SDET Preparation Kit – Java & Selenium Foundations

Manual Testing

1. What is the difference between Verification and Validation?

- **Verification** checks whether we are building the product **right** (process-based).
- **Validation** checks whether we are building the **right product** (output-based).

Verification → Reviewing documents/designs.
Validation → Executing test cases.

Example:

- Verification: Requirement or design review
- Validation: Functional testing, UAT

2. What is the difference between Regression and Retesting?

- **Regression Testing** ensures that new changes didn't break existing features.
- **Retesting** verifies whether specific defects are fixed.

Regression → Broad scope, can be automated.
Retesting → Narrow scope, usually manual.

3. What is Severity and Priority?

- **Severity:** How serious the defect's impact is.
- **Priority:** How soon it needs to be fixed.

Examples:

- High Severity + High Priority → Login button not working
- Low Severity + Medium Priority → Typo on homepage

4. What is a Test Case? What are its key components?

A **test case** defines how to verify a specific functionality.

Key Components:

- Test Case ID
- Test Scenario
- Preconditions
- Test Steps
- Expected Result
- Actual Result
- Status

Example:

```
TC001: Verify login with valid credentials
Steps:
1. Launch the app
2. Enter valid username and password
3. Click Login
Expected Result: User should navigate to Dashboard
```

5. What is a Test Plan and what does it include?

A **Test Plan** defines the scope, approach, resources, and schedule of testing activities.

Includes:

- Objectives
- Scope
- Test Strategy
- Test Environment Setup
- Entry/Exit Criteria
- Deliverables

- Risk & Mitigation Plan

6. What is the Defect Life Cycle (Bug Life Cycle)?

The **Defect Life Cycle** represents all stages a defect passes through:

```
New → Assigned → Open → Fixed → Retest → Verified → Closed
          ↑                         ↓
       Reopen ← Failed Verification
```

7. What are Entry and Exit Criteria?

- **Entry Criteria:** Conditions to start testing (e.g., environment ready, test data available).
- **Exit Criteria:** Conditions to stop testing (e.g., all critical defects fixed, 95% test cases passed).

8. How do you ensure 100% test coverage?

- Use a **Requirement Traceability Matrix (RTM)** to link test cases with requirements.
- Include positive, negative, and boundary value test cases.
- Perform peer reviews.

Example RTM:

```
Requirement ID | Test Case ID
REQ-101        | TC001, TC002
REQ-102        | TC003
```

9. What is Risk-Based Testing?

Testing focused on **high-risk or business-critical areas first.**
This ensures limited time is used efficiently.

Example:
In a banking app, test *Funds Transfer* before *Profile Picture Upload*.

10. What would you do if you find a bug in production?

1. Reproduce and collect evidence (screenshots, logs).
2. Log the bug with clear steps and severity.
3. Notify stakeholders immediately.
4. After fix, perform **Root Cause Analysis (RCA)**.

Bug Report Example:

```
Title: Amount not deducted during transfer
Severity: Critical
Environment: Production
Steps to Reproduce:
1. Initiate transfer of ₹1000
2. Amount not deducted, but confirmation shown
```

Java

1. What is the difference between JDK, JRE, and JVM?

- **JDK (Java Development Kit):** Used to develop Java applications (includes compiler `javac`, debugger, JRE).
- **JRE (Java Runtime Environment):** Used to run Java applications (includes JVM + libraries).
- **JVM (Java Virtual Machine):** Executes Java bytecode and makes Java platform-independent.

Flow:

```
Java Source Code (.java) → Compiler → Bytecode (.class) → JVM →
Machine Code
```

2. What are the OOP Principles in Java?

The **four pillars of OOP** are:

1. **Encapsulation:** Binding data and methods.
2. **Abstraction:** Hiding implementation details.
3. **Inheritance:** Reusing code from parent class.
4. **Polymorphism:** One method behaves differently based on context.

Example:

```
class Animal {
  void sound() { System.out.println("Animal makes sound"); }
}
```

https://www.linkedin.com/in/vishnupriyaravichandran

```
class Dog extends Animal {
  void sound() { System.out.println("Dog barks"); }
}

public class Main {
  public static void main(String[] args) {
    Animal obj = new Dog();  // Polymorphism
    obj.sound();
  }
}
```

3. What is the difference between == and `.equals()` in Java?

- == compares **references (memory addresses)**.
- `.equals()` compares **values/content**.

Example:

```
String s1 = new String("Test");
String s2 = new String("Test");
System.out.println(s1 == s2);      // false (different memory)
System.out.println(s1.equals(s2)); // true (same content)
```

4. Difference between ArrayList and LinkedList

Feature	ArrayList	LinkedList
Storage	Dynamic array	Doubly linked list
Access speed	Faster (O(1))	Slower (O(n))
Insertion/Deletion	Slower	Faster
Use case	Read-heavy	Insert/delete-heavy

Example:

```
List<String> list = new ArrayList<>();
list.add("A"); list.add("B");
System.out.println(list.get(1)); // Output: B
```

5. Difference between HashMap, LinkedHashMap, and TreeMap

Type	Ordering	Performance
`HashMap`	Unordered	Fastest
`LinkedHashMap`	Maintains insertion order	Slightly slower
`TreeMap`	Sorted by keys	Slowest

Example:

```
Map<Integer, String> map = new TreeMap<>();
map.put(2, "B");
map.put(1, "A");
System.out.println(map); // Output: {1=A, 2=B}
```

6. Difference between Abstract Class and Interface

Feature	Abstract Class	Interface
Methods	Can have abstract & concrete	Only abstract (Java 7), default/static allowed (Java 8+)
Variables	Can have instance variables	Only constants (public static final)
Inheritance	Single	Multiple

Example:

```
abstract class Car {
  abstract void drive();
}

interface MusicSystem {
  void playMusic();
}
```

```
class Tesla extends Car implements MusicSystem {
  void drive() { System.out.println("Self-driving"); }
  public void playMusic() { System.out.println("Playing songs"); }
}
```

7. Difference between final, finally, and finalize()

Keyword	Purpose
`final`	Used for constants, prevents inheritance or overriding
`finally`	Used in exception handling, executes always
`finalize()`	Called by garbage collector before object destruction

Example:

```
try {
  int x = 10 / 0;
} catch(Exception e) {
  System.out.println("Error");
} finally {
  System.out.println("Cleanup code");
}
```

8. Explain Exception Handling in Java

Handled using `try`, `catch`, `finally`, `throw`, `throws`.

Example:

```
try {
  int a = 10 / 0;
} catch (ArithmeticException e) {
  System.out.println("Cannot divide by zero");
} finally {
  System.out.println("Finally block executes always");
}
```

9. What is Multithreading in Java?

- Allows concurrent execution of two or more threads.
- Increases performance in multi-core systems.

Example:

```
class MyThread extends Thread {
  public void run() {
    System.out.println("Thread running: " +
Thread.currentThread().getName());
  }
}

public class Demo {
  public static void main(String[] args) {
    MyThread t1 = new MyThread();
    t1.start();
  }
}
```

10. How is Java used in Selenium or Automation Frameworks?

- Used for **writing test scripts** and **framework logic** (TestNG, POM).
- Supports **OOP concepts** for modular test design.
- Integrates with **Maven**, **Jenkins**, and **Cucumber**.

Example:

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

Core Java OOPs Concepts

1. What is Object-Oriented Programming (OOP)?

OOP is a programming paradigm that organizes code into **objects** — reusable units that combine data (fields) and behavior (methods).
It improves modularity, scalability, and reusability.

Example:

```
class Car {
  String color = "Red";
  void drive() { System.out.println("Driving..."); }
}

public class Main {
  public static void main(String[] args) {
    Car car = new Car();
    car.drive();
  }
}
```

2. What are the four main OOP principles in Java?

1. **Encapsulation** – Wrapping data & methods together.

2. **Abstraction** – Hiding implementation details.
3. **Inheritance** – Reusing properties of parent class.
4. **Polymorphism** – One method, multiple behaviors.

Example:

```
class Shape {
  void draw() { System.out.println("Drawing
shape"); }
}
class Circle extends Shape {
  void draw() { System.out.println("Drawing
circle"); }
}
```

3. What is the difference between Abstraction and Encapsulation?

Feature	Abstraction	Encapsulation
Focus	Hides complexity	Protects data
Achieved by	Abstract classes & interfaces	Access modifiers (private, public)
Example	`abstract void run()`	Private fields with getters/setters

Example:

```
class Employee {
  private int salary = 50000; // Encapsulation
  public int getSalary() { return salary; }
}
```

4. What is Inheritance in Java?

Inheritance allows a class to **acquire properties and behavior** of another class using the `extends` keyword.

Example:

```
class Parent {
  void greet() { System.out.println("Hello from
Parent"); }
}

class Child extends Parent {
  void message() { System.out.println("Hello from
Child"); }
}

public class Test {
  public static void main(String[] args) {
    Child obj = new Child();
    obj.greet(); // Access from parent
    obj.message();
  }
}
```

5. What is Polymorphism?

Polymorphism means *"many forms"*. The same method behaves differently depending on the context.

Types:

- Compile-time (Method Overloading)
- Runtime (Method Overriding)

Example:

```
class MathOps {
  int add(int a, int b) { return a + b; }
  double add(double a, double b) { return a + b; } //
Overloading
}
```

6. What is Method Overloading and Method Overriding?

Type	Definition	Occurs In
Overloading	Same method name, different parameters	Same class
Overriding	Same method name & parameters	Parent-Child classes

Example:

```
class Animal {
  void sound() { System.out.println("Generic
sound"); }
}
class Dog extends Animal {
  void sound() { System.out.println("Bark"); } //
Overriding
}
```

7. Can you override static or private methods?

No

- **Static methods** belong to the class, not instance.
- **Private methods** are not accessible outside the class.

Example:

```
class A {
  static void display() { System.out.println("A"); }
}
class B extends A {
  static void display() { System.out.println("B"); }
// Method hiding, not overriding
}
```

8. What is the role of `this` and `super` keywords?

Keyword	Purpose

`this`	Refers to current class instance
`super`	Refers to parent class instance

Example:

```
class Parent {
  String name = "Parent";
}

class Child extends Parent {
  String name = "Child";
  void show() {
    System.out.println(this.name);  // Child
    System.out.println(super.name); // Parent
  }
}
```

9. What is an Interface and how is it different from an Abstract Class?

Feature	Interface	Abstract Class
Type	Contract	Partial implementation
Methods	Abstract, default, static	Abstract + concrete
Variables	public static final	Instance + static

Example:

```
interface Payment {
  void pay();
}

abstract class CardPayment {
  abstract void validate();
}

class CreditCard extends CardPayment implements
Payment {
  void validate() { System.out.println("Card
validated"); }
  public void pay() { System.out.println("Payment
done"); }
}
```

10. What is a Design Pattern and how does it relate to OOP?

Design Patterns are **reusable solutions** to common design problems in software development.
They are built upon **OOP principles** like abstraction, inheritance, and encapsulation.

Example – Singleton Pattern:

```
class Singleton {
  private static Singleton instance;
  private Singleton() {}
  public static Singleton getInstance() {
    if (instance == null)
      instance = new Singleton();
    return instance;
  }
}
```

Java Coding for SDET

1. Reverse a String without using reverse() method

Reverse a string using a loop or `StringBuilder`.

```
public class ReverseString {
  public static void main(String[] args) {
    String str = "Automation";
    String reversed = "";
    for (int i = str.length() - 1; i >= 0; i--) {
      reversed += str.charAt(i);
    }
    System.out.println("Reversed: " + reversed);
  }
}
```

2. Check if a String is a Palindrome

A palindrome reads the same backward and forward.

```
public class PalindromeCheck {
  public static void main(String[] args) {
    String str = "madam";
    String rev = new
StringBuilder(str).reverse().toString();
    System.out.println(str.equals(rev) ?
"Palindrome" : "Not Palindrome");
  }
}
```

3. Find duplicate characters in a String

Use a HashMap to count character occurrences.

```
import java.util.*;

public class DuplicateChars {
  public static void main(String[] args) {
    String str = "testing";
    Map<Character, Integer> map = new HashMap<>();
    for (char ch : str.toCharArray()) {
      map.put(ch, map.getOrDefault(ch, 0) + 1);
    }
    for (Map.Entry<Character, Integer> e :
map.entrySet()) {
      if (e.getValue() > 1)
        System.out.println(e.getKey() + " = " +
e.getValue());
    }
  }
}
```

4. Count occurrences of each character in a String

Similar to finding duplicates, but display all counts.

```
import java.util.*;

public class CharCount {
  public static void main(String[] args) {
    String str = "java";
    Map<Character, Integer> countMap = new
HashMap<>();
    for (char c : str.toCharArray())
      countMap.put(c, countMap.getOrDefault(c, 0) +
1);
    System.out.println(countMap);
  }
}
```

5. Check if two Strings are Anagrams

Sort both strings and compare.

```
import java.util.Arrays;

public class AnagramCheck {
  public static void main(String[] args) {
    String s1 = "listen";
    String s2 = "silent";
    char[] a = s1.toCharArray();
    char[] b = s2.toCharArray();
    Arrays.sort(a);
    Arrays.sort(b);
    System.out.println(Arrays.equals(a, b) ?
"Anagram" : "Not Anagram");
  }
}
```

6. Find the first non-repeated character in a String

Use a `LinkedHashMap` to preserve insertion order.

```
import java.util.*;

public class FirstUniqueChar {
  public static void main(String[] args) {
    String str = "swiss";
    Map<Character, Integer> map = new
LinkedHashMap<>();
```

```
    for (char c : str.toCharArray())
      map.put(c, map.getOrDefault(c, 0) + 1);
    for (Map.Entry<Character, Integer> e :
map.entrySet()) {
      if (e.getValue() == 1) {
        System.out.println("First non-repeated: " +
e.getKey());
        break;
      }
    }
  }
}
```

7. Reverse each word in a sentence

Split by spaces, reverse each word individually.

```
public class ReverseWords {
  public static void main(String[] args) {
    String sentence = "Java Testing Framework";
    String[] words = sentence.split(" ");
    for (String w : words) {
      String rev = new
StringBuilder(w).reverse().toString();
      System.out.print(rev + " ");
    }
  }
}
```

8. Find the largest and smallest number in an array

Use `Math.max()` or manual comparison.

```
public class MinMaxArray {
  public static void main(String[] args) {
    int[] arr = {5, 9, 2, 10, 3};
    int min = arr[0], max = arr[0];
    for (int n : arr) {
      if (n < min) min = n;
      if (n > max) max = n;
    }
    System.out.println("Min: " + min + ", Max: " +
max);
  }
}
```

9. Find the second highest number in an array

Sort and pick the second last element.

```
import java.util.Arrays;

public class SecondHighest {
  public static void main(String[] args) {
    int[] arr = {10, 4, 7, 9, 3};
    Arrays.sort(arr);
    System.out.println("Second highest: " +
arr[arr.length - 2]);
  }
}
```

10. Use Java 8 Stream to filter even numbers from a list

Use `filter()` and `forEach()` with lambda expressions.

```
import java.util.*;
import java.util.stream.*;

public class EvenNumbers {
  public static void main(String[] args) {
    List<Integer> nums = Arrays.asList(1, 2, 3, 4, 5,
6);
    nums.stream()
        .filter(n -> n % 2 == 0)
        .forEach(System.out::println);
  }
}
```

Selenium WebDriver

1. What is Selenium and what are its components?

Selenium is an open-source automation tool for web applications.
It supports multiple browsers and programming languages.

Main Components:

- **Selenium IDE** – Record and playback tool.
- **Selenium WebDriver** – Executes test scripts across browsers.
- **Selenium Grid** – Runs tests in parallel on multiple machines.
- **Selenium RC** – Older version, now deprecated.

2. What is the difference between findElement() and findElements()?

- findElement() returns the **first matching WebElement.**

- findElements() returns a **list of all matching elements.**

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

3. What are different types of waits in Selenium WebDriver?

1. **Implicit Wait:** Applies globally for all elements.
2. **Explicit Wait:** Waits for a specific condition.
3. **Fluent Wait:** Defines polling frequency and ignores exceptions.

```
driver.manage().timeouts().implicitlyWait(Duration.of
Seconds(10));
```

```
WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(15));
wait.until(ExpectedConditions.visibilityOfElementLoca
ted(By.id("login")));
```

4. How do you handle dynamic web elements in Selenium?

Use **XPath with contains(), starts-with()**, or **CSS selectors**.

```
driver.findElement(By.xpath("//input[contains(@id,'user')]")).sendKeys("admin");
driver.findElement(By.cssSelector("button[class*='submit']")).click();
```

5. How do you handle dropdowns in Selenium?

Use the **Select** class for <select> elements.

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

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

6. How do you handle alerts and pop-ups in Selenium WebDriver?

Switch to the alert using driver.switchTo().alert().

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

7. What is Page Object Model (POM)?

POM is a **design pattern** where each page of the application is represented by a separate class.
It improves code **reusability** and **maintenance**.

```
public class LoginPage {
  WebDriver driver;
  By user = By.id("username");
  By pass = By.id("password");
  By loginBtn = By.id("login");

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

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

8. How do you handle multiple browser windows or tabs?

Use getWindowHandles() to switch between windows.

```
String parent = driver.getWindowHandle();
Set<String> allWindows = driver.getWindowHandles();
for (String win : allWindows) {
  if (!win.equals(parent)) {
    driver.switchTo().window(win);
  }
}
```

9. What is StaleElementReferenceException and how do you handle it?

Occurs when the referenced WebElement is no longer attached to the DOM.
To handle it:

- Re-locate the element.
- Use **Explicit Wait** or retry logic.

```
WebElement btn = driver.findElement(By.id("submit"));
driver.navigate().refresh();
btn = driver.findElement(By.id("submit"));
btn.click();
```

10. How do you generate reports in Selenium (ExtentReports/Allure)?

Use **ExtentReports** for detailed HTML reports.

```
ExtentReports report = new ExtentReports();
ExtentSparkReporter spark = new
ExtentSparkReporter("report.html");
report.attachReporter(spark);

ExtentTest test = report.createTest("Login Test");
test.pass("Login successful");
report.flush();
```