JUnit 5

Assertion, Assumption, Tagging, Parameterization, Parallel Execution

JUnit5 Introduction

What is JUnit5?

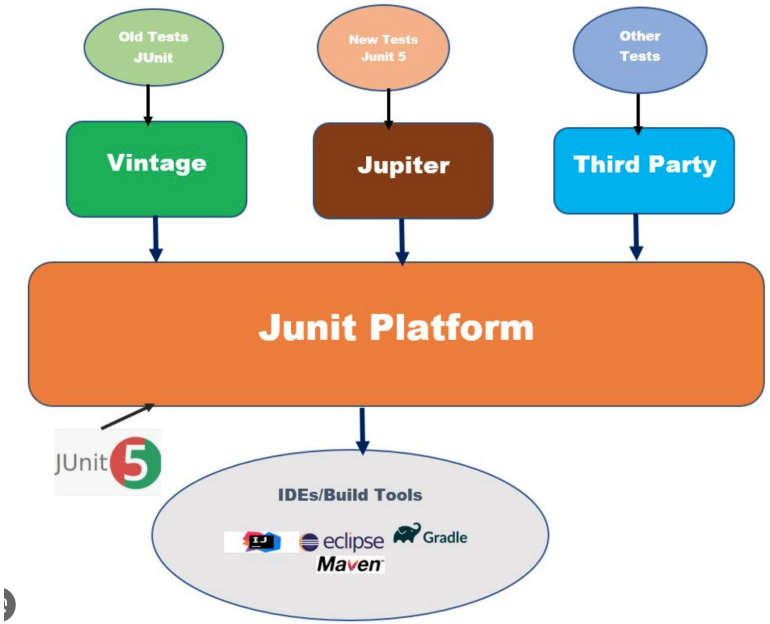
It's one of the most popular tools for **Unit Testing** Java applications and it is widely used in the industry.

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage.

JUnit Platform – this module is responsible for test discovery, execution, and reporting.

JUnit Jupiter - Providing APIs for assertions

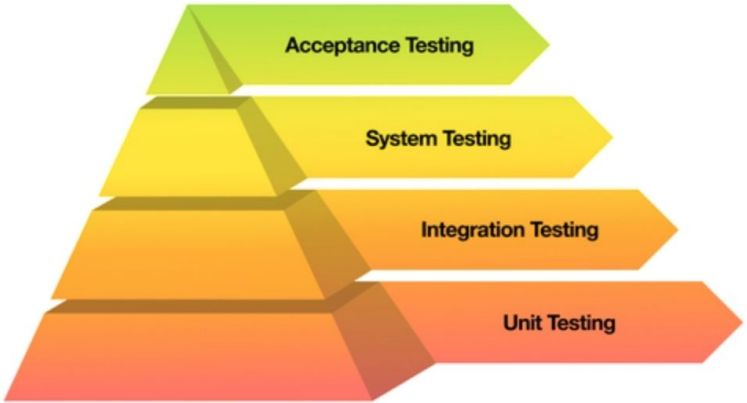
JUnit Vintage - Compatibility with older version of JUnit



Unit Testing

What is Unit Testing?

“Unit” means the smallest piece of software. As a type of white box testing which requires knowledge of the code being tested, are usually performed by **developers**. It is the first level of testing.



Assertions

What is Assertion?

They are static methods in the org.junit.jupiter.api.Assertions class. If the assertion conditions are not met during the test run, an AssertionError is thrown, indicating that the test has failed.

- assertEquals/NotEquals()
- assertTrue/False()
- assertNull/NotNull()
- assertSame/NotSame()

Advanced Assertions

- **assertThrows()**
  - Assert that the exception thrown by the code block is the same or belongs to the subtypes of given classes.
- **assertThrows Exactly()**
  - Assert that an exact type of exception has been thrown by the code block.
- **assertTimeout() / assertTimeoutPreemptively()**
  - Assert that a code block can finish execution within a certain time period.
- **assertAll()**
  - Check and report all the assertions

Assumption

What is Assumption?

Assumptions in JUnit 5 are used to perform conditional test execution. When an assumption fails, it doesn't fail the test, but rather the test is simply skipped **(A test has three states: success, fail and skipped).**

In JUnit5, we have:

- assumeTrue()
- assumeFalse()
- assumingThat()

Lifecycle Methods

- BeforeAll()
  - Executed before all the methods, must be static by default
- BeforeEach()
  - Executed before every method
- AfterAll()
  - Executed after all the methods, must be static by default
- AfterEach()
  - Executed after every method

Conditional Execution

Except for using assumptions, we can use @Enable annotation to perform conditional execution of tests.

- EnabledIf()
  - Based on customized conditions
- EnabledOnOs()
  - Based on the operating system that the tests are running on
- EnabledOnJre()
  - Based on the version of Java Running Environment

Execution Order

By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious. For most of the time, we should not care about the execution order. But there are times when it is necessary to enforce a specific test method execution order.

Types of Execution Order

- By following @Order Annotation
- By following the methodName
- By following the displayName
- Random order

Test Instance

Q: During the test execution, how many instances of the test class will be created?

A: Every test method belongs to one test instance, so it is equal to the number of test methods

Q: Why do the @BeforeAll and @AfterAll methods must be static?

A: The purpose of these two methods is to initialize/tear down some objects that can be shared across different test methods, the only way to share is through static variables.

Q: What if I only want one test instances created?

A: @TestInstance(TestInstance.Lifecycle.PER_CLASS)

Advanced Topics

- Tagging & Filtering
- Nested Tests
- Parameterization
- Parallel Execution

Any Questions?