. Selenium must switch into the frame before locating elements inside it. For nested frames, switch into the parent frame first and then the child frame.

```
// Outer frame
driver.switchTo().frame("outerFrame");

// Inner/nested frame
WebElement innerFrame =
    driver.findElement(By.id("innerFrame"));

driver.switchTo().frame(innerFrame);

// Element inside nested frame
driver.findElement(By.id("submit")).click();

// Return to parent document
driver.switchTo().defaultContent();
```

Important methods

```
switchTo().frame(String nameOrId)

switchTo().frame(int index)

switchTo().frame(WebElement frameElement)
```

parentFrame() returns to the immediate parent frame.

defaultContent() returns to the main page.

Q87 Print numbers in reverse order and omit odd numbers

If the requirement means reverse the digits and print only even digits, check each extracted digit before printing.

```
public class ReverseEvenDigits {
    public static void main(String[] args) {

        int number = 123456789;

        while (number != 0) {
            int digit = number % 10;

            if (digit % 2 == 0) {
                System.out.print(digit + " ");
            }

            number /= 10;
        }
    }
}

// Output:
// 8 6 4 2
```

Q88 How do you know whether an element is visible on the page?

Use WebElement.isDisplayed(). It returns true when Selenium considers the element displayed and false when it is not displayed.

```
WebElement element =
        driver.findElement(By.id("submit"));

if (element.isDisplayed()) {
    System.out.println("Element is visible");
} else {
    System.out.println("Element is not visible");
}
```

With an explicit wait

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(10));

boolean visible = wait.until(
    ExpectedConditions.visibilityOfElementLocated(
        By.id("submit")
    )
).isDisplayed();
```

If the locator itself is absent, findElement throws NoSuchElementException. For a safe check without exception, findElements() and check whether the list is non-empty, or use a wait according to the test requirement.

Q89 How will you run failed cases in TestNG?

There are several approaches. The most common is to use TestNG's generated failed-test XML or an IRetryAnalyzer for transient failures. In a CI pipeline, you can also execute a separate failed-suite step.

Retry Analyzer example

```
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class RetryAnalyzer implements IRetryAnalyzer {

    private int count = 0;
    private static final int MAX_RETRY = 2;

    @Override
    public boolean retry(ITestResult result) {
        if (count < MAX_RETRY) {
            count++;
            return true;
        }
        return false;
    }
}

@Test(retryAnalyzer = RetryAnalyzer.class)
public void testLogin() {
    // test code
}
```

Important interview point

Retries should be used carefully. They can help with genuinely transient failures, but they should not hide real application defects or poor synchronization.

Q90 TestNG priority and dependency order – Priority 1, dependsOn A priority 2, and C priority -34

Suppose A has priority=1, B has priority=2 and dependsOnMethods=A, and C has priority=-34.

```
@Test(priority = 1)
public void A() {}

@Test(priority = 2, dependsOnMethods = "A")
public void B() {}

@Test(priority = -34)
public void C() {}
```

Expected order: C → A → B.

C has the lowest priority (-34), so it is scheduled first among these independent tests.

A then runs because it has priority 1.

B depends on A, so B cannot execute before A, regardless of its priority 2.

Dependency constraints take precedence over simply sorting all methods by priority.

For interview questions like this, always consider dependencies first, then priority among methods that are otherwise eligible to run.

Q91 Run entire package and exclude only two methods in a class without changing code

Use TestNG XML to include the package and exclude methods by name. This changes the suite configuration rather than the Java source code.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM
  "https://testng.org/testng-1.0.dtd">

<suite name="PackageSuite">

    <test name="AllExceptTwo">
        <packages>
            <package name="tests"/>
        </packages>

        <methods>
            <exclude name="methodToExclude1"/>
            <exclude name="methodToExclude2"/>
        </methods>
    </test>

</suite>
```

Note: TestNG method exclusion is typically configured under the relevant <class> definition when selecting individual classes/methods. For broad package selection, another robust approach is to use groups: assign groups to tests you want to exclude/include and configure them in testng.xml.

```
<classes>
    <class name="tests.LoginTest">
        <methods>
            <exclude name="methodToExclude1"/>
            <exclude name="methodToExclude2"/>
        </methods>
```

```
    </class>
</classes>
```

If the interviewer asks specifically for an entire package with exclusions, explain that package selection and per-class method exclusion may need explicit class entries depending on the desired suite structure; groups are often cleaner for large suites.

Q92 Static keyword in Java

static indicates that a member belongs to the class rather than to each individual object.

```
class Counter {
    static int count = 0;

    Counter() {
        count++;
    }

    static void showCount() {
        System.out.println(count);
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        new Counter();
        new Counter();
        Counter.showCount(); // 2
    }
}
```

Static variable: one shared copy per class.

Static method: can be called using the class name and cannot directly access instance members.

Static block: executes when the class is initialized.

Static nested class: a nested class that does not require an outer-class instance.

Q93 Run some testcases in parallel and some sequentially

Use separate TestNG <test> sections and configure parallel execution only for the tests that can safely run concurrently. Keep dependent/shared-state tests in a sequential section.

```
<suite name="MixedExecution" parallel="tests" thread-count="3">

    <test name="Parallel_Login">
        <classes>
            <class name="tests.LoginTest"/>
        </classes>
    </test>

    <test name="Parallel_Search">
        <classes>
            <class name="tests.SearchTest"/>
        </classes>
    </test>

    <test name="Sequential_Flow">
        <classes>
            <class name="tests.CreateTest"/>
            <class name="tests.UpdateTest"/>
            <class name="tests.DeleteTest"/>
```

```
        </classes>
    </test>

</suite>
```

For true sequential business flow, use dependencies where appropriate and ensure WebDriver/test data is isolated. Parallel execution should not be enabled blindly when tests share accounts, files, database records or static state.

Q94 Verify checkbox is selected; if not, select it

```
WebElement checkbox =
        driver.findElement(By.id("terms"));

if (!checkbox.isSelected()) {
    checkbox.click();
}

System.out.println(
        "Selected: " + checkbox.isSelected()
);
```

For custom checkbox controls where the input is hidden, locate the associated visible label/container or use an accessibility/state attribute such as aria-checked when appropriate.

Q95 CSV and XML file format reader

CSV reader

```
import java.io.*;
import java.nio.file.*;

public class CsvReader {
    public static void main(String[] args) throws Exception {

        try (BufferedReader br =
                Files.newBufferedReader(
                Path.of("testdata.csv"))) {

            String line;

            while ((line = br.readLine()) != null) {
                String[] values = line.split(",");
                for (String value : values) {
                    System.out.print(value.trim() + " | ");
                }
                System.out.println();
            }
        }
    }
}
```

For CSV files containing quoted commas, embedded newlines or complex escaping, use a CSV library rather than split(",").

XML reader using DOM

```
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.*;

public class XmlReader {
    public static void main(String[] args) throws Exception {

        Document doc =
                DocumentBuilderFactory.newInstance()
                .newDocumentBuilder()
                .parse("testdata.xml");

        NodeList users =
                doc.getElementsByTagName("user");

        for (int i = 0; i < users.getLength(); i++) {
            Element user = (Element) users.item(i);
            System.out.println(
                    user.getElementsByTagName("name")
                    .item(0).getTextContent()
            );
        }
    }
}
```

Q96 Handle dropdown in Selenium

For a standard HTML <select>, use Selenium's Select class. For a custom Angular/React/Material dropdown, interact with the visible component and its option elements instead.

```
import org.openqa.selenium.support.ui.Select;

WebElement country =
    driver.findElement(By.id("country"));

Select select = new Select(country);

select.selectByVisibleText("India");
select.selectByValue("IN");
select.selectByIndex(2);
```

Custom dropdown

```
driver.findElement(By.id("countryDropdown")).click();

driver.findElement(
    By.xpath("//mat-option[normalize-space()='India']")
).click();
```

Select works only with real HTML <select> elements.

For custom dropdowns, inspect the DOM and locate the option/listbox elements.

Use explicit waits when options are loaded dynamically.

Q97 What is StaleElementReferenceException?

A StaleElementReferenceException occurs when Selenium has a reference to an element that is no longer attached to the current DOM. This often happens after a page refresh, navigation, AJAX update, Angular re-render or DOM replacement.

Example situation

```
WebElement button =
    driver.findElement(By.id("submit"));

driver.navigate().refresh();

button.click(); // may throw stale element exception
```

How to handle

Locate the element again after the DOM changes.

Use explicit waits for the new/current element.

Avoid storing WebElement references longer than necessary when the DOM is frequently refreshed.

For retry logic, re-find the element rather than repeatedly using the stale object.

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement button = wait.until(
    ExpectedConditions.elementToBeClickable(
        By.id("submit")
    )
);
button.click();
```

Q98 Code to extract HTML source

```
String html = driver.getPageSource();

System.out.println(html);
```

getPageSource() returns the page source available to the WebDriver implementation. If you need the current DOM representation of a particular element, getAttribute("outerHTML") is often more targeted.

```
String elementHtml =
    driver.findElement(By.id("container"))
          .getAttribute("outerHTML");
```

Q99 quit and close – parameters and return value

driver.close(): closes the current browser window/tab controlled by the driver. It takes no parameters and returns void.

driver.quit(): closes all browser windows/tabs opened by that WebDriver session and ends the session. It takes no parameters and returns void.

```
driver.close(); // current window
driver.quit();  // entire WebDriver session
```

Use quit() in teardown to fully release the WebDriver session. If multiple windows exist, close() affects only the currently focused window.

Q100 Difference between implicit wait, WebDriverWait and FluentWait

Type	Scope	Polling	Typical use
Implicit Wait	Global element lookup	Driver-controlled	Basic synchronization for findElement
WebDriverWait	Specific condition	Regular polling	Recommended explicit condition-based waits
FluentWait	Specific object/condition	Configurable	Custom timeout, polling and ignored exceptions

```
driver.manage().timeouts()
      .implicitlyWait(Duration.ofSeconds(5));

WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(
    By.id("message")
));

Wait<WebDriver> fluent = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(20))
    .pollingEvery(Duration.ofSeconds(2))
    .ignoring(NoSuchElementException.class);
```

Do not mix implicit and explicit waits casually because their interactions can make timing behavior less predictable. Prefer explicit waits for specific dynamic conditions.

Q101 Actions class import – Selenium or Java?

Actions is a Selenium class, not a Java AWT class. Import it from Selenium's interaction package.

```
import org.openqa.selenium.interactions.Actions;

Actions actions = new Actions(driver);
actions.moveToElement(element).click().perform();
```

Do not confuse it with java.awt.Robot, which is used for lower-level native keyboard/mouse events.

Q102 Access modifiers and their use

Modifier	Access	Use
private	Same class only	Encapsulation/internal implementation
default (no keyword)	Same package	Package-level access
protected	Same package + subclasses in other packages (through inheritance rules)	Inheritance-oriented access
public	Everywhere subject to normal type/module visibility	Public API

Top-level classes can be public or package-private; private/protected are not permitted for top-level classes.

Q103 How to check whether a radio button is clicked/selected

```
WebElement radio =
        driver.findElement(By.id("male"));

if (!radio.isSelected()) {
    radio.click();
}

System.out.println("Selected: "
        + radio.isSelected());
```

isSelected() works for selectable input elements such as radio buttons and checkboxes. If the control is custom, inspect the DOM for state attributes such as aria-checked or a selected CSS class.

Q104 Find the nearest element to a given element using XPath

XPath axes allow you to navigate from a known element to its nearby parent, sibling or ancestor. There is no universal 'nearest' axis; the correct expression depends on the DOM relationship.

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

// Nearest preceding sibling
//input[@id='email']
    /preceding-sibling::label[1]

// Nearest ancestor container
//input[@id='email']
    /ancestor::div[contains(@class,'form-row')][1]

// First following matching element
//label[normalize-space()='Email']
    /following::input[1]
```

Use [1] carefully: following-sibling::*[1] means the immediate next sibling, while following::input[1] selects the first matching input in document order after the context node.

Q105 How will you trigger an event in Selenium?

The best approach is usually to perform the real user action that causes the event: click, sendKeys, hover, select, etc. JavaScript can be used when the application specifically requires a DOM event or when a targeted workaround is justified.

```
// Real user interaction
driver.findElement(By.id("button")).click();

// JavaScript click
JavascriptExecutor js =
    (JavascriptExecutor) driver;

js.executeScript(
    "arguments[0].click();",
    driver.findElement(By.id("button"))
);
```

Dispatch a DOM event

```
js.executeScript(
    "arguments[0].dispatchEvent(" +
    "new Event('change', {bubbles:true}));",
    element
);
```

Prefer genuine Selenium actions because they more closely represent real user behavior. JavaScript event triggering can bypass browser interaction behavior and should be used only when appropriate.

Q106 Count vowels, digits, consonants and spaces in a string

```
public class CharacterCount {
    public static void main(String[] args) {

        String input = "Java Selenium 123";
        int vowels = 0;
        int consonants = 0;
        int digits = 0;
        int spaces = 0;

        for (char ch : input.toCharArray()) {

            if (Character.isDigit(ch)) {
                digits++;
            } else if (Character.isWhitespace(ch)) {
                spaces++;
            } else if (Character.isLetter(ch)) {
                char c = Character.toLowerCase(ch);

                if (c=='a' || c=='e' || c=='i'
                        || c=='o' || c=='u') {
                    vowels++;
                } else {
                    consonants++;
                }
            }
        }

        System.out.println("Vowels: " + vowels);
        System.out.println("Consonants: " + consonants);
        System.out.println("Digits: " + digits);
        System.out.println("Spaces: " + spaces);
    }
}
```

Q107 Check whether a number is an Armstrong number

An Armstrong number equals the sum of its digits raised to the power of the number of digits. For example, 153 = $1^3 + 5^3 + 3^3$.

```
public class ArmstrongNumber {
    public static void main(String[] args) {

        int number = 153;
        int original = number;
        int digits = String.valueOf(number).length();
        int sum = 0;

        while (number != 0) {
            int digit = number % 10;
```

```
            sum += (int) Math.pow(digit, digits);
            number /= 10;
        }

        if (sum == original) {
            System.out.println("Armstrong number");
        } else {
            System.out.println("Not an Armstrong number");
        }
    }
}
```

Q108 Java code to scroll a web page

Scroll by pixels

```
JavascriptExecutor js =
    (JavascriptExecutor) driver;

js.executeScript("window.scrollBy(0, 500);");
```

Scroll to bottom

```
js.executeScript(
    "window.scrollTo(0, document.body.scrollHeight);"
);
```

Scroll to an element

```
WebElement element =
    driver.findElement(By.id("footer"));

js.executeScript(
    "arguments[0].scrollIntoView({block:'center'});",
    element
);
```

Selenium 4 Actions

```
new Actions(driver)
    .scrollToElement(element)
    .perform();
```

Prefer the Selenium Actions approach when suitable; JavaScript scrolling is useful when you need specific scroll behavior.

Q109 Check whether web pages are loaded or not

For a normal page, use document.readyState. It should become 'complete' after the page's document has finished loading. For modern SPAs, readyState alone may not mean the application is ready; wait for a meaningful application element or API-driven state too.

```
WebDriverWait wait =
    new WebDriverWait(driver, Duration.ofSeconds(30));

wait.until(webDriver ->
    ((JavascriptExecutor) webDriver)
        .executeScript("return document.readyState")
        .equals("complete")
```

```
);

System.out.println("Page load completed");
```

SPA note

```
wait.until(ExpectedConditions.visibilityOfElementLocated(
    By.id("mainContent")
));
```

The strongest approach is often: document ready state + application-specific readiness condition.

Q110 Create a method to get specified tag elements and store them in a List

```
public List<WebElement> getElementsByTag(
        WebDriver driver, String tagName) {

    return driver.findElements(By.tagName(tagName));
}

// Usage
List<WebElement> links =
        getElementsByTag(driver, "a");

System.out.println("Links: " + links.size());
```

CSS alternative

```
List<WebElement> inputs =
    driver.findElements(By.cssSelector("input"));
```

findElements returns a List and returns an empty list when no matching elements are found, unlike findElement which throws NoSuchElementException.

Q111 Upload an image/file when the upload button uses a file input

For a standard HTML input type='file', Selenium can directly set the file path using sendKeys(). This is generally better than automating the native file chooser.

```
String filePath =
    Paths.get("testdata", "image.png")
         .toAbsolutePath()
         .toString();

WebElement upload =
    driver.findElement(
        By.cssSelector("input[type='file']")
    );

upload.sendKeys(filePath);
```

If the UI button opens a native chooser

If there is no accessible file input and the application opens an operating-system file chooser, Selenium alone cannot reliably control the native dialog. Depending on the application, use the underlying input element, application-specific upload API, or a suitable desktop automation mechanism.

Important

Use an absolute path when the working directory may vary.

Verify the file exists before upload.

Do not use Robot if the hidden file input can be accessed directly.

Q112 Find the Nth largest number in an array

A simple interview solution is to sort the array in descending order and return the element at index N-1. If built-in sorting is prohibited, implement a sort or use a heap/selection algorithm.

```
import java.util.Arrays;

public class NthLargest {
    public static void main(String[] args) {

        int[] arr = {10, 5, 20, 8, 30, 15};
        int n = 3;

        Arrays.sort(arr);

        int result = arr[arr.length - n];

        System.out.println(
                n + "rd largest = " + result
        );
    }
}

// Output: 3rd largest = 15
```

Distinct Nth largest

```
int[] arr = {10, 20, 20, 30, 40, 40};
int n = 2;

int result = Arrays.stream(arr)
        .boxed()
        .distinct()
        .sorted(Comparator.reverseOrder())
        .skip(n - 1)
        .findFirst()
        .orElseThrow();

System.out.println(result);
```

Clarify with the interviewer whether duplicates should count. The second example treats only distinct values as candidates.

Q113 Provide a unique email every time to a textbox

A practical approach is to generate an email using a timestamp or UUID. UUID is especially useful when uniqueness across parallel runs is important.

```
String email =
    "test_" + UUID.randomUUID()
                  .toString()
                  .replace("-", "")
    + "@example.com";

driver.findElement(By.id("email"))
      .sendKeys(email);
```

Timestamp approach

```
String email =
    "test_" + System.currentTimeMillis()
    + "@example.com";
```

For parallel tests, UUID is safer than a timestamp alone because multiple threads can execute within the same millisecond.

Q114 Types of reports – Extent Report and Allure Report

Extent Reports

Extent Reports is a reporting library commonly integrated into Selenium/TestNG/Cucumber frameworks. It can show pass/fail/skip status, steps, logs, screenshots and execution details in an HTML report.

```
// Conceptual example
ExtentReports extent = new ExtentReports();
ExtentTest test = extent.createTest("Login Test");

test.pass("Login successful");
test.fail("Error occurred");

extent.flush();
```

Allure Report

Allure is a reporting framework that generates interactive reports from test results. It supports steps, attachments, screenshots, categories, metadata and integration with common test frameworks.

```
// Example with TestNG
import io.qameta.allure.Description;
import org.testng.annotations.Test;

public class LoginTest {

    @Test
    @Description("Validate successful login")
    public void login() {
        // test steps
    }
}
```

Extent vs Allure

Feature	Extent	Allure
Output	Interactive HTML	Interactive web report
Attachments	Screenshots/logs	Screenshots/logs/files
Test framework integration	TestNG/Cucumber etc.	TestNG/Cucumber/JUnit etc.
Strength	Simple customizable HTML reporting	Rich test analysis, history/metadata ecosystem
CI use	Good	Good