SDET Preparation Kit-Frameworks & Automation Scenarios

Maven

1. What is Maven and why is it used in automation frameworks?

Maven is a **build automation and dependency management tool** used mainly for Java projects.
It simplifies project setup, builds, and dependency handling through a configuration file called `pom.xml`.

Key Benefits:

- Centralized dependency management
- Automated build lifecycle (compile → test → package)
- Easy integration with CI/CD tools like Jenkins

2. What is `pom.xml` in Maven?

`pom.xml` (Project Object Model) is the **core configuration file** in Maven.
It defines project dependencies, plugins, and goals.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.qa</groupId>
  <artifactId>AutomationFramework</artifactId>
```

```
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.10.0</version>
    </dependency>
  </dependencies>
</project>
```

3. What is the difference between `build` and `clean` in Maven?

Command	Description
`mvn clean`	Deletes the `target` folder containing compiled files.
`mvn compile`	Compiles the project source code.
`mvn test`	Runs the unit tests.
`mvn package`	Packages the code into a `.jar` or `.war`.
`mvn install`	Installs the packaged file to the local repository.

4. How does Maven manage dependencies?

Maven downloads project dependencies from a **central repository** (or private repository) defined in `pom.xml`.
Dependencies are stored in the **local repository** (`.m2` folder).

```
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.9.0</version>
```

```
</dependency>
```

5. What are Maven repositories and their types?

Maven repositories store project libraries and plugins.

Types:

1. **Local Repository** – On your system (`.m2/repository`).
2. **Central Repository** – Public Maven repository.
3. **Remote Repository** – Company-maintained custom repo (like Nexus/Artifactory).

6. What is a Maven lifecycle?

Maven has **three built-in lifecycles:**

1. **Clean** – Cleans the project (removes target folder).
2. **Default (Build)** – Handles compile, test, package, install, deploy.
3. **Site** – Generates project documentation.

Each lifecycle has multiple **phases** (e.g., `compile`, `test`, `package`, `install`).

7. What is the purpose of Maven plugins?

Plugins add functionality to the Maven build process.
Examples include **compiler**, **surefire**, and **shade** plugins.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.11.0</version>
  <configuration>
    <source>17</source>
    <target>17</target>
  </configuration>
</plugin>
```

8. How do you run tests using Maven?

You can execute test cases integrated with **TestNG or JUnit** using Maven commands.

```
mvn test
```

Example TestNG configuration in `pom.xml`:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
```

```
  <version>3.0.0</version>
  <configuration>
    <suiteXmlFiles>
      <suiteXmlFile>testng.xml</suiteXmlFile>
    </suiteXmlFiles>
  </configuration>
</plugin>
```

9. What is the difference between `install` and `deploy` phase in Maven?

Phase	Description
`install`	Installs the package into the local repository.
`deploy`	Uploads the package to a remote repository for sharing.

10. How is Maven used in a Selenium or Test Automation Framework?

- Used for managing dependencies like Selenium, TestNG, Cucumber.
- Organizes project structure (`src/main/java`, `src/test/java`).
- Integrates with Jenkins for CI/CD.
- Supports parallel test execution using plugins.

Example folder structure:

```
project
 ├── pom.xml
 ├── src
     ├── main/java
     └── test/java
```

TestNG

1. What is TestNG and why is it used in automation testing?

TestNG (Next Generation) is a testing framework inspired by JUnit and NUnit.
It's used to **design structured test cases**, handle **data-driven testing**, and enable **parallel execution**.

Key Advantages:

- Supports annotations like `@Test`, `@BeforeMethod`, `@AfterSuite`.
- Generates detailed HTML reports.
- Integrates easily with Selenium and Maven.

2. What are the main annotations used in TestNG?

Annotation	Description
`@BeforeSuite`	Executes before all tests in the suite.
`@BeforeClass`	Runs once before any method in the class.
`@BeforeMethod`	Runs before each test method.
`@Test`	Defines a test case.
`@AfterMethod`	Runs after each test method.
`@AfterSuite`	Executes after all tests in the suite.

```
@BeforeMethod
public void setup() {
```

```
  driver = new ChromeDriver();
}

@Test
public void verifyTitle() {
  driver.get("https://example.com");
  Assert.assertEquals(driver.getTitle(), "Example
Domain");
}

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

3. What is the difference between `@BeforeTest` and `@BeforeMethod`?

Annotation	Scope
`@BeforeTest`	Runs once before all test methods in a `<test>` tag (in `testng.xml`).
`@BeforeMethod`	Runs before **each** test method.

4. How do you create dependencies between tests in TestNG?

Use `dependsOnMethods` or `dependsOnGroups`.

It ensures one test runs only after another passes.

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

@Test(dependsOnMethods = {"login"})
public void dashboard()
{ System.out.println("Dashboard loaded"); }
```

5. How do you perform parameterization in TestNG?

Pass values from `testng.xml` using the `@Parameters` annotation.

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

<parameter name="browser" value="chrome"/>
```

6. What is DataProvider in TestNG?

`@DataProvider` supplies multiple sets of data to a single test method, enabling **data-driven testing**.

```
@DataProvider(name = "loginData")
public Object[][] getData() {
  return new Object[][] { {"user1", "pass1"},
{"user2", "pass2"} };
}

@Test(dataProvider = "loginData")
public void loginTest(String user, String pass) {
  System.out.println("Login with " + user + " / " +
pass);
}
```

7. How can you group test cases in TestNG?

Use the `groups` attribute in `@Test` and specify them in `testng.xml`.

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

@Test(groups = {"regression"})
public void regressionTest()
{ System.out.println("Regression Test"); }

<groups>
  <run>
    <include name="smoke"/>
```

```
  </run>
</groups>
```

8. How do you prioritize tests in TestNG?

Use the `priority` attribute.

Lower priority value executes first.

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

@Test(priority = 2)
public void search() {}

@Test(priority = 3)
public void logout() {}
```

9. How to skip a test in TestNG?

Use `enabled = false` or throw `SkipException`.

```
@Test(enabled = false)
public void disabledTest() {}

@Test
public void skipTest() {
  throw new SkipException("Skipping this test
```

```
intentionally");
}
```

10. How do you generate reports in TestNG?

After execution, TestNG automatically creates reports inside the `test-output` folder.
You can also integrate with **ExtentReports** or **Allure** for detailed HTML reports.

```
ExtentReports extent = new ExtentReports();
ExtentTest test = extent.createTest("Login Test");
test.pass("Login passed successfully");
extent.flush();
```

Cucumber BDD

1. What is Cucumber and why is it used in automation?

Cucumber is a **Behavior-Driven Development (BDD)** tool used to write tests in a **human-readable format**.
It bridges the communication gap between technical and non-technical stakeholders.

Key Benefits:

- Uses Gherkin language (`Given`, `When`, `Then`).
- Integrates with Selenium and TestNG/JUnit.
- Promotes collaboration and clarity.

2. What is Gherkin in Cucumber?

Gherkin is a **domain-specific language** that describes test scenarios in plain English.
It uses keywords such as `Feature`, `Scenario`, `Given`, `When`, `Then`.

```
Feature: Login Functionality
  Scenario: Valid login
    Given user is on the login page
    When user enters valid credentials
```

```
    Then user should see the homepage
```

3. What are the main components of a Cucumber framework?

Component	Description
Feature File	Contains test cases written in Gherkin syntax.
Step Definition	Contains Java code mapped to each Gherkin step.
Runner Class	Executes feature files using Cucumber options.

4. How do you create Step Definitions in Cucumber?

Step definitions link Gherkin steps to Java code.

```
@Given("user is on the login page")
public void user_on_login_page() {
  driver.get("https://example.com/login");
}

@When("user enters valid credentials")
public void enter_credentials() {

driver.findElement(By.id("username")).sendKeys("admin
");

driver.findElement(By.id("password")).sendKeys("12345
");
}

@Then("user should see the homepage")
```

```
public void verify_homepage() {
  Assert.assertEquals(driver.getTitle(),
"Dashboard");
}
```

5. What is the purpose of the Cucumber Runner Class?

The **Runner class** triggers execution of the feature files. It uses `@CucumberOptions` to define paths, plugins, and reporting.

```
@RunWith(Cucumber.class)
@CucumberOptions(
  features = "src/test/resources/features",
  glue = "stepDefinitions",
  plugin = {"pretty", "html:target/cucumber-reports"}
)
public class TestRunner {}
```

6. How do you share data between steps in Cucumber?

Use **Dependency Injection** (via `ScenarioContext` or `PicoContainer`) or define variables globally.

```
public class SharedData {
  public static String username;
```

```
}

@When("user enters username {string}")
public void enterUsername(String user) {
  SharedData.username = user;
}
```

7. How can you run specific scenarios or feature files?

Use **tags** to control test execution.

```
@Smoke
Scenario: Verify login
  Given user is on the login page
  When user enters credentials
  Then homepage is displayed
```

Runner configuration:

```
@CucumberOptions(tags = "@Smoke")
```

8. How do you handle Data Tables in Cucumber?

Data tables help pass structured data from Gherkin to step definitions.

```
Scenario: Login with multiple users
  When user enters credentials
    | username | password |
    | user1    | pass1    |
    | user2    | pass2    |

@When("user enters credentials")
public void enterCredentials(DataTable table) {
  List<Map<String, String>> data = table.asMaps();
  for (Map<String, String> row : data) {
    System.out.println(row.get("username") + " / " +
row.get("password"));
  }
}
```

9. What are hooks in Cucumber?

Hooks are used to execute code **before or after each scenario**.

Examples: setup, teardown, or screenshot capture.

```
@Before
public void setup() { driver = new ChromeDriver(); }

@After
public void teardown() { driver.quit(); }
```

10. How do you integrate Cucumber with Selenium and TestNG?

Cucumber integrates seamlessly with Selenium for automation and TestNG for reporting.

You define test logic in step definitions, run through the Cucumber runner, and use Maven for dependencies.

Example Maven dependency:

```
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-java</artifactId>
  <version>7.14.0</version>
</dependency>
```

Page Object Model (POM)

1. What is the Page Object Model (POM) in Selenium?

Page Object Model is a **design pattern** used in test automation to create an **object repository for web elements**. Each web page of the application is represented by a separate class that contains:

- Locators for elements.
- Methods to perform actions on those elements.

This helps in improving **code reusability**, **readability**, and **maintainability**.

2. What are the main advantages of using POM?

- Reduces code duplication.
- Enhances readability and maintainability.
- Centralizes all web elements.
- Separates **test logic** from **UI logic**.

3. How do you implement the Page Object Model in Selenium?

Create separate classes for each web page and define locators and actions inside.

```
public class LoginPage {
  WebDriver driver;

  By username = By.id("username");
  By password = By.id("password");
  By loginBtn = By.id("login");

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

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

4. How do you call methods from the Page Class in the Test Class?

Create an object of the page class and call its methods from your test.

```
@Test
public void testLogin() {
  WebDriver driver = new ChromeDriver();
  driver.get("https://example.com");
  LoginPage page = new LoginPage(driver);
  page.login("admin", "password123");
}
```

5. What is the Page Factory in Selenium?

Page Factory is an extension of POM that uses annotations like `@FindBy` to initialize elements.

It helps in writing **cleaner and faster code**.

```
public class LoginPage {
  @FindBy(id="username") WebElement username;
  @FindBy(id="password") WebElement password;
  @FindBy(id="login") WebElement loginBtn;

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

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

```
}
```

6. What is the difference between POM and Page Factory?

Feature	POM	Page Factory
Element initialization	Manual (By locators)	Automated (`@FindBy`)
Code readability	Slightly longer	Cleaner
Recommended	For beginners	For advanced structured frameworks

7. How does POM support test maintenance?

When the UI changes, only the corresponding **page class** needs updating — not the entire test script.
This isolates changes and simplifies maintenance.

8. Can you use POM with other frameworks like TestNG or Cucumber?

Yes, POM is framework-independent.
It can be integrated with TestNG, Cucumber, or JUnit for execution and reporting.

Example with TestNG:

```
@Test
public void loginTest() {
```

```
  LoginPage login = new LoginPage(driver);
  login.login("admin", "admin123");
}
```

9. How do you handle multiple pages using POM?

Create a class for each page and use method chaining or class references to move between them.

```
HomePage home = loginPage.login("admin", "1234");
ProfilePage profile = home.goToProfile();
profile.updateName("John");
```

10. How do you organize a framework using POM structure?

A standard POM-based Selenium framework follows this structure:

```
project
 ├── src/test/java
 │    ├── pages
 │    │    ├── LoginPage.java
 │    │    ├── HomePage.java
 │    ├── tests
 │    │    ├── LoginTest.java
 │    ├── utilities
```

```
│   │   ├── ConfigReader.java
│   │   ├── DriverFactory.java
├── pom.xml
```

Real-Time Selenium Coding

1. How to launch a browser and open a URL in Selenium?

Use the `WebDriver` interface and specific browser drivers.

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

2. How to handle multiple browser windows or tabs?

Use `getWindowHandles()` and switch between window IDs.

```
String parent = driver.getWindowHandle();
Set<String> allWindows = driver.getWindowHandles();

for (String win : allWindows) {
  if (!win.equals(parent)) {
    driver.switchTo().window(win);
    System.out.println("Child Window Title: " +
driver.getTitle());
  }
}
driver.switchTo().window(parent);
```

3. How to handle dropdown menus in Selenium?

Use the `Select` class for HTML `<select>` dropdowns.

```
WebElement dropdown =
driver.findElement(By.id("country"));
Select select = new Select(dropdown);
select.selectByVisibleText("India");
```

4. How to handle alerts and pop-ups?

Use `driver.switchTo().alert()` to accept or dismiss alerts.

```
Alert alert = driver.switchTo().alert();
System.out.println(alert.getText());
alert.accept();  // Click OK
```

5. How to perform mouse hover or drag-and-drop?

Use the `Actions` class to simulate advanced user interactions.

```
Actions action = new Actions(driver);
WebElement element =
driver.findElement(By.id("menu"));
```

```
action.moveToElement(element).perform();

WebElement src = driver.findElement(By.id("source"));
WebElement dest =
driver.findElement(By.id("target"));
action.dragAndDrop(src, dest).perform();
```

6. How to handle frames and iframes?

Switch to frames by index, name, or WebElement.

```
driver.switchTo().frame("frameName");
driver.findElement(By.id("submit")).click();
driver.switchTo().defaultContent();
```

7. How to take screenshots in Selenium?

Use `TakesScreenshot` interface and store as a file.

```
File src =
((TakesScreenshot)driver).getScreenshotAs(OutputType.
FILE);
FileUtils.copyFile(src, new
File("./screenshots/page.png"));
```

8. How to wait for elements dynamically?

Use **Explicit Wait** with `WebDriverWait`.

```
WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLoca
ted(By.id("username")));
element.sendKeys("admin");
```

9. How to run tests in headless mode?

Use ChromeOptions or FirefoxOptions for headless execution.

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless");
WebDriver driver = new ChromeDriver(options);
driver.get("https://example.com");
System.out.println(driver.getTitle());
```

10. How to read test data from Excel in Selenium Framework?

Use **Apache POI** to handle Excel files.

```
FileInputStream file = new
FileInputStream("data.xlsx");
XSSFWorkbook workbook = new XSSFWorkbook(file);
XSSFSheet sheet = workbook.getSheet("Login");
String username =
sheet.getRow(1).getCell(0).getStringCellValue();
System.out.println("Username: " + username);
workbook.close();
```