JUNIT 5

Junit 5 i used to wirte test and run test manually , but **maven surefire** is a test runner , that automatically runs all your test written in any framework like junit 5 or Testing

---

How to setup env for junit 5 ?

1) Create a maven Project
2) Add Junit 5 Dependency in pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">


    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>junit5-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.8.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.8.2</version>
            <scope>test</scope>
        </d>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M5</version>
            </plugin>
        </plugins>
    </build>
</project>>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M5</version>
            </plugin>
        </plugins>
    </build>
</project>Screenshot from 2024-05-23 11-23-14
```

3) Solve any error / bugs
4) Import Libraries

import org.junit.jupiter.api.Test;

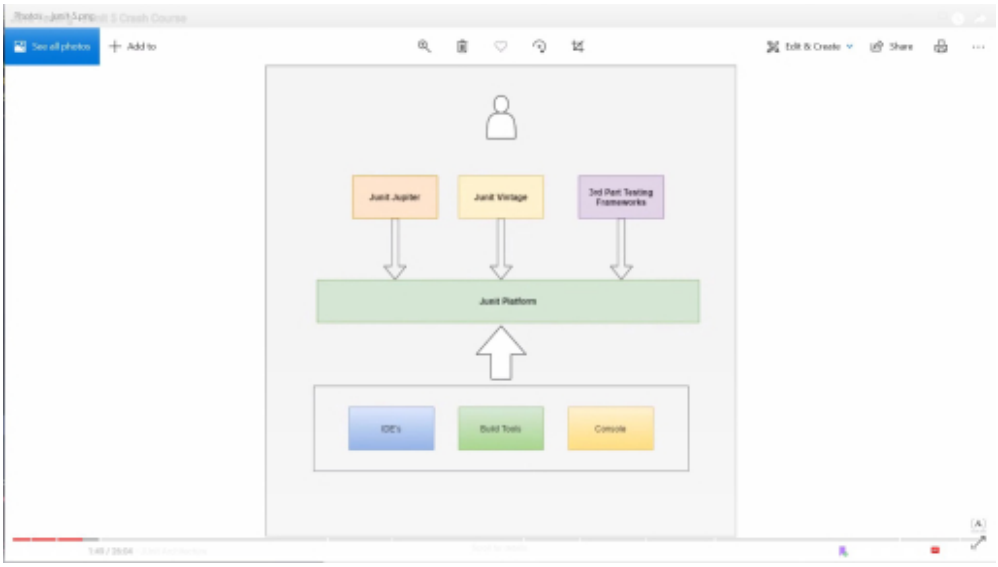
import static org.junit.jupiter.api.Assertions.assertEquals;

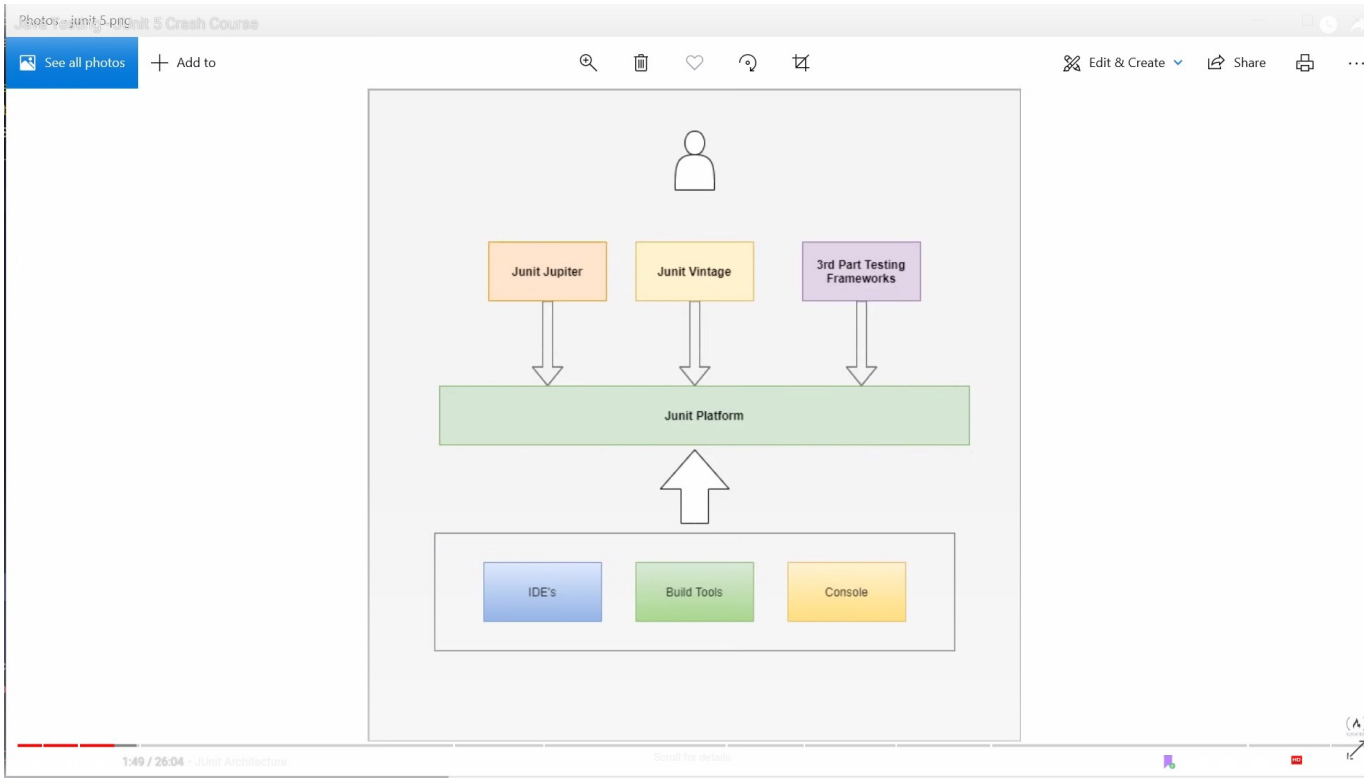
5) Write test with @Test

6) Run using mvn test

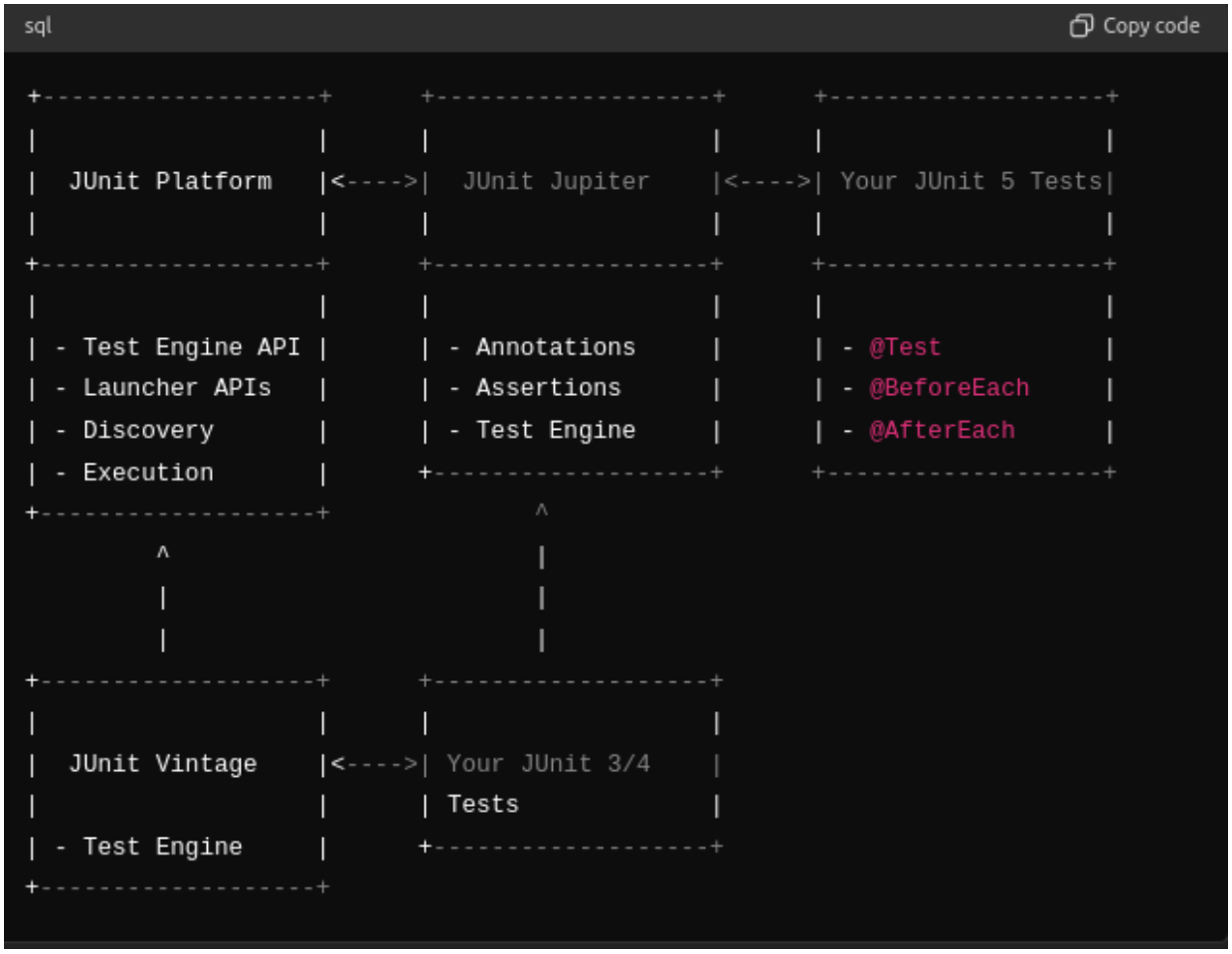
Libraries and Module of Junit5

- **junit-jupiter-api:** Essential for writing JUnit 5 tesets.
- **junit-jupiter-engine:** Essential for running JUnit 5 tests.
- **junit-platform-suite:** Optional, for creating and running test suites.
- **junit-jupiter-params:** Optional, for writing parameterized tests





Architecture of Junit 5



JUnit Platform

Test Discovery: Identifies test classes and methods.
Test Execution: Manages the execution of tests and aggregates results.
Launcher APIs: Allows integration with IDEs and build tools.
Console Launcher: Enables running tests from the command line.

JUnit Jupiter

Annotations: Provides new annotations such as `@Test`, `@ParameterizedTest`, `@BeforeEach`, `@AfterEach`, `@Nested`, `@Tag`, `@DisplayName`, etc.
Assertions: Offers various assertions for testing conditions, including `assertEquals`, `assertTrue`, `assertThrows`, etc.
Extensions: Introduces an extensible model for adding custom behavior via extensions (e.g., `@ExtendWith`).
Dynamic Testing: Allows for the generation of tests at runtime using `DynamicTest` and `@TestFactory`.
e JUnit Platform to run older tests

JUnit Vintage

.Backward Compatibility: Enables running tests written in JUnit 3 and 4.
Test Engine:
Integrates with the JUnit Platform to run older tests.

- **JUnit Platform:** The foundation that supports running tests and integrating with tools.
- **JUnit Jupiter:** The new programming and extension model for writing tests in JUnit 5.
- **JUnit Vintage:** Provides backward compatibility to run JUnit 3 and 4 tests on the JUnit 5 platform.

Methods of Junit

assertTrue , assertFalse, assertEquals

Annotations

1. **`@TestInstance(TestInstance.Lifecycle.PER_CLASS)`**: Ensures that `@BeforeAll` and `@AfterAll` methods can be non-static.
   - **Use Case**: Used to initialize and cleanup resources once for the entire test class, rather than for each test method.
2. **`@BeforeAll`**: Runs once before all test methods**. (Static method)**
   - **Use Case**: Initialize resources that are shared across all tests, such as setting up a database connection.
3. **`@BeforeEach`**: Runs before each test method.
   - **Use Case**: Setup the initial state for each test, such as creating a new instance of `AccountService` and setting up an initial account.
4. **`@Test`**: Marks methods as test methods.
   - **Use Case**: Tests for creating accounts, depositing money, withdrawing money, and checking for sufficient funds.
5. **`@DisplayName`**: Provides a custom display name for test methods.
   - **Use Case**: Makes the test reports more readable and descriptive.
6. **`@ParameterizedTest`**: Marks a method as a parameterized test.
   - **Use Case**: Tests the deposit functionality with multiple amounts using the same test method.
7. **`@ValueSource`**: Provides data for the parameterized test.
   - **Use Case**: Supplies different deposit amounts to the parameterized test.
8. **`@Disabled`**: Disables a test method.
   - **Use Case**: Temporarily disable the negative deposit test until the related issue is fixed.
9. **`@AfterEach`**: Runs after each test method.
   - **Use Case**: Cleanup after each test, such as clearing all accounts to ensure a clean state for the next test.
10. **`@AfterAll`**: Runs once after all test methods**.(Static Method)**
    - **Use Case**: Cleanup resources that were initialized in `@BeforeAll`, like closing a database connection.
11. **@TestInstance(TestInstance.Lifecycle.PER_CLASS) or PER_METHOD**

The `@TestInstance(TestInstance.Lifecycle.PER_CLASS)` annotation in JUnit 5 is used to control how many instances of your test class are created during testing.
By default, JUnit creates a new instance of your test class for every single test method. This is known as the `PER_METHOD` lifecycle.
The `@TestInstance(TestInstance.Lifecycle.PER_CLASS)` annotation tells JUnit to **create eonly one instance of the test class for the entire class**. This instance is then reused for all the test methods within that class

---

Assumptions

Purpose: **Skip tests** under certain conditions to avoid false negatives.
Key Annotations: `assumeTrue`, `assumeFalse`, `assumingThat(condition,statementToExce)`
Usage Scenarios: Environment-specific tests, OS-specific tests, conditional resource checks.

```
@Test
void testOnlyOnCiServer() {          Assumptions.assumeTrue("CI".equals(System.getenv("ENV")));
 }
@Test
void testOnlyOnDeveloperMachine() {
Assumptions.assumeTrue("DEV".equals(System.getenv("ENV")));}
```

When an assumption is used, the condition is checked at the beginning of the test. If the condition is not met, the test is skipped.This helps to avoid running tests under conditions where they are not supposed to run, thereby avoiding false negatives.

@RepeatedTest

Repeated tests allow a **single test method to be executed multiple times.**

```
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.RepetitionInfo;import static org.junit.jupiter.api.Assertions.assertTrue;
public class RepeatedTestWithInfoExample {
@RepeatedTest(5,name=RepeatedTest.LONG_DISPLAY_NAME)
void repeatedTestWithInfo(RepetitionInfo repetitionInfo) {
int currentRepetition = repetitionInfo.getCurrentRepetition(); int totalRepetitions =
repetitionInfo.getTotalRepetitions(); System.out.println("Repetition " + currentRepetition + " of " +
totalRepetitions);
// Test logic here assertTrue(Math.random() > 0.1); }
}
```