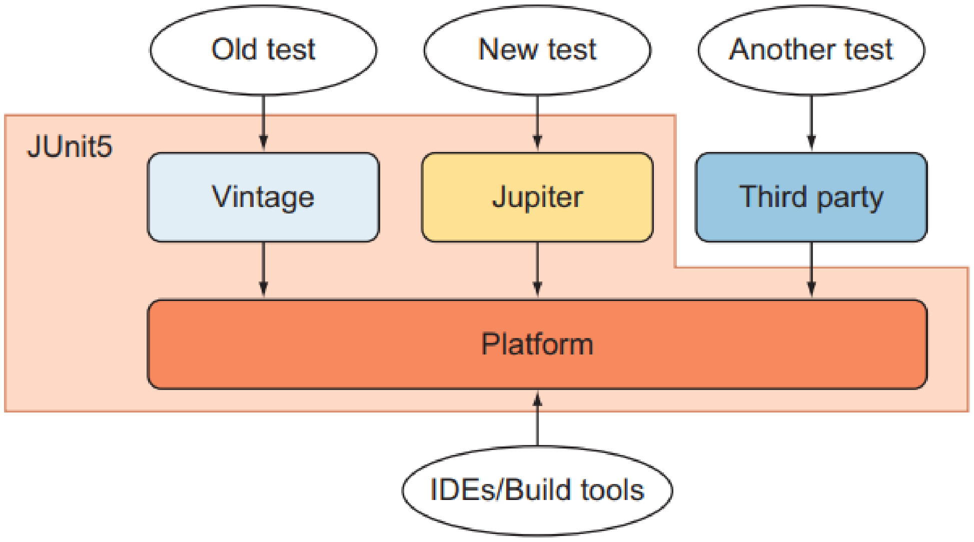
Junit 5



JUnit Platform
JUnit Jupiter
JUnit Vintage

Figure 3.7 The modular architecture of JUnit 5

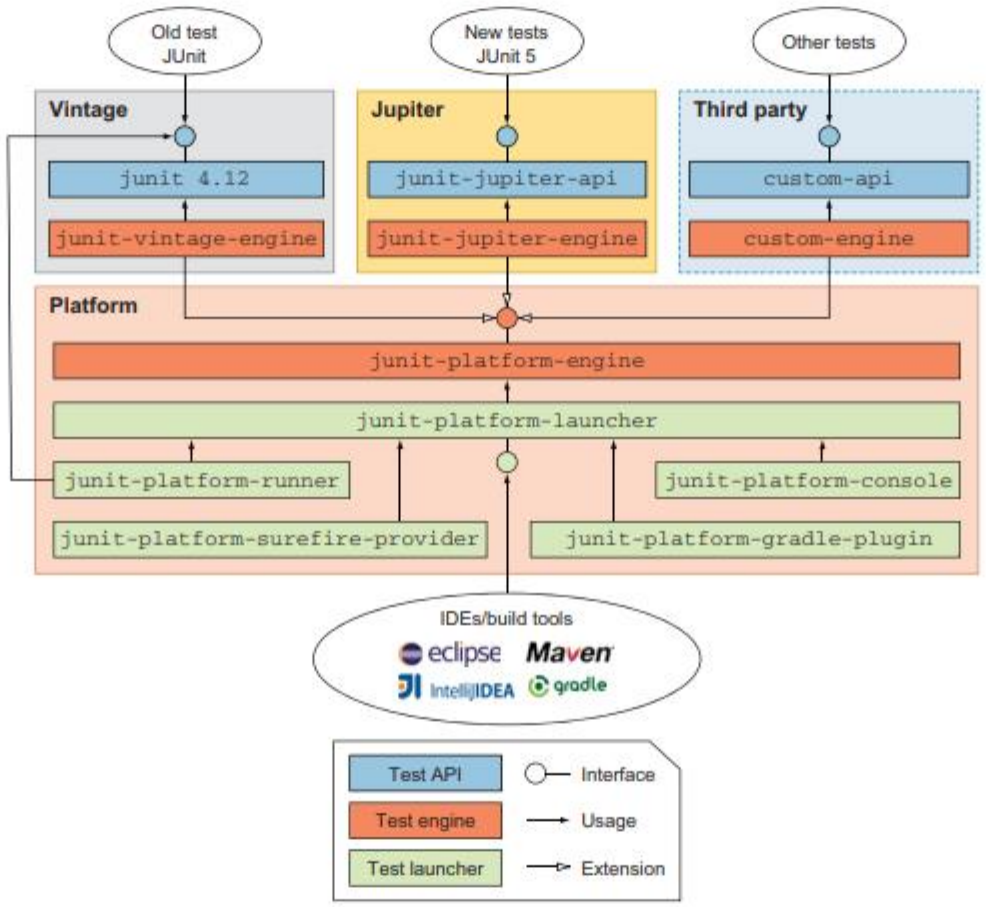


Figure 3.9 A detailed picture of the JUnit 5 architecture

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. Furthermore, the platform provides a Console Launcher to launch the platform from the command line and the JUnit Platform Suite Engine for running a custom test suite using one or more test engines on the platform. First-class support for the JUnit Platform also exists in popular IDEs (see IntelliJ IDEA, Eclipse, NetBeans, and Visual Studio Code) and build tools (see Gradle, Maven, and Ant).

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.

Reference: JUnit 5 User Guide
Book: Junit 5 in Action third edition

Junit -4

JUnit 4, released in 2006, has a simple, monolithic architecture. All of its functionality is concentrated inside a single jar file (figure 3.4). If a programmer wants to use JUnit 4 in a project, all they need to do is add that jar file on the classpath

Figure 3.4 The monolithic architecture of JUnit 4: a single jar file

Junit 4?
- is there any problem with junit -4 – No, this is still working. But new features are not available in Junit-4
- Junit 4 is > 10-year-old
- not up to date with newer testing patterns
- not up to date with java language features
- monolithic architecture
- bugs and features requests piled up

Junit 5

A crowdfunding campaign started, companies donated
- initiated by the core team
- called junit lambda
- many companies and individual contributed
- started the path to JUnit 5
- Junit 5 != Junit 4 + 1(feature)
- Junit 5 with new architecture
- new project was created for Junit 5

- Jupiter name was given because 5th planet in universe.

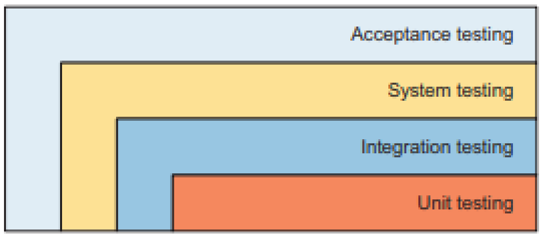


Figure 5.2 The four types of tests, from the innermost (narrowest) to the outermost (broadest)

pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.9.3</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.9.3</version>
        <scope>test</scope>
    </dependency>
</dependencies>


<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
        </plugin>
    </plugins>
</build>
```

src/ main/java

```
public class DivideNumber {
    public double divideNumbers(double a, double b){
        return a/b;
    }
}
```

src/test/java

```
class DivideNumberTest {

    private DivideNumber unitUnderTest = new DivideNumber();

    @Test
    @DisplayName("Test double division method")
    void testDivideMethod() {
        double result = unitUnderTest.divideNumbers(10.0, 2.0);
        Assertions.assertEquals(5.0, result);
    }

    @Test
    @DisplayName("Test division when divide by zero")
    void testDivideByZero() {
        double result = unitUnderTest.divideNumbers(10.0, 0.0);
        Assertions.assertEquals(Double.POSITIVE_INFINITY, result);
    }

    @Test
    @DisplayName("Test division when zero divide by zero")
    void testZeroDivideByZero() {
        double result = unitUnderTest.divideNumbers(0.0, 0.0);
        Assertions.assertEquals(Double.NaN, result);
    }

    @Test
    @DisplayName("Test division when divide by zero")
    void testNegativeNumberDivideByZero() {
        double result = unitUnderTest.divideNumbers(-10.0, 0.0);
        Assertions.assertEquals(Double.NEGATIVE_INFINITY, result);
    }
}
```

@BeforeAll	Executes only once per class before all tests. This method should be static. `@BeforeAll public static void beforeAll() { //connect db System.out.println("BeforeAll"); }`
@AfterAll	Executes only once per class after all tests executed. This method should be static. `@AfterAll public static void afterAll() { //disconnect db System.out.println("afterAll"); }`
@BeforeEach	Executes before each method `@BeforeEach public void beforeEach() { System.out.println("BeforeEach Test"); }`
@AfterEach	`@AfterEach public void afterEach() { System.out.println("afterEach Test"); }` Executes after each method
@DisplayName	Can be used for test method and test class to provide meaningful test description
@Disabled	Disable the test, it will not be executed.