Getting Started with TestNG: A Complete Beginner's Guide to Java Test Automation

TestNG Blog 01

1. Introduction to TestNG

TestNG (Test Next Generation) is a testing framework inspired by JUnit and NUnit, designed to simplify and streamline the process of writing automated test cases in Java. It supports annotations, test prioritization, grouping, parallel execution, and dependency management, making it a popular choice for Java-based test automation.

Dewmini Abeywardhana

2. Advantages of Using TestNG

- Clean and simple annotation-based test design
- Supports prioritization and test execution order control
- Built-in test grouping and dependency management
- Generates HTML and XML test reports automatically
- Compatible with CI/CD tools like Jenkins
- Supports parallel execution for faster test runs

3. Setting Up a Java Project with TestNG in IntelliJ IDE

🛠 **Step-by-Step Instructions:**

- Open IntelliJ IDEA and select New Project.
- Choose Java as the project type and click Next.
- Name the project (e.g., TestNGDemo) and click Finish.
- Right-click the src folder > New > Java Class > name it (e.g., FirstTest).
- To add TestNG as a dependency:

If using Maven:

Add this to pom.xml (Sample)

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

If not using Maven:

Go to File > Project Structure > Libraries > Add TestNG jar manually.

4. Writing and Running the First TestNG Test

✍ **Sample Test Code:**

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

public class FirstTest {

    @Test
    public void sampleTest() {
```

Dewmini Abeywardhana

```
        System.out.println("TestNG test executed successfully.");
    }
}
```

▶ **Run the Test:**

- Right-click on the test class > Run 'FirstTest'.
- Check the console for output and test result.

5. Default Execution Order of TestNG Tests

By default, TestNG executes test methods in alphabetical order if no priority is specified. This may lead to unexpected execution sequences, especially when tests depend on specific workflows.

6. Controlling Execution Order Using Priority

The priority attribute in the @Test annotation helps control the execution order:

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


@Test(priority = 2)
public void dashboardTest() {}
```

- Lower priority numbers are executed first.
- If priority is not defined, execution follows alphabetical order.

Dewmini Abeywardhana

7. Skipping a Test Case

A test can be skipped using the enabled attribute:

```
@Test(enabled = false)
public void disabledTest() {
   System.out.println("This test will be skipped.");
}
```

This is useful for temporarily disabling a test without removing the code.

8. Understanding Test Dependencies

Test dependency is when one test case depends on another for successful execution. TestNG handles such scenarios using dependsOnMethods.

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

@Test(dependsOnMethods = "loginTest")
public void dashboardTest() {}
```

Here, dashboardTest will only be executed if loginTest passes.

Dewmini Abeywardhana

9. Using alwaysRun to Ensure Execution

Sometimes, certain tests must run regardless of whether the dependent test passes or fails. This can be handled using the alwaysRun attribute:

```
@Test(dependsOnMethods = "loginTest", alwaysRun = true)
public void logoutTest() {
   System.out.println("Cleanup step: logout test executed.");
}
```

alwaysRun = true ensures this test executes even if loginTest fails.

10. Conclusion

TestNG offers a solid foundation for creating maintainable, organized, and flexible test cases in Java. Key features such as execution order control, skipping, and test dependencies empower testers to handle real-world scenarios efficiently. Practicing these basics lays the groundwork for more advanced topics like data providers, parameterization, and test grouping.

Stay tuned, keep learning, and happy testing! 🧪✨

Dewmini Abeywardhana