@techwithvishalraj

JUnit 5

Follow

Share

@techwithvishalraj

What is JUnit?

- **JUnit is a widely used testing framework in Java for writing and running unit tests. It allows developers to write test cases that verify the behavior of individual units of code, typically methods or classes, to ensure they work as expected.**

- **JUnit 5 requires Java 8 (or higher) at runtime. However, you can still test code that has been compiled with previous versions of the JDK.**

JUnit 5 = *JUnit Platform + JUnit Jupiter + JUnit Vintage*

- **The JUnit Platform serves as a foundation for launching testing frameworks on the JVM. It also defines the TestEngine API for developing a testing framework that runs on the platform.**

- **JUnit Jupiter is the combination of the programming model and extension model for writing tests and extensions in JUnit 5. The Jupiter sub-project provides a TestEngine for running Jupiter based tests on the platform.**

- **JUnit Vintage provides a TestEngine for running JUnit 3 and JUnit 4 based tests on the platform. It requires JUnit 4.12 or later to be present on the class path or module path.**

- **All core annotations are located in the org.junit.jupiter.api package in the junit-jupiter-api module.**

Why Use JUnit?

- **Automated Testing:** JUnit automates the process of running tests, so you can run all your tests with a single command, reducing manual effort.
- **Early Bug Detection:** By writing unit tests, you can catch bugs early in the development cycle. Each time you make a change to your code, you can rerun the tests to ensure that the new code doesn't break existing functionality.
- **Regression Testing:** JUnit allows you to test your codebase continuously as it evolves. Running your test suite regularly ensures that new features or changes do not introduce regressions (i.e., errors in previously working functionality).
- **Code Quality:** Writing unit tests encourages you to think about how your code is structured, leading to better design, modular code, and fewer defects.
- **Integration with Build Tools:** JUnit integrates seamlessly with build tools like Maven, Gradle, and CI/CD pipelines (like Jenkins). It helps automate the testing process in continuous integration systems.
- **Support for TDD:** JUnit is frequently used in Test-Driven Development (TDD), where you write the tests before writing the actual code. This process helps you define the desired behavior before implementation, resulting in cleaner, more reliable code.

Key Features of JUnit

- **Annotations:** JUnit uses annotations such as @Test to mark methods as test methods, @Before and @After to set up and clean up before and after each test, and @BeforeClass and @AfterClass for global setup and teardown.
- **Assertions:** JUnit provides assertion methods that allow you to check if the actual result matches the expected result.
    1. assertEquals(expected, actual)
    2. assertTrue(condition)
    3. assertFalse(condition)
    4. assertNotNull(object)
    5. assertThrows(Class<T> expectedType, Executable executable)

- **Test Runners:** JUnit test runners execute the test cases and report the results, showing which tests passed, failed, or were skipped.

JUnit Jupiter supports the following annotations for configuring tests and extending the framework.

@techwithvishalraj

Annotation	Description
@Test	Denotes that a method is a test method. Unlike JUnit 4's @Test annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are inherited unless they are overridden.
@ParameterizedTest	Denotes that a method is a parameterized test. Such methods are inherited unless they are overridden.
@RepeatedTest	Denotes that a method is a test template for a repeated test. Such methods are inherited unless they are overridden.
@TestFactory	Denotes that a method is a test factory for dynamic tests. Such methods are inherited unless they are overridden.
@TestTemplate	Denotes that a method is a template for test cases designed to be invoked multiple times depending on the number of invocation contexts returned by the registered providers. Such methods are inherited unless they are overridden.
@TestClassOrder	Used to configure the test class execution order for @Nested test classes in the annotated test class. Such annotations are inherited.
@TestMethodOrder	Used to configure the test method execution order for the annotated test class; similar to JUnit 4's @FixMethodOrder. Such annotations are inherited.
@TestInstance	Used to configure the test instance lifecycle for the annotated test class. Such annotations are inherited.
@DisplayName	Declares a custom display name for the test class or test method. Such annotations are not inherited.
@DisplayNameGeneration	Declares a custom display name generator for the test class. Such annotations are inherited.
@BeforeEach	Denotes that the annotated method should be executed before each @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4's @Before. Such methods are inherited unless they are overridden.
@AfterEach	Denotes that the annotated method should be executed after each @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4's @After. Such methods are inherited unless they are overridden.

JUnit Jupiter supports the following annotations for configuring tests and extending the framework.

@techwithvishalraj

Annotation	Description
@BeforeAll	Denotes that the annotated method should be executed before all @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4's @BeforeClass. Such methods are inherited unless they are overridden and must be static unless the "per-class" test instance lifecycle is used.
@AfterAll	Denotes that the annotated method should be executed after all @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4's @AfterClass. Such methods are inherited unless they are overridden and must be static unless the "per-class" test instance lifecycle is used.
@Nested	Denotes that the annotated class is a non-static nested test class. On Java 8 through Java 15, @BeforeAll and @AfterAll methods cannot be used directly in a @Nested test class unless the "per-class" test instance lifecycle is used. Beginning with Java 16, @BeforeAll and @AfterAll methods can be declared as static in a @Nested test class with either test instance lifecycle mode. Such annotations are not inherited.
@Tag	Used to declare tags for filtering tests, either at the class or method level; analogous to test groups in TestNG or Categories in JUnit 4. Such annotations are inherited at the class level but not at the method level.
@Disabled	Used to disable a test class or test method; analogous to JUnit 4's @Ignore. Such annotations are not inherited.
@AutoClose	Denotes that the annotated field represents a resource that will be automatically closed after test execution.
@Timeout	Used to fail a test, test factory, test template, or lifecycle method if its execution exceeds a given duration. Such annotations are inherited.
@TempDir	Used to supply a temporary directory via field injection or parameter injection in a lifecycle method or test method; located in the org.junit.jupiter.api.io package. Such fields are inherited.
@ExtendWith	Used to register extensions declaratively. Such annotations are inherited.
@RegisterExtension	Used to register extensions programmatically via fields. Such fields are inherited.

@techwithvishalraj

Explore more about Junit and its feature on its Official Website.

https://junit.org/junit5/docs/snapshot/user-guide/

Thank you!

vishal-bramhankar

techwithvishalraj

Vishall0317