Fun Doo TESTERS

Interview Guide

Test Automation

vol.1

Questions & Answers

FUN DOO TESTERS

1 Write a Java program for the largest number from three numbers

```java
import java.util.Scanner;

public class LargestNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter first number:");
        int num1 = scanner.nextInt();

        System.out.println("Enter second number:");
        int num2 = scanner.nextInt();

        System.out.println("Enter third number:");
        int num3 = scanner.nextInt();

        int largest;

        if (num1 ⩾ num2 && num1 ⩾ num3) {
            largest = num1;
        } else if (num2 ⩾ num1 && num2 ⩾ num3) {
            largest = num2;
        } else {
            largest = num3;
        }

        System.out.println("The largest number is: " + largest);
    }
}
```

2. What is SDLC and STLC? And Explain its phases.

SDLC (Software Development Life Cycle) is a process used by the software industry to design, develop, and test high-quality software. The phases include:

- Planning: Define objectives, feasibility analysis.
- Requirement Analysis: Gather and analyze requirements.
- Design: System and software design, creating architecture.
- Implementation (Coding): Writing code based on designs.
- Testing: Testing the software for defects.
- Deployment: Releasing the software to production.
- Maintenance: Updating and maintaining the software post-deployment.

STLC (Software Testing Life Cycle) is a sequence of different activities performed during the software testing process. The phases include:

- Requirement Analysis: Understand what needs to be tested.
- Test Planning: Define the strategy and plan for testing.
- Test Case Development: Write test cases and scripts.
- Environment Setup: Prepare the hardware and software for testing.
- Test Execution: Execute test cases.
- Test Closure: Conclude testing, create reports.

3. What is regression testing?

Regression Testing is a type of software testing that ensures that recent code changes have not adversely affected existing functionalities. It is performed by re-running previously completed tests and checking whether the previously fixed defects have re-emerged.

4. What are different methodologies of SDLC? Explain each.

Common SDLC methodologies include:

- Waterfall Model: Sequential design process, often used in software development processes, where progress is seen as flowing steadily downwards through phases.
- V-Model (Validation and Verification Model): An extension of the waterfall model that includes testing activities at each stage of development.
- Iterative Model: Development starts with a small set of requirements and iteratively enhances the evolving versions until the complete system is implemented.
- Spiral Model: Combines the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- Agile Model: Promotes iterative development and collaboration, with requirements and solutions evolving through the collaborative effort of cross-functional teams.
- DevOps: Aims to shorten the systems development life cycle and provide continuous delivery with high software quality.

5. Define Agile?

Agile is a software development methodology that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Agile methodology emphasizes flexibility, collaboration, customer feedback, and small, rapid releases.

6. Define Scrum and Sprint?

- Scrum: A framework within Agile methodology for managing work with an emphasis on software development. It is characterized by regular, time-boxed iterations called Sprints.
- Sprint: A set period during which specific work has to be completed and made ready for review. Sprints are the basic unit of development in Scrum, lasting usually 2-4 weeks.

7. What is the estimation in Sprint?

Estimation in Sprint refers to predicting the amount of effort required to complete the tasks in a sprint. Techniques like Planning Poker, T-shirt Sizing, and Story Points are commonly used to estimate tasks.

8. What is sprint backlog?

Sprint Backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint. It is a subset of the product backlog and represents the work the team commits to complete in the current sprint.

9. What are the different reports in Testing?

Common reports in testing include:

- Test Case Report: Details about test cases, their execution, and results.
- Defect Report: Information about defects found during testing.
- Test Execution Report: Summary of test cases executed, passed, and failed.
- Test Summary Report: An overall summary of testing activities and results.
- Traceability Matrix: Maps requirements to test cases to ensure coverage.

10. What are the key components of the Test Case report?

Key components of a Test Case report include:

- Test Case ID
- Test Description
- Preconditions
- Test Steps
- Expected Results
- Actual Results
- Status (Pass/Fail)
- Comments

11. What are the components of a defect report?

Components of a defect report include:

- Defect ID
- Summary
- Description
- Steps to Reproduce
- Expected Result
- Actual Result
- Severity
- Priority
- Status
- Reported By
- Assigned To
- Environment
- Attachments (if any)

12. What is Jira?

Jira is a proprietary issue tracking product developed by Atlassian that allows bug tracking and agile project management. It is widely used for issue tracking, bug tracking, and project management.

13. What is a sprint?

A Sprint is a fixed-length iteration in the Scrum framework, typically lasting 2-4 weeks, during which the Scrum team works to complete a set of pre-defined tasks from the sprint backlog.

14. Define black box and white box testing.

- Black Box Testing: Testing technique where the tester doesn't have knowledge of the internal workings of the application. Tests are based on requirements and functionality.
- White Box Testing: Testing technique where the tester has knowledge of the internal workings of the application. Tests are based on code structure, internal logic, and paths.

15. Define functional testing.

Functional Testing is a type of testing that validates the software system against the functional requirements/specifications. It involves testing user interactions and ensuring the software behaves as expected.

16. Define the OOP concept in Java.

- OOP (Object-Oriented Programming) in Java includes:
- Encapsulation: Wrapping data and code together into a single unit, e.g., a class.
- Inheritance: Mechanism where one class inherits the properties and behavior of another class.
- Polymorphism: Ability of a variable, function, or object to take on multiple forms.
- Abstraction: Hiding the complex implementation details and showing only the necessary features of the object.

17. Give me examples of OOPs which you used in your framework.

In a testing framework, examples of OOP usage include:

- Encapsulation: Using classes to encapsulate test data and methods.
- Inheritance: Creating base test classes with common setup/teardown methods that other test classes inherit.
- Polymorphism: Using interfaces or abstract classes for test components.
- Abstraction: Abstracting complex operations into reusable methods.

18. What is TestNG?

TestNG is a testing framework inspired by JUnit and NUnit, designed to cover all categories of tests: unit, functional, end-to-end, integration, etc. It introduces some new functionalities that make it more powerful and easier to use.

19. What is usability testing?

Usability Testing is a technique used to evaluate how easy and user-friendly a software application is. It involves observing real users as they attempt to complete tasks on the software.

20. How will you handle the dropdown in Selenium?

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

WebElement dropdownElement = driver.findElement(By.id("dropdownId"));
Select dropdown = new Select(dropdownElement);
dropdown.selectByVisibleText("OptionText");
```

21. Different types of wait in Selenium? Explain each of them.

```
//Implicit Wait: Sets a default wait time for the entire session until
//the element is found.

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);


//Explicit Wait: Waits for a specific condition to be met before continuing.

WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementId")));

//Fluent Wait: Waits for a specific condition, but also specifies the
//frequency with which to check the condition.

Wait<WebDriver> wait = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(10))
    .pollingEvery(Duration.ofSeconds(2))
    .ignoring(NoSuchElementException.class);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementId")));
```

22. Difference between hard and soft assertion?

```
//Hard Assertion: Fails the test immediately if the assertion fails.
Assert.assertEquals(actual, expected);


//Soft Assertion: Collects errors during assertions
//without throwing exceptions immediately, allowing the test to continue.
SoftAssert softAssert = new SoftAssert();
softAssert.assertEquals(actual, expected);
softAssert.assertAll(); // Collects all assertions
```

23. Why are we using "WebDriver driver = new ChromeDriver()"? Why can't we write RemoteDriver driver = new ChromeDriver()?

WebDriver is an interface that ChromeDriver implements, which means WebDriver driver = new ChromeDriver() uses polymorphism, allowing for flexibility and easier maintenance. RemoteWebDriver is a class, and directly referencing it would limit the ability to switch to other browser drivers.

24. Explain the different annotations in TestNG?

TestNG annotations include:

- @Test: Marks a method as a test method.
- @BeforeMethod: Executes before each test method.
- @AfterMethod: Executes after each test method.
- @BeforeClass: Executes before the first method of the current class.
- @AfterClass: Executes after all the methods of the current class.
- @BeforeSuite: Executes before the suite starts.
- @AfterSuite: Executes after the suite ends.

25. Define Priority and Severity of the Bug?

- Priority: Indicates the urgency and order in which a bug should be fixed.
- Severity: Indicates the impact of the bug on the application's functionality.

26. What are differences between GET and POST request? Explain in detail

GET and POST are two of the most commonly used HTTP methods for sending requests to a server. They serve different purposes and have distinct characteristics.

1. Purpose and Usage

- GET Request:
  - Purpose: Retrieve data from the server.
  - Usage: When you want to request data without modifying any resources. Commonly used for fetching web pages, images, or search queries.
- POST Request:
  - Purpose: Send data to the server to create or update resources.
  - Usage: When you need to submit data, such as form submissions, file uploads, or posting comments.

2. Data Transmission

- GET Request:
  - Data in URL: Parameters are appended to the URL as query strings, which are visible in the URL.
  - Example: http://example.com/page?name=John&age=30
  - Limitations: Limited to a maximum URL length (typically around 2048 characters), though the exact limit depends on the browser and server.
- POST Request:
  - Data in Body: Data is included in the request body, not in the URL.

3. Security
- GET Request:
  - Less Secure: Data is visible in the URL, making it easy to bookmark or share. Sensitive data can be exposed in server logs and browser history.
  - Cachable: Responses can be cached by browsers and intermediate proxies.
- POST Request:
  - More Secure: Data is sent in the request body, not the URL. It is not visible in the browser history or server logs.
  - Not Cachable: Generally, responses to POST requests are not cached by browsers.

4. Idempotence
- GET Request:
  - Idempotent: Multiple identical requests should have the same effect as a single request. It does not change the state of the resource.
- POST Request:
  - Not Idempotent: Multiple identical requests can result in different effects, such as multiple entries being created.

5. Use Cases
- GET Request:
  - Fetching web pages.
  - Retrieving images or files.
  - Performing search queries.
- POST Request:
  - Submitting form data.
  - Uploading files.
  - Creating new resources (e.g., user registration).
  - Updating existing resources.

27. Can you tell me differences between List and Set in Java?

List and Set are both interfaces that extend the Collection interface but have distinct characteristics and usage scenarios. Here are the key differences between List and Set in Java:

1. Order
- List:
  - Ordered Collection: Elements in a List are stored in a specific order, which is maintained when the list is traversed.
  - Access by Index: Allows access to elements based on their index positions.
  - Example: ArrayList, LinkedList.
- Set:
  - Unordered Collection: Elements in a Set are not stored in a specific order (except for some implementations like LinkedHashSet which maintains insertion order, and TreeSet which maintains sorted order).
  - No Index-Based Access: Elements cannot be accessed by an index.
  - Example: HashSet, LinkedHashSet, TreeSet.

2. Duplicates
- List:
  - Allows Duplicates: A List can contain duplicate elements.
  - Example: List<String> list = new ArrayList<>(Arrays.asList("a", "b", "a")); // Valid
- Set:
  - No Duplicates: A Set cannot contain duplicate elements. If you try to add a duplicate, the existing element remains unchanged.
  - Example: Set<String> set = new HashSet<>(Arrays.asList("a", "b", "a")); // Only "a" and "b" will be stored

3. Performance
- List:
  - Performance: The performance depends on the implementation:
    - ArrayList: Provides fast random access and iteration, but slow insertions and deletions (except at the end of the list).
    - LinkedList: Provides fast insertions and deletions, but slow random access.
- Set:
  - Performance: The performance also depends on the implementation:
    - HashSet: Provides constant-time performance for basic operations (add, remove, contains, and size), assuming the hash function disperses elements properly among the buckets.
    - LinkedHashSet: Slightly slower than HashSet due to maintaining insertion order.
    - TreeSet: Provides log(n) time cost for basic operations, as it is based on a Red-Black tree.

4. Use Cases
- List:
  - When you need an ordered collection of elements.
  - When you need to access elements by their index.
  - When duplicates are allowed.
  - Examples: To-do lists, playlists, collections of items where order matters.
- Set:
  - When you need a collection with no duplicates.
  - When order does not matter (or you need specific order control with LinkedHashSet or TreeSet).
  - Examples: Unique user IDs, sets of unique elements, collections requiring fast membership tests.

5. Common Methods

- List:
  - add(int index, E element): Inserts the specified element at the specified position.
  - get(int index): Returns the element at the specified position.
  - set(int index, E element): Replaces the element at the specified position with the specified element.
  - indexOf(Object o): Returns the index of the first occurrence of the specified element.
- Set:
  - add(E e): Adds the specified element to the set if it is not already present.
  - remove(Object o): Removes the specified element from the set if it is present.
  - contains(Object o): Returns true if the set contains the specified element.
  - iterator(): Returns an iterator over the elements in the set.

28. How to run failed test cases in Java & selenium based automation framework?

1. Using the testng-failed.xml File
TestNG generates a testng-failed.xml file in the test-output directory when test cases fail. You can rerun the failed test cases using this file.

Steps:

1. Run your test suite: Execute your test suite as usual. If any test cases fail, TestNG will generate the testng-failed.xml file.
2. Locate testng-failed.xml: Navigate to the test-output directory of your project to find the testng-failed.xml file.
3. Run the testng-failed.xml file: You can rerun the failed test cases by running the testng-failed.xml file using your IDE or command line.

Command:
mvn test -Dsurefire.suiteXmlFiles=test-output/testng-failed.xml

2. Using a Custom Retry Analyzer

You can create a custom retry analyzer to automatically rerun failed test cases a specified number of times.

Steps:

1. Create a Retry Analyzer Class: Create a class that implements the IRetryAnalyzer interface.
2. Implement the Retry Logic: Define the retry logic within the retry method.
3. Apply the Retry Analyzer to Test Methods: Use the @Test annotation's retryAnalyzer attribute to apply the retry analyzer to your test methods.

Example:
RetryAnalyzer.java:

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

public class RetryAnalyzer implements IRetryAnalyzer {
    private int retryCount = 0;
    private static final int maxRetryCount = 3;

    @Override
    public boolean retry(ITestResult result) {
        if (retryCount < maxRetryCount) {
            retryCount++;
            return true;
        }
        return false;
    }
}
```

3. Using TestNG Listeners for Retry

You can also use TestNG listeners to retry failed test cases. This method involves creating a custom listener that implements the IAnnotationTransformer interface to dynamically set the retry analyzer.

Steps:

1. Create a Retry Analyzer Class: As shown in the previous example.
2. Create a Custom Listener: Implement the IAnnotationTransformer interface to set the retry analyzer dynamically.

Example: RetryListener.java:

```
import org.testng.IAnnotationTransformer;
import org.testng.annotations.ITestAnnotation;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

public class RetryListener implements IAnnotationTransformer {

    @Override
    public void transform(ITestAnnotation annotation,
    Class testClass, Constructor testConstructor, Method testMethod) {
        annotation.setRetryAnalyzer(RetryAnalyzer.class);
    }
}
```

29 can you explain me access modifires of java?

1. Private Access Modifier (private)

- Scope: Within the same class only.
- Usage: Members declared as private are not accessible outside the class they are declared in.

2. Default Access Modifier (no modifier)

- Scope: Within the same package.
- Usage: Members declared without any explicit access modifier (default access) are accessible only within the same package.

3. Protected Access Modifier (protected)

- Scope: Within the same package and subclasses (including those in different packages).
- Usage: Members declared as protected are accessible within the same package and by subclasses in other packages.

4. Public Access Modifier (public)

- Scope: Everywhere.
- Usage: Members declared as public are accessible from any other class.

Modifier	Class	Package	Subclass (Same Package)	Subclass (Different Package)	Global (Other Packages)
`private`	Yes	No	No	No	No
(default)	Yes	Yes	Yes	No	No
`protected`	Yes	Yes	Yes	Yes	No
`public`	Yes	Yes	Yes	Yes	Yes

30. What are the differences between Interface and Abstract class?

1. Definition and Purpose
- Interface:
  - An interface is a reference type in Java, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types.
  - The purpose of an interface is to specify a contract that classes can implement.
- Abstract Class:
  - An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed.
  - It can have a mix of abstract methods (without implementation) and concrete methods (with implementation).

2. Implementation and Inheritance
- Interface:
  - A class can implement multiple interfaces.
  - Interfaces are implemented using the implements keyword.
  - All methods in an interface are implicitly public and abstract (except default and static methods introduced in Java 8).
- Abstract Class:
  - A class can inherit from only one abstract class (single inheritance).
  - Abstract classes are extended using the extends keyword.
  - Abstract classes can have both abstract methods and concrete methods.

3. Method Implementation
- Interface:
  - Cannot contain any method implementations (prior to Java 8).
  - Java 8 introduced default methods (with default keyword) and static methods with implementation.
  - Java 9 introduced private methods in interfaces.

- Abstract Class:
  - Can contain method implementations.
  - Can have abstract methods, which must be implemented by subclasses.
  - Can contain constructors, fields, and methods with any access modifier.

4. Fields and Constants
- Interface:
  - Can only contain public, static, and final fields (constants).
  - Fields in an interface are implicitly public, static, and final.
- Abstract Class:
  - Can have instance variables and static fields.
  - Fields can have any access modifier (private, protected, public).

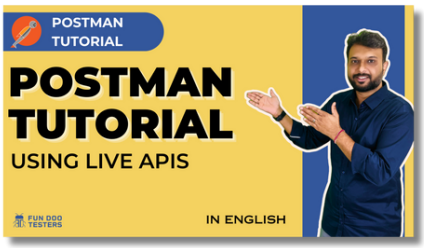
5. Constructors
- Interface:
  - Cannot have constructors.
  - Interfaces cannot be instantiated.
- Abstract Class:
  - Can have constructors.
  - Constructors are used to initialize fields in the abstract class or to perform setup tasks.

6. Use Cases
- Interface:
  - Used to define a contract that multiple classes can implement, ensuring a certain set of methods are available.
  - Ideal for defining capabilities that can be shared across unrelated classes (e.g., Comparable, Serializable).
- Abstract Class:
  - Used when you want to share code among closely related classes.
  - Ideal for defining a base class that provides common functionality, while leaving specific implementations to subclasses.

Note: These interview questions and answers have been identified from multiple resources and through discussions with interviewers for reference purposes and to improve knowledge.

Latest Videos

Free Tutorials



In English

In Hindi

Checkout "**Fun Doo Testers**"
YouTube

For more insights on Testing & Test Automation

Follow "Fun Doo Testers"

Write us on: contact@fundootesters.com