

FULL STACK

JUnit 5 Testing



A Day in the Life of a Full Stack Developer

All the desired results were achieved in the last sprint and the sprint was marked as completed. Joe has been appreciated for the last project.

He is now assigned a new project which he needs to complete in a week. As a part of developing an ecommerce web application, a test-suite has to be created to do unit testing of all backend components in the web application. This project will test the user authentication class. This project will be a standalone Java application. Joe has to test only the classes that have the business logic.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Outline the steps involved in migrating from JUnit 4 to JUnit 5
- 🕒 Describe JUnit 5 architecture
- 🕒 Demonstrate how assertions work
- 🕒 Define nested and repeated tests and dynamic tests
- 🕒 Use a dependency injection



FULL STACK

Moving from JUnit 4 to JUnit 5

Moving from JUnit 4 to JUnit 5

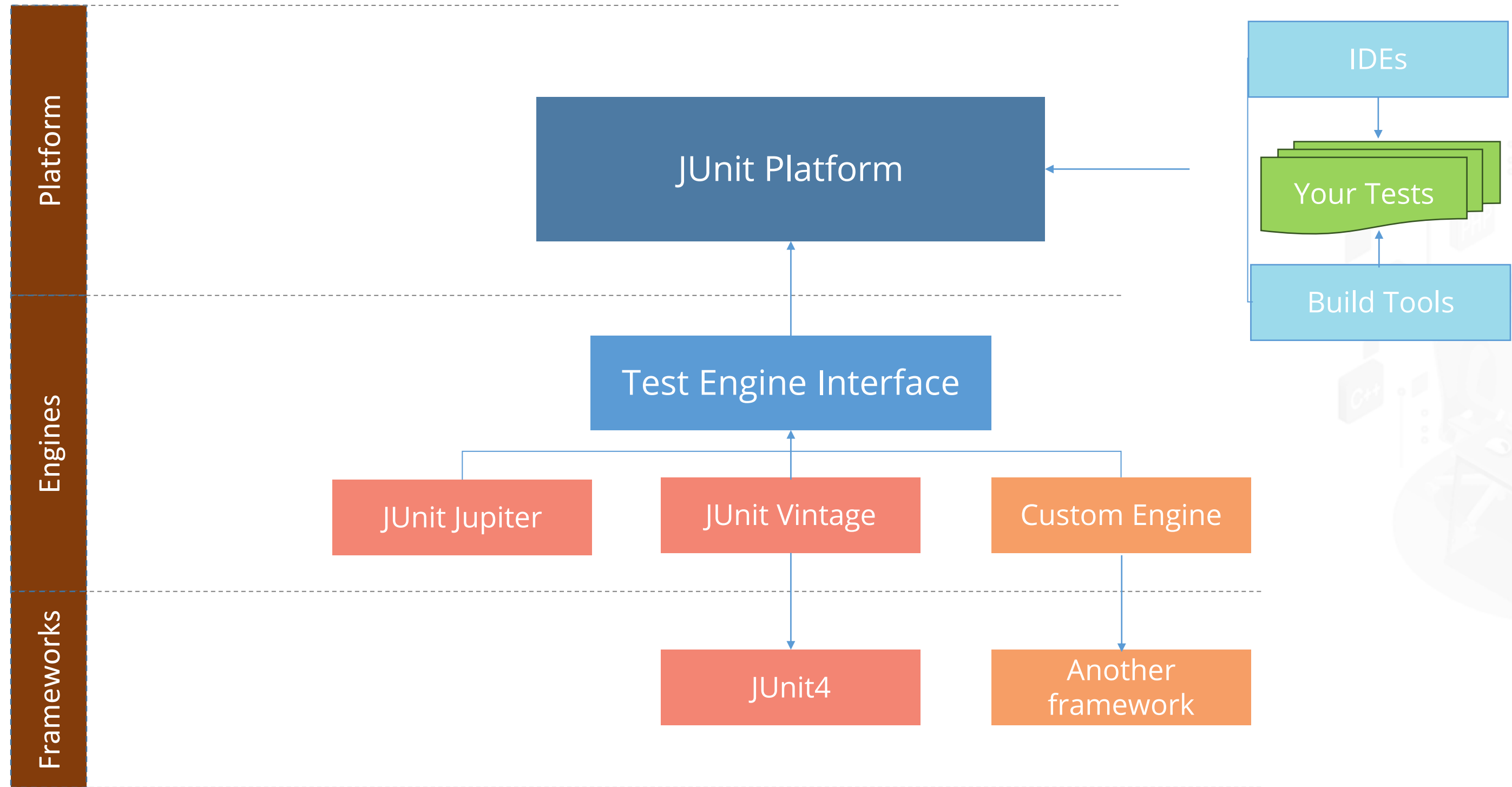
JUnit 5 provides a gentle migration path via the JUnit Vintage test engine, which allows for execution of legacy test cases on top of the JUnit platform.

Feature	JUnit 4	JUnit 5
Annotation package	org.junit	org.junit.jupiter.api
Declaring a test	@Test	@Test
Setup for all tests	@BeforeClass	@BeforeAll
Setup per test	@Before	@BeforeEach
Tear down for all tests	@AfterClass	@AfterAll
Tear down per test	@After	@AfterEach
Tagging and filtering	@Category	@Tag
Disabling a test method or a class	@Ignore	@Disabled
Nested tests	NA	@Nested
Repeated tests	Using custom rule	@Repeated
Dynamic tests	NA	@TestFactory
Test templates	NA	@TestTemplate

FULL STACK

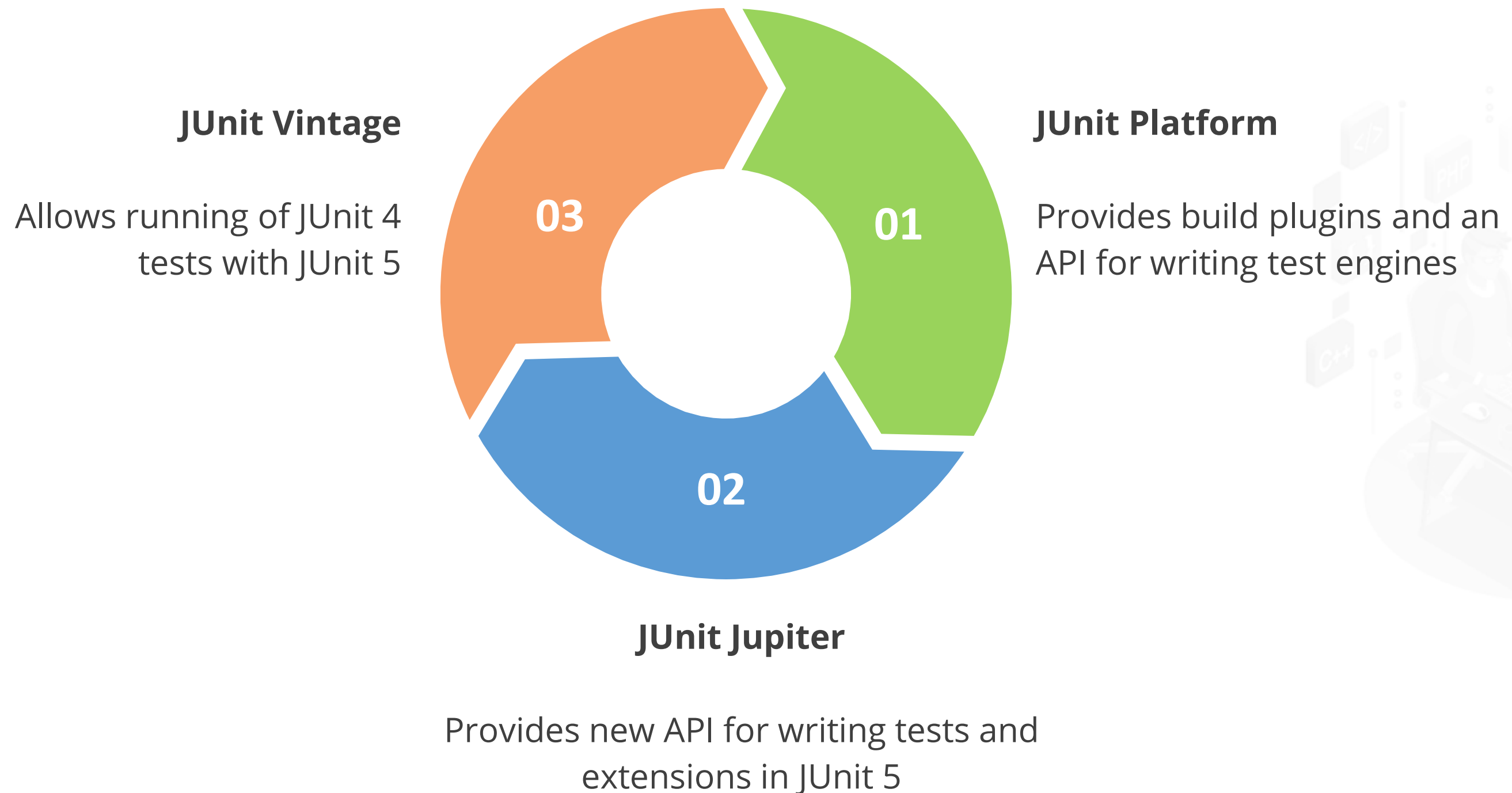
JUnit 5 Architecture

JUnit 5 Architecture



JUnit 5 Architecture

JUnit 5 is split into three different subprojects.

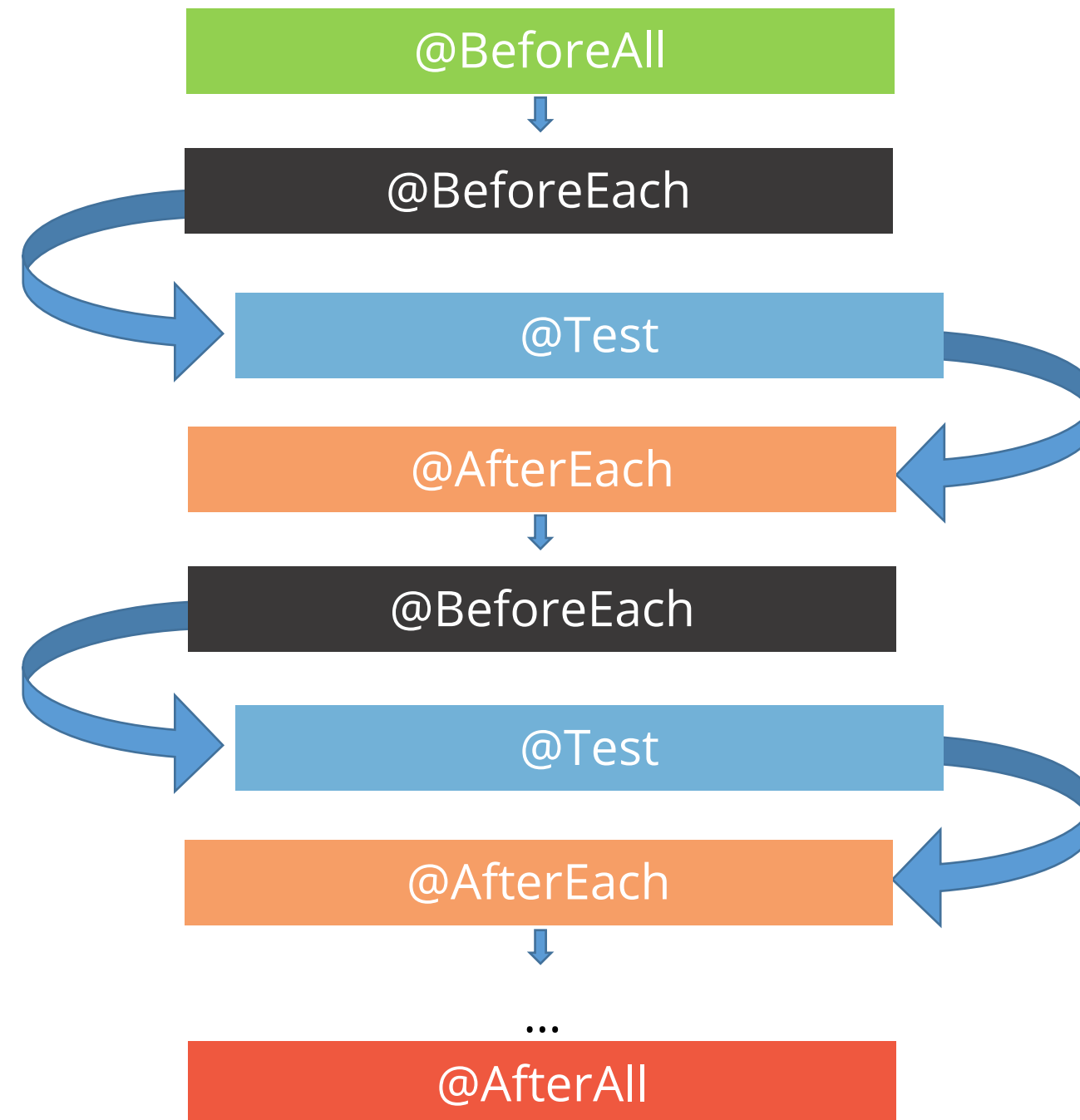


FULL STACK

Standard Tests

Standard Tests

In JUnit 5, the `@Test` annotation represents a test. Any public method can be annotated with `@Test` to make it a test method.



Annotations

The following annotations are used for configuring tests in JUnit5:

Annotation	Description
@Test	Denotes a test method
@DisplayName	Declares a custom display name for the test class or test method
@BeforeEach	Denotes that the annotated method should be executed before each test method
@AfterEach	Denotes that the annotated method should be executed after each test method
@BeforeAll	Denotes that the annotated method should be executed before all test methods
@AfterAll	Denotes that the annotated method should be executed after all test methods
@Disable	Used to disable a test class or test method
@Nested	Denotes that the annotated class is a nested and non-static test class
@Tag	Declares tags for filtering tests
@ExtendWith	Registers custom extensions



JUnit5: Standard Tests



Duration: 10 min.

Problem Statement:

Write a program to demonstrate standard tests.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate standard tests:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate standard tests
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories



FULL STACK

Assertions

Assertions

Every test method is evaluated using assertions so that the test can be executed. JUnit Jupiter assertions are found in the **org.junit.jupiter.api.Assertions** class.

Assertion	Description
assertEquals(expected, actual)	Fails when expected does not equal actual
assertFalse(expression)	Fails when expression is not false
assertNull(actual)	Fails when actual is not null
assertNotNull(actual)	Fails when actual is null
assertAll()	Groups many assertions; every assertion is executed even if one or more of them fail
assertTrue(expression)	Fails if expression is not true
assertThrows()	Class to be tested is expected to throw an exception

Assertions



Duration: 10 min.

Problem Statement:

Write a program to demonstrate assertions.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate assertions:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate assertions
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories



Conditional Test Execution

Conditional Test Execution

JUnit 5 extension model is used to establish custom conditions for test execution. The **ExecutionCondition** is used for this purpose.

Conditional Annotation Type	Annotation List
Operating system conditions	@EnabledOnOs, @DisabledOnOs
Java runtime environment conditions	@EnabledOnJre, @DisabledOnJre
System property conditions	@EnabledIfSystemProperty, @DisabledIfSystemProperty
Environment variable conditions	@EnabledIfEnvironmentVariable, @DisabledIfEnvironmentVariable
Script-based conditions	@EnabledIf, @DisabledIf

Conditional Test Execution



Duration: 10 min.

Problem Statement:

Write a program to demonstrate conditional test executions.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate conditional test execution:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate conditional test execution
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

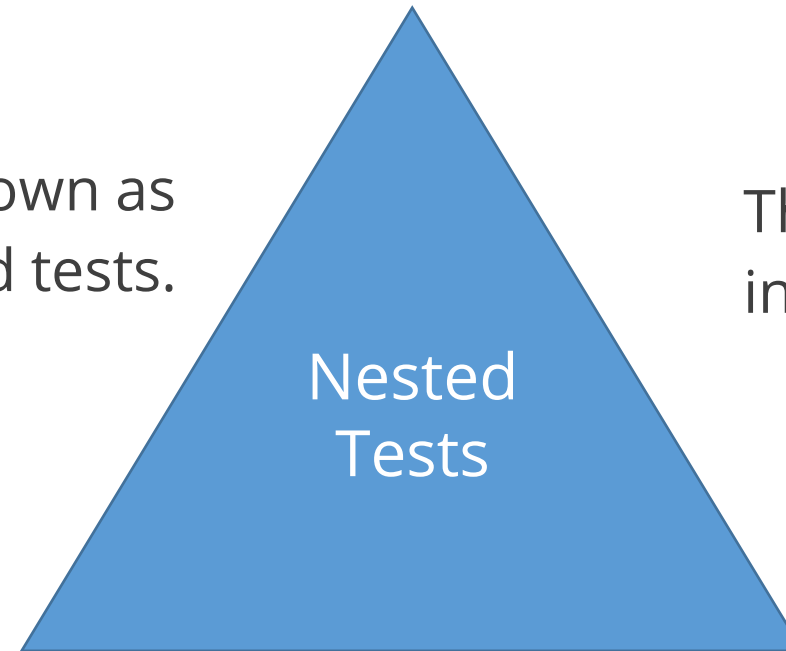


Nested and Repeated Tests

Nested Tests

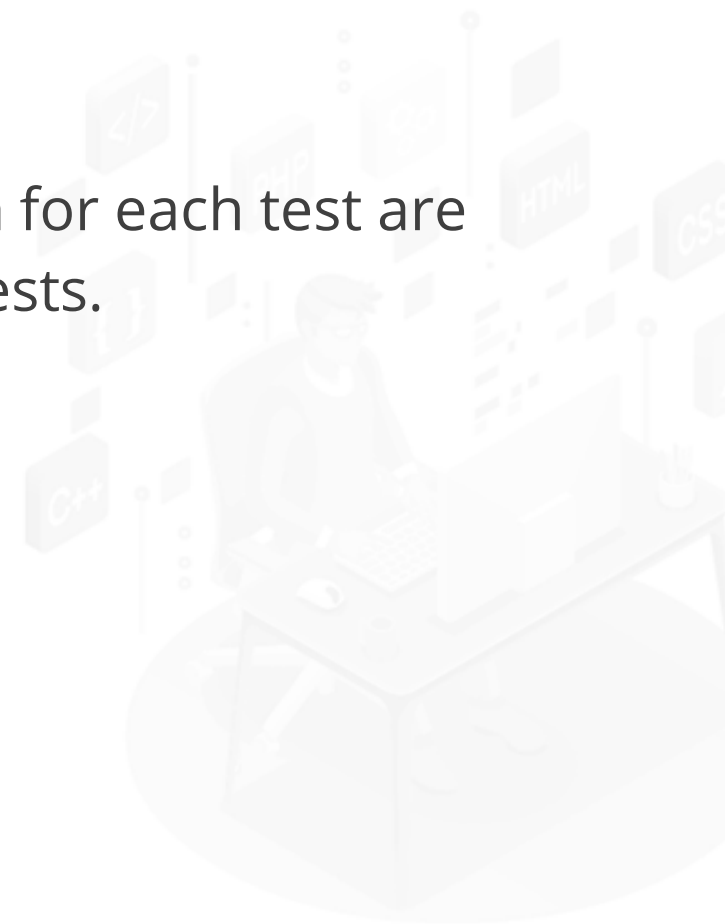
@Nested annotation is used to mark a nested class to be included in the test cases.

Only non-static nested classes, also known as inner classes, can serve as @Nested tests.



The setup and tear down for each test are inherited in the nested tests.

@TestInstance(Lifecycle.PER_CLASS) is used to override the restriction that Java does not allow static members in inner classes.



Repeated Tests

A test can be repeated a fixed number of times by annotating a method with `@RepeatedTest`, specifying the total number of repetitions desired.

`{displayName}`

Name of the `@RepeatedTest` method

Dynamic
Placeholders
Supported

`{totalRepetitions}`

total number of repetitions

`{currentRepetition}`

Current repetition count



Nested and Repeated Tests



Duration: 10 min.

Problem Statement:

Write a program to demonstrate nested and repeated tests.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate nested and repeated tests:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate nested and repeated tests
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories



FULL STACK

Dynamic Tests

Dynamic Tests

JUnit 5 allows to generate tests at runtime by a factory method that is annotated with `@TestFactory`.

Properties of Dynamic Tests:

Must return a Stream, Collection, Iterable, or Iterator of `DynamicNode` instances

Stream returned properly closed by calling `stream.close()`,

Dynamic Tests



Duration: 10 min.

Problem Statement:

Write a program to demonstrate dynamic tests.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate dynamic tests:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate dynamic tests
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories



FULL STACK

Dependency Injection

Dependency Injection

JUnit5 allows both test constructors and methods to include parameters.

TestInfoParameterResolver

Supplies an instance of
TestInfo corresponding to the
current container

RepetitionInfoParameter
Resolver

Supplies an instance of
RepetitionInfo

TestReporterParameter
Resolver

Supplies an instance of
TestReporter

Dependency Injection



Duration: 10 min.

Problem Statement:

Write a program to demonstrate a dependency injection.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate dependency injection:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate a dependency injection
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories



Key Takeaways

- JUnit is a unit testing framework for Java programming language, and it follows the TDD approach
- JUnit 5 allows multiple runners to work simultaneously
- Dependency injection is a technique in which one object supplies dependencies to the other
- Assertions are validation rules used to test results that check your target service platform



Handling User Authentication

Duration: 30 min.

Problem Statement:

Set up a standalone project to do unit testing of the user authentication class which is used in the main web application. The objective is to create a JUnit class that will test all aspects of the authentication class.

