Unit Testing Tool: JUnit 5 for Android Kotlin

By Choyon Sarker

October 2024

1. What is JUnit 5?

JUnit 5 is a modern and flexible testing framework used for writing unit, integration, and system tests in Java and Kotlin. It is particularly well-suited for Android development due to its modular design, advanced features, and seamless integration with Kotlin, the preferred language for modern Android apps.

Key Features:

- **Modular Architecture:** JUnit 5 consists of three main modules: JUnit Platform, JUnit Jupiter, and JUnit Vintage.
- **Declarative Annotations:** Simplifies test writing through the use of annotations such as @Test, @BeforeEach, and @AfterEach, making test code easier to write and understand.
- **Lambda Support:** JUnit 5 integrates well with Kotlin's functional programming style and supports lambdas.

2. Why Use JUnit 5 for Android Kotlin Projects?

JUnit 5 offers several advantages when developing Android apps in Kotlin:

Key Advantages:

- **Kotlin-Friendly:** JUnit 5 naturally integrates with Kotlin, allowing developers to take advantage of Kotlin's concise syntax and powerful features.
- **Comprehensive Testing:** It supports unit testing, integration testing, and behavior-driven testing.
- **Improved Test Readability:** JUnit 5 allows for writing expressive, readable, and maintainable test cases.
- **Flexibility:** JUnit 5's modular architecture lets you include only the necessary components, improving performance.

To add JUnit 5 to an Android project, include the following dependency:

testImplementation "org.junit.jupiter:junit-jupiter:5.8.1"

3. Writing Unit Tests for Arithmetic Operations in Kotlin

Unit tests validate individual units of code, ensuring each method or class behaves as expected. Here's an example of testing an addition operation.

3.1. Calculator Class Example

This is the `Calculator` class that performs addition:

```
class Calculator {
    fun add(a: Int, b: Int): Int {
        return a + b
    }
}
```

3.2. Writing Unit Tests for the Addition Method

We'll write unit tests to ensure the `add` method works for different scenarios, including positive numbers, negative numbers, and zero.

```
import org.junit.jupiter.api.Assertions.assertEquals
import org.junit.jupiter.api.Test

class CalculatorTest {

    private val calculator = Calculator()

    @Test
    fun `should return sum of two positive numbers`() {
        val result = calculator.add(5, 3)
        assertEquals(8, result, "5 + 3 should equal 8")
    }

    @Test
    fun `should return sum of positive and negative number`() {
        val result = calculator.add(5, -2)
        assertEquals(3, result, "5 + (-2) should equal 3")
    }

    @Test
    fun `should return sum of two negative numbers`() {
        val result = calculator.add(-4, -6)
        assertEquals(-10, result, "-4 + (-6) should equal -10")
    }

    @Test
```

```
    fun `should return correct sum when adding zero`() {
        val result = calculator.add(7, 0)
        assertEquals(7, result, "7 + 0 should equal 7")
    }

}
```

4. Assertions and Assumptions in JUnit 5

JUnit 5 provides various assertions to verify the correctness of tests. Some of the most commonly used assertions are:

- assertEquals(expected, actual, message): Ensures the expected result matches the actual value.
- assertTrue(condition, message): Asserts that the condition is true.
- assertThrows(exceptionClass, executable): Verifies that a specific exception is thrown during execution.

5. Parameterized Tests

JUnit 5 supports parameterized tests, enabling the execution of the same test with multiple sets of inputs.

Here's an example of a parameterized test for the `add` method:

```
import org.junit.jupiter.params.ParameterizedTest
import org.junit.jupiter.params.provider.CsvSource
import org.junit.jupiter.api.Assertions.assertEquals

class ParameterizedCalculatorTest {

    private val calculator = Calculator()

    @ParameterizedTest
    @CsvSource(
        "5, 3, 8",
        "-4, 6, 2",
        "0, 0, 0"
    )
    fun `should return correct sum for various inputs`(a: Int, b: Int, expected: Int) {
        assertEquals(expected, calculator.add(a, b))
    }
```

}

6. How to Integrate JUnit 5 with Android

To integrate JUnit 5 with Android, you need to configure your project properly since Android projects use a different environment compared to standard Java projects. While JUnit 4 is natively supported by Android Studio, JUnit 5 requires some additional configuration steps.

Steps to Integrate JUnit 5 with Android Projects:

6.1. Modify Gradle Files

You need to add the necessary dependencies in your build.gradle files. The key is to ensure that the Android project uses the JUnit 5 test engine.

1. **Add JUnit 5 dependencies in the `build.gradle` file**:

Add the following dependencies in your **app-level** build.gradle file to include JUnit 5 and its platform runner for Android:

```
// Enable JUnit 5
testImplementation "org.junit.jupiter:junit-jupiter-api:5.8.1"
testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine:5.8.1"

// Support JUnit 5 on the Android platform
testImplementation "de.mannodermaus.junit5:android-test-core:1.3.0"
testRuntimeOnly "de.mannodermaus.junit5:android-test-runner:1.3.0"

// Mocking library for unit testing (optional, if needed)
testImplementation "org.mockito:mockito-core:3.11.2"
```

These dependencies include:

- `junit-jupiter-api`: The main JUnit 5 API to write unit tests.
- `junit-jupiter-engine`: The engine that runs JUnit 5 tests.

- `android-test-core` and `android-test-runner`: These enable the Android platform to run JUnit 5 tests.

1. **Update the test runner**:

In your **module-level** build.gradle file, configure the test options to run JUnit 5 with Android:

```
android {
    testOptions {
        unitTests.includeAndroidResources = true
    }
}
```

This ensures that your unit tests have access to Android resources like Context.

6.2. Running JUnit 5 Tests in Android Studio

Once the dependencies are set up, you can run JUnit 5 tests directly in **Android Studio**. Simply right-click on a test class or method and select **Run**. You can also run tests via Gradle from the terminal with the following command:

```
./gradlew test
```

This command runs all unit tests in the project. If you only want to run tests from a specific package, use:

```
./gradlew test --tests "com.example.yourpackage.*"
```

6.3. Continuous Integration (CI) Support

JUnit 5 integrates smoothly with CI/CD pipelines like **Jenkins**, **CircleCI**, and **GitHub Actions**. Ensure that your CI/CD configuration runs ./gradlew test as part of your build process to automatically execute tests after each code push.

7. Why Choose JUnit 5 for Android Kotlin Testing?

JUnit 5 offers a comprehensive, flexible, and powerful testing solution for Android Kotlin projects. Its features allow developers to write maintainable and scalable tests, with full integration into modern development workflows.

8. References

1. JUnit 5 User Guide: https://junit.org/junit5/docs/current/user-guide/

2. Testing Kotlin with JUnit 5: https://www.jetbrains.com/help/idea/testing-kotlin.html

3. Unit Testing in Android with JUnit 5:
https://developer.android.com/training/testing/fundamentals