JUNIT

What is JUnit?

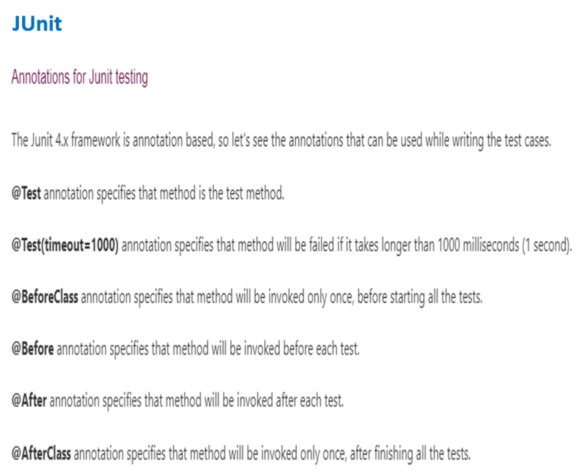
JUnit is a **unit testing framework for Java**. It allows developers to write and run tests to ensure their code works as expected. It's part of the **xUnit family** of testing frameworks and is widely used in **test-driven development (TDD)**.

Advantages of JUnit

Feature	Benefit
✅ **Automated Testing**	Run tests automatically, no need to manually verify output.
✅ **Regression Testing**	Quickly detect if new code breaks existing features.
✅ **TDD Support**	Encourages writing tests before code.
✅ **Integrated with IDEs**	Easily run/debug tests from Eclipse, IntelliJ, etc.
✅ **CI/CD Ready**	Works smoothly with Jenkins, Maven, Gradle, etc.
✅ **Lightweight & Fast**	Simple to use and quick to execute.

📱 Applications of JUnit

- ✅ Testing **methods and classes** in Java applications
- ✅ Used in **TDD (Test Driven Development)**
- ✅ Used in **integration testing** and **API testing**
- ✅ Helps ensure **reliability, maintainability**, and **code quality**


JUnit

Annotations for Junit testing

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

@Test annotation specifies that method is the test method.

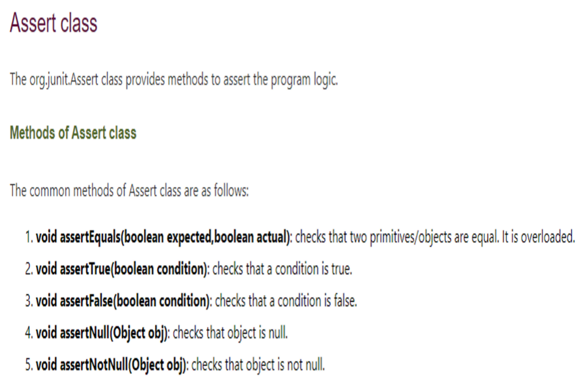
@Test(timeout=1000) annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

@BeforeClass annotation specifies that method will be invoked only once, before starting all the tests.

@Before annotation specifies that method will be invoked before each test.

@After annotation specifies that method will be invoked after each test.

@AfterClass annotation specifies that method will be invoked only once, after finishing all the tests.


Assert class

The org.junit.Assert class provides methods to assert the program logic.

Methods of Assert class

The common methods of Assert class are as follows:

1. **void assertEquals(boolean expected,boolean actual)**: checks that two primitives/objects are equal. It is overloaded.
2. **void assertTrue(boolean condition)**: checks that a condition is true.
3. **void assertFalse(boolean condition)**: checks that a condition is false.
4. **void assertNull(Object obj)**: checks that object is null.
5. **void assertNotNull(Object obj)**: checks that object is not null.

Example: Calculator and CalculatorTest

```
package com.junit;
public class Calculator {
	public static void main(String[] args) {
		Calculator cal = new Calculator();
		System.out.println(cal.add(1, 2));
		System.out.println(cal.divide(6, 3));
		System.out.println(cal.getNullValue());
	}
	public int add(int a, int b) {
		return a + b;
	}
	public int divide(int a, int b) {
		return a / b;
	}
	public String getNullValue() {
		return null;
	}
}
```

```
3
2
null
```

CalculatorTest.java (JUnit 4 version with all annotations & assertions)

```java
package com.junit;
public class CalculatorTest {
  private static Calculator calculator;
  @BeforeClass
  public static void setupBeforeAllTests() {
     System.out.println("BeforeClass: Executed once before all test methods.");
     calculator = new Calculator();
  }
  @Before
  public void setupBeforeEachTest() {
     System.out.println("Before: Executed before each test method.");
  }
  @Test
  public void testAddition() {
     int result = calculator.add(10, 5);
     assertEquals("Addition test failed", 15, result);              // assertEquals
     assertTrue("Result should be greater than 10", result > 10);   // assertTrue
     assertFalse("Result should not be negative", result < 0);      // assertFalse
  }
  @Test(timeout = 1000)
  public void testWithTimeout() {
     // simple operation that completes within 1 second
     int result = calculator.add(3, 2);
     assertEquals(5, result);
  }
  @Test
  public void testNullAndNotNull() {
     String value = calculator.getNullValue();
     assertNull("Expected null value", value);                      // assertNull
     String notNull = "JUnit";
     assertNotNull("Expected non-null value", notNull);             // assertNotNull
  }
  @After
  public void tearDownAfterEachTest() {
     System.out.println("After: Executed after each test method.");
  }
  @AfterClass
  public static void tearDownAfterAllTests() {
     System.out.println("AfterClass: Executed once after all test methods.");
  }
}
```

Output

```
BeforeClass: Executed once before all test methods.
Before: Executed before each test method.
After: Executed after each test method.
... (repeated for each test)
AfterClass: Executed once after all test methods.
```