Exploring Core Junit

# Core Concepts

## Test Class

*A test class* may be **a top-level class**, **a static member class**, or **an inner class annotated as @Nested** that contains one or more test methods.

**Test classes cannot be abstract.**

**They must have a single constructor**. **The constructor must have no arguments, or arguments that can be dynamically resolved at runtime through dependency injection.** (We discuss the details of dependency injection in section 2.6.)

A test class is allowed to be package-private as a minimum requirement for visibility. It is no longer required that test classes be public, as was the case up to JUnit 4.x.

## Test Method

*A test method* is an instance method that is annotated with @Test, @Repeated-Test, @ParameterizedTest, @TestFactory, or @TestTemplate.

Test methods must not be abstract **and must not return a value** (the return type should be void).

## Life-Cycle Method

*A life cycle method* is a method that is annotated with @BeforeAll, @AfterAll, @BeforeEach, or @AfterEach.

## How Everything works together

* JUnit creates **a new instance of the test class** before invoking each @Test method to ensure the independence of test methods and prevent unintentional side effects in the test code.
* it is a universally accepted fact **that the tests must produce the same result independent of the order of their execution.**
* Because each test method runs on a new test class instance, you cannot reuse instance variable values across test methods. One test instance is created for the execution of each test method, which is **the default behavior in JUnit 5 and all previous versions**.
* If you annotate your **test class** with @TestInstance(Lifecycle.PER\_CLASS)as opposed to the default of Lifecycle.PER\_METHOD, JUnit 5 will execute all test methods on the same test instance. A new test instance will be created for each test class when using this annotation.

Page 18…