



# Automated Testing for Android with Kotlin





Somkiat Puisungnoen

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชั่นนาฎกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชั่นนาฎกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata



Facebook somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc  
@somkiat.cc

Home Posts Videos Photos

Liked Following Share ... + Add a Button



**<https://github.com/up1/course-automated-test-android-kotlin>**



# Agenda day 1

Introduction of testing

Why we need to test ?

Types of tests

Testing pyramid concept

Android testing

Code coverage

Workshop (step-by-step)



# Agenda day 2

Espresso workshop

Testable application

Working with REST APIs (Networking)

Continuous Integration

Working with fastlane



# Testing for Android app



**"Program testing can be used  
to show the presence of bugs,  
but never their absence."**

Edsger Dykstra, 1970, Notes on Structured Programming



# Why we need to test ?

Help you to catch bugs

Develop features faster

Enforce modularity of your project



**But,  
It's take time to learning and  
practice !!**



# Goals

How to **THINK** when and where  
you should test



# What you need to know ?

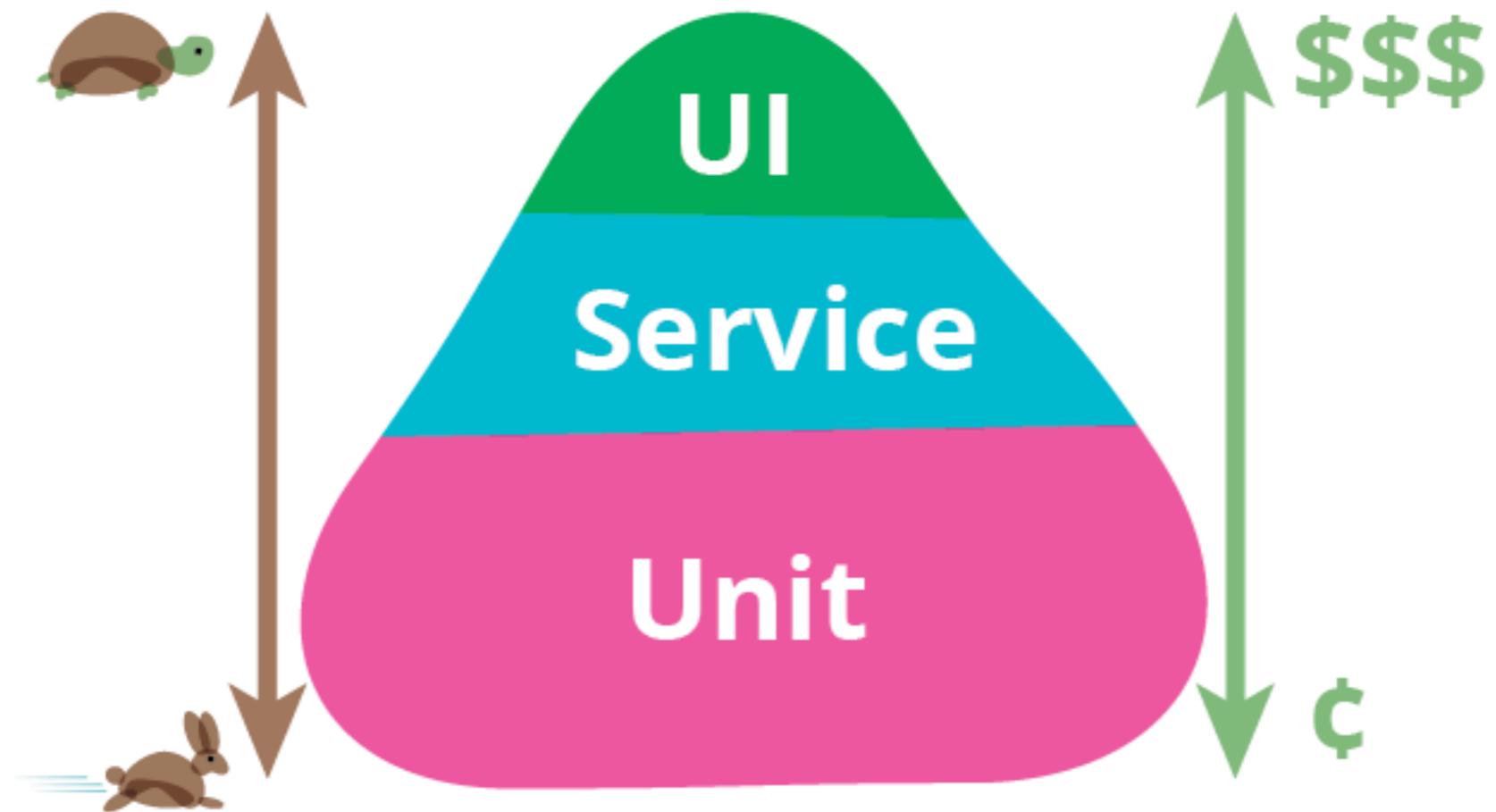
Android/Kotlin  
Android Studio  
**JUnit 4**  
**Espresso**



# Type of testing



# Testing Pyramid



<https://martinfowler.com/bliki/TestPyramid.html>



# Android Testing

Android



# Android Testing

Android

Java



# Java run on JVM



JVM (Java Virtual Machine)



# Run android need device



Build app -> Install to device -> Test



# Android Testing

JVM

Device

JVM unit test

/src/test



*Business logic with pure java code*



# Android Testing

JVM

Device

JVM unit test

Instrumentation unit  
test

/src/test

/src/androidTest

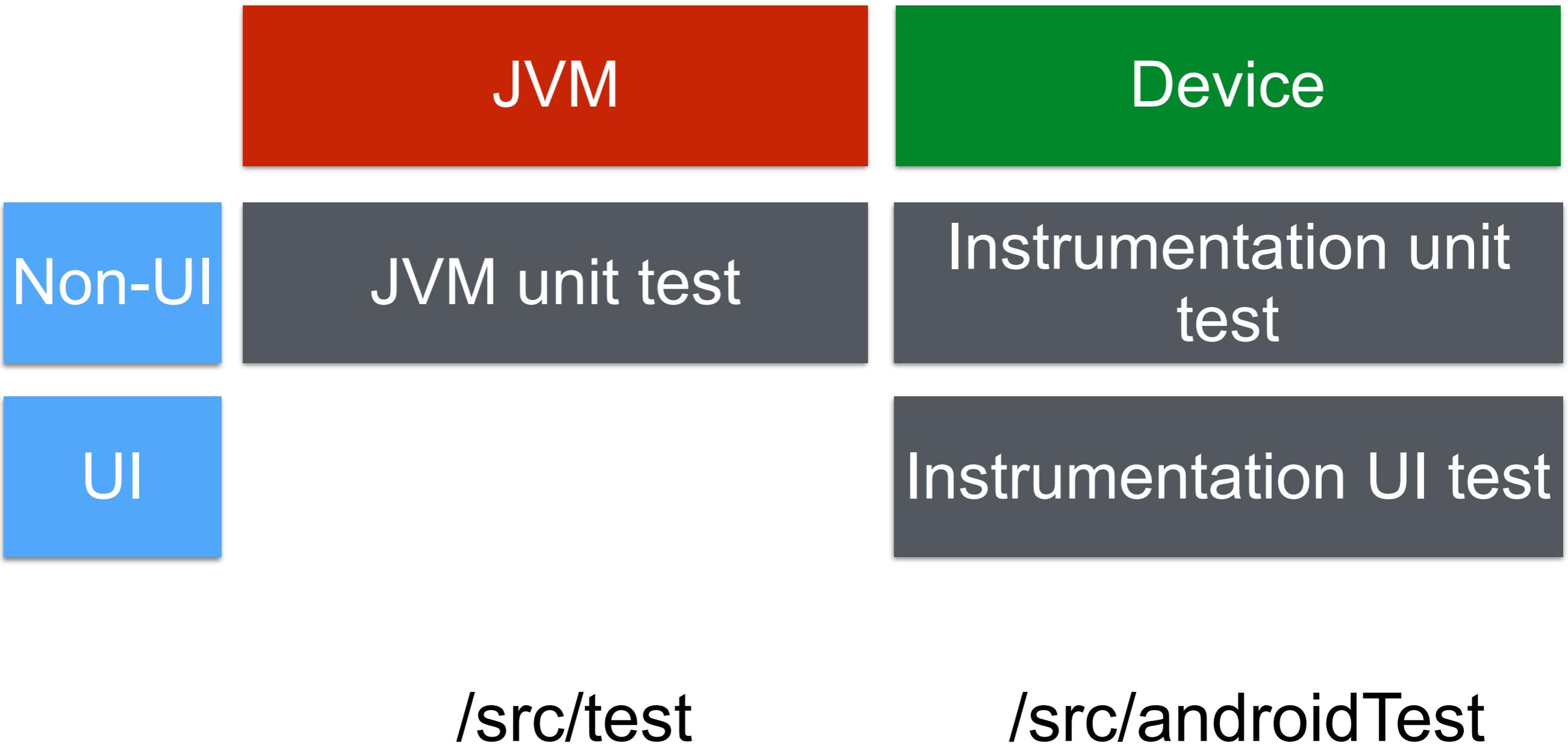
*Working with Android specific code, you need run on device such as AssetManager, SharedPreference*



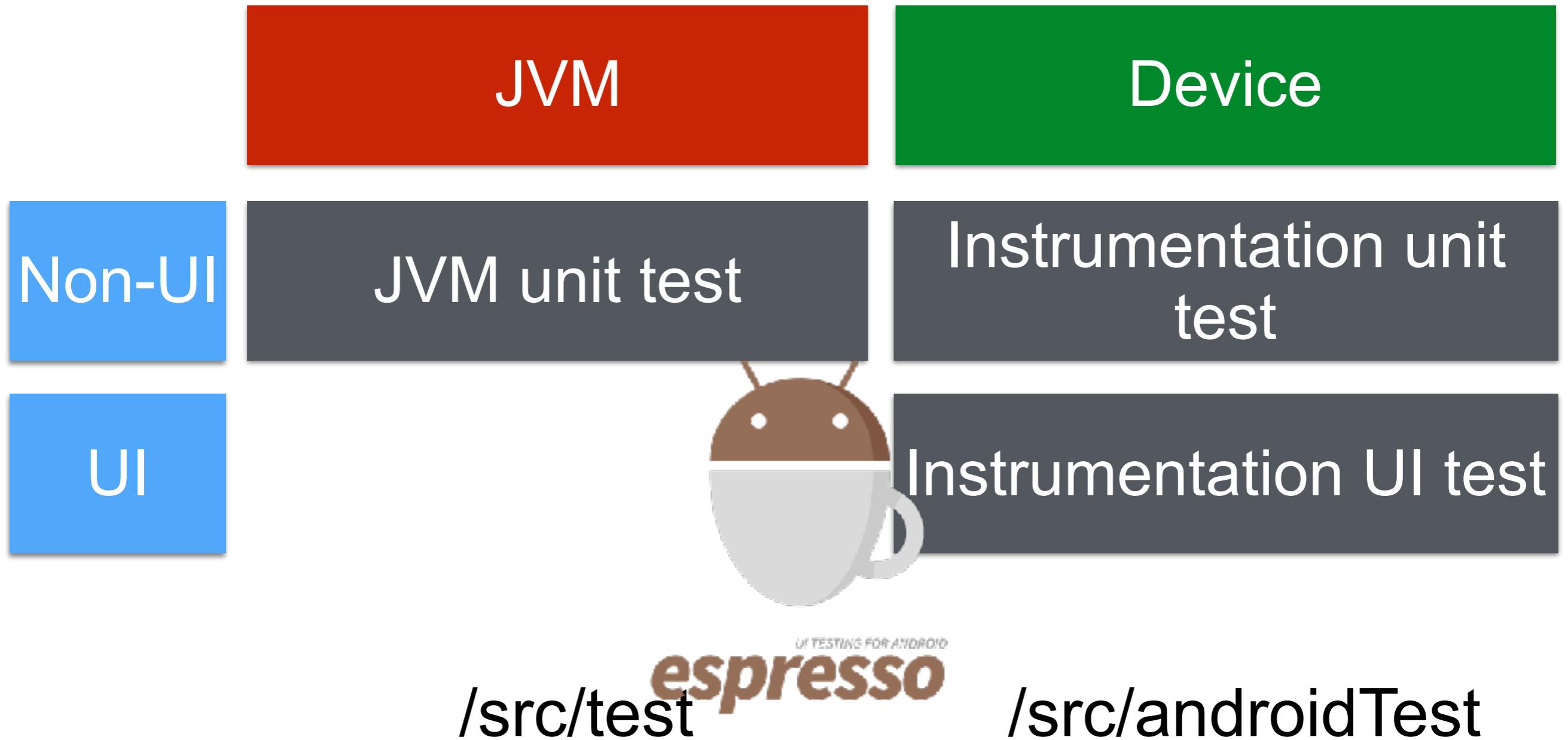
# UI vs Non-UI



# Android Testing



# Android Testing



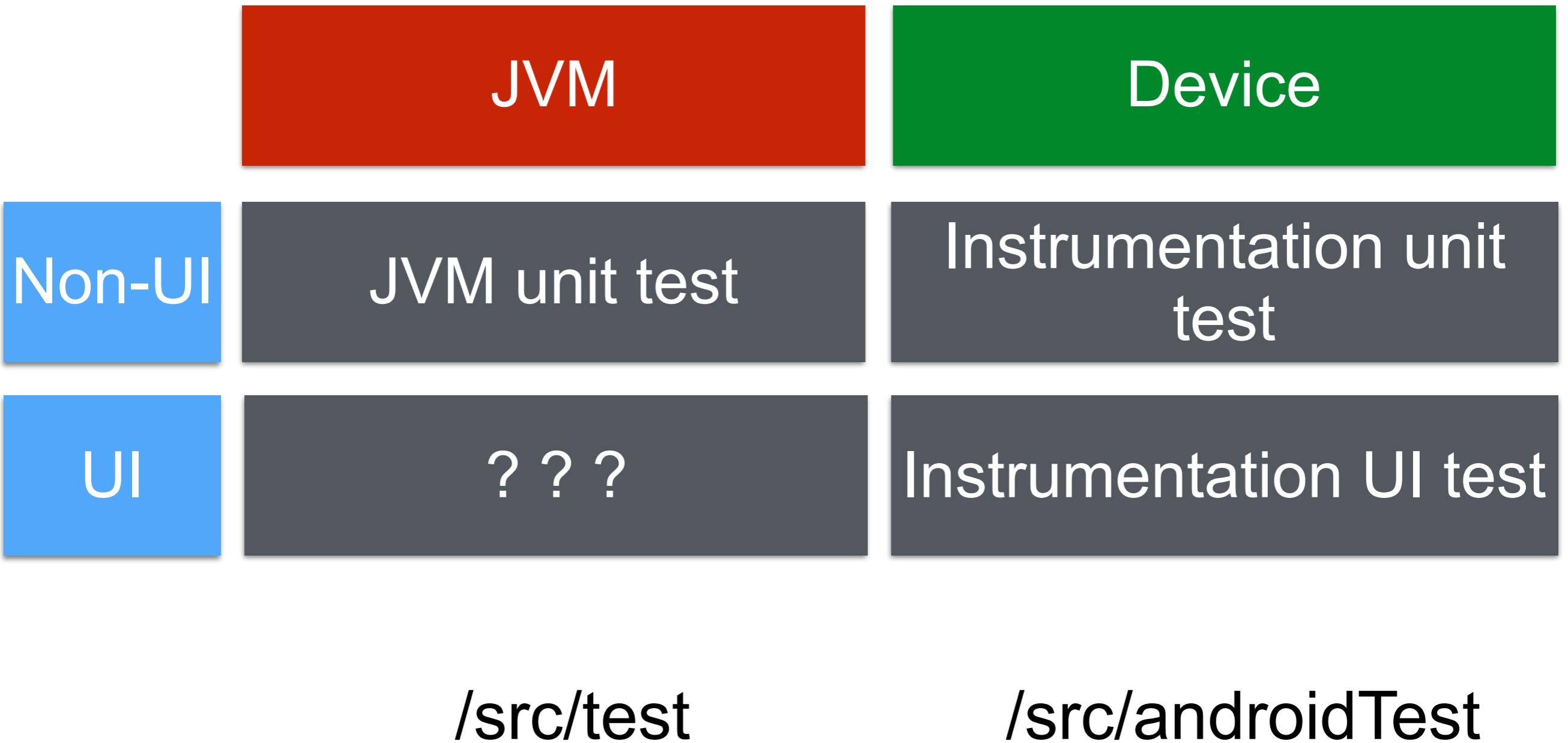
# Resources

<https://developer.android.com/studio/test/index.html>

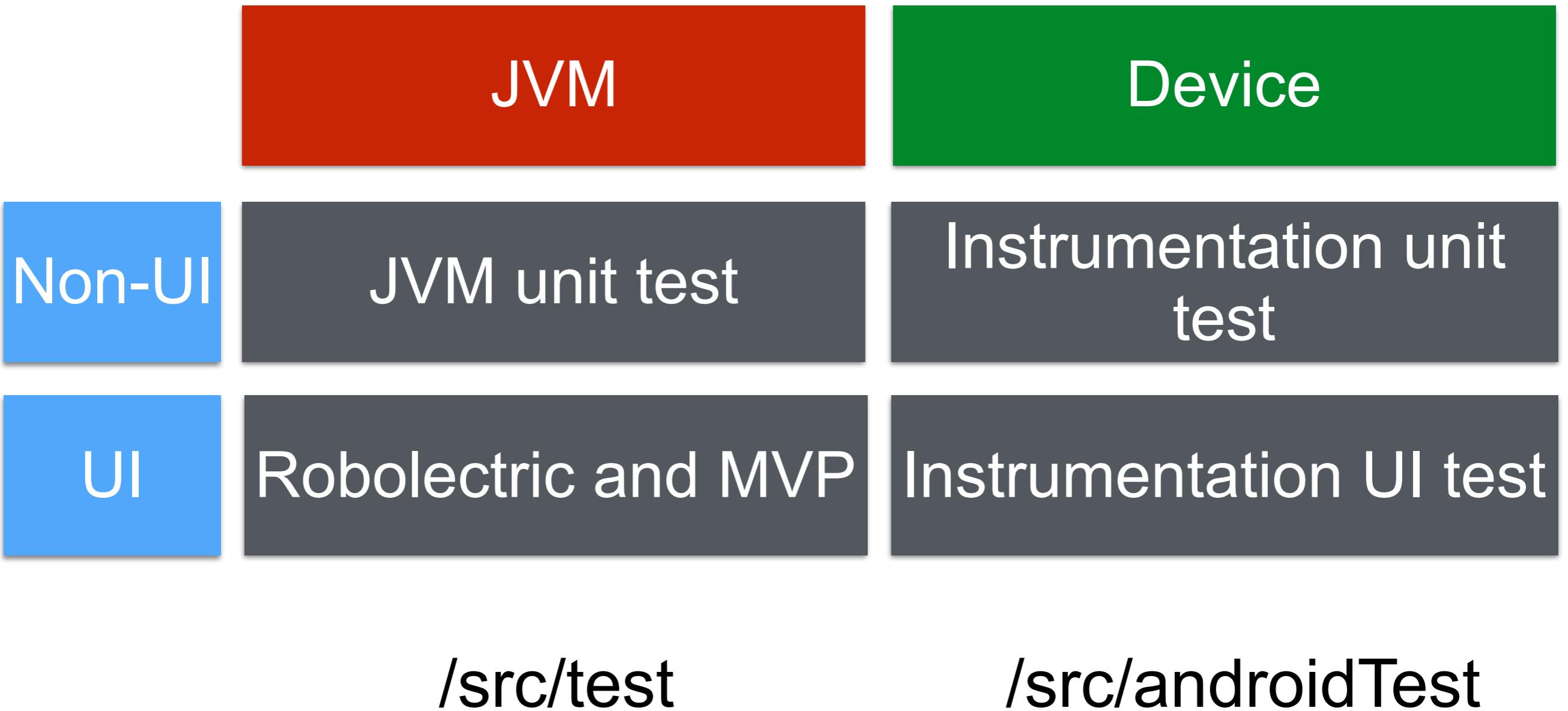
<https://developer.android.com/topic/libraries/testing-support-library/index.html#Espresso>



# Android Testing



# Android Testing



# Rule of thumb

Instrumentation tests are **slower** than JVM tests.

Try to separate the standard Java code from  
Android-dependent code.



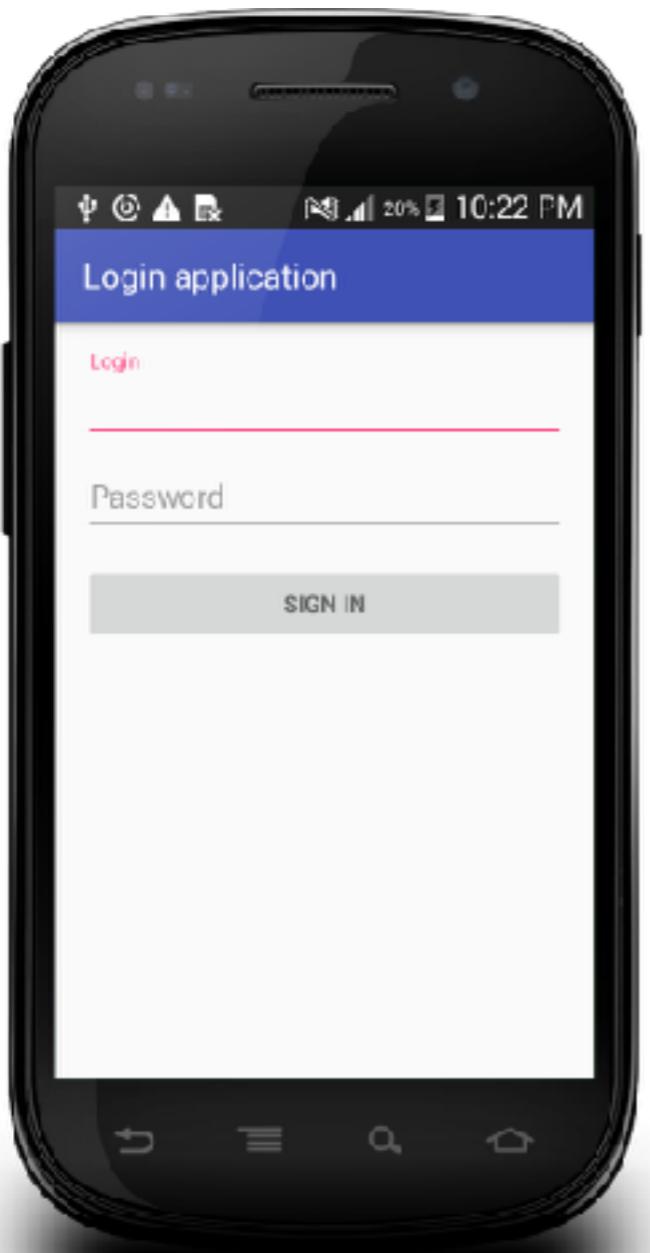
# Workshop with Testing

## Hello Testing with Kotlin



# Login application

Login Page



Result

Success

Failure



# Test cases

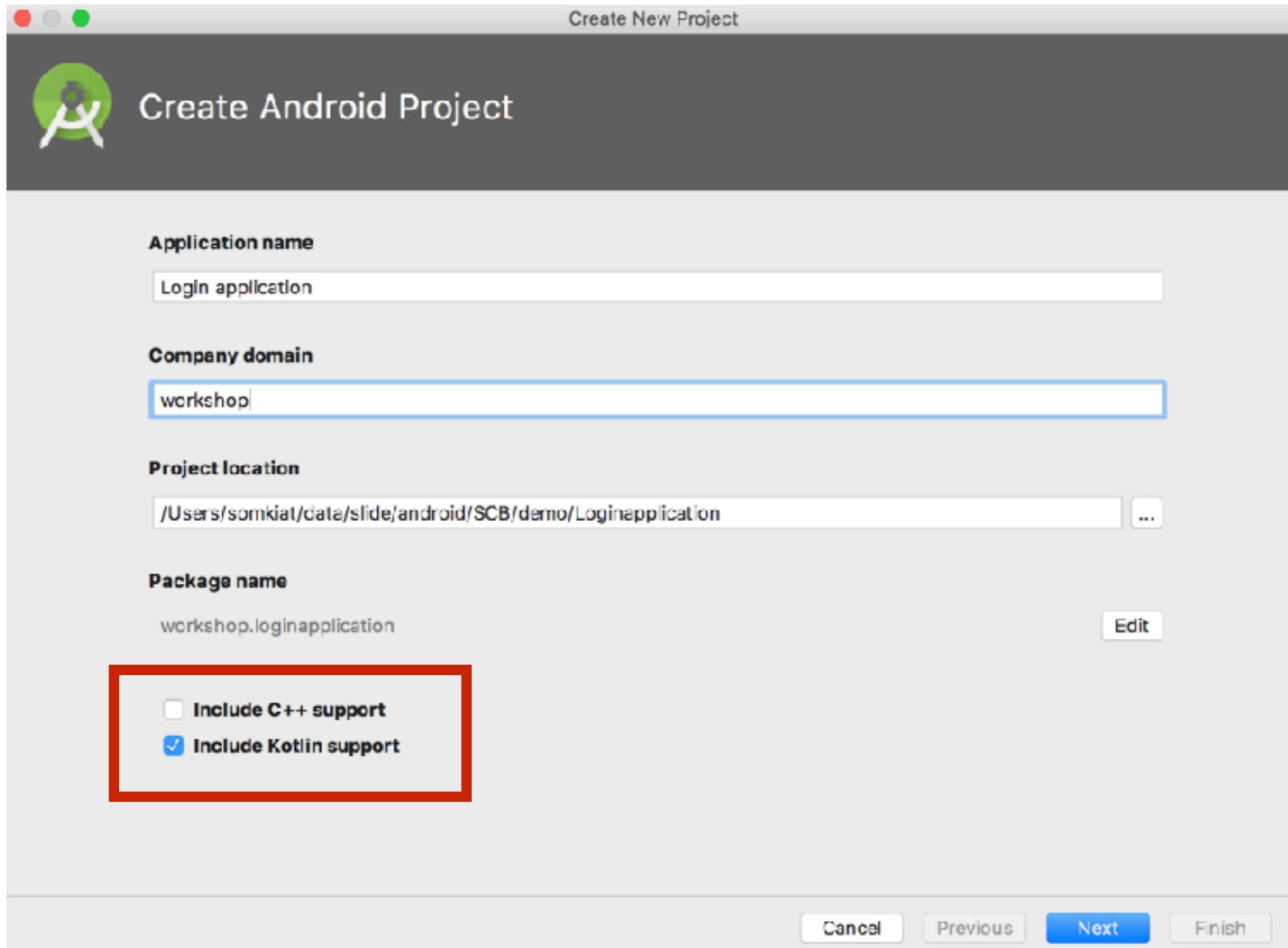
Test Case Name	Email	Password	Expected Result
<b>Login success</b>	somkiat@xxx.com		Show <b>success</b> message
Login fail			Show <b>fail</b> message



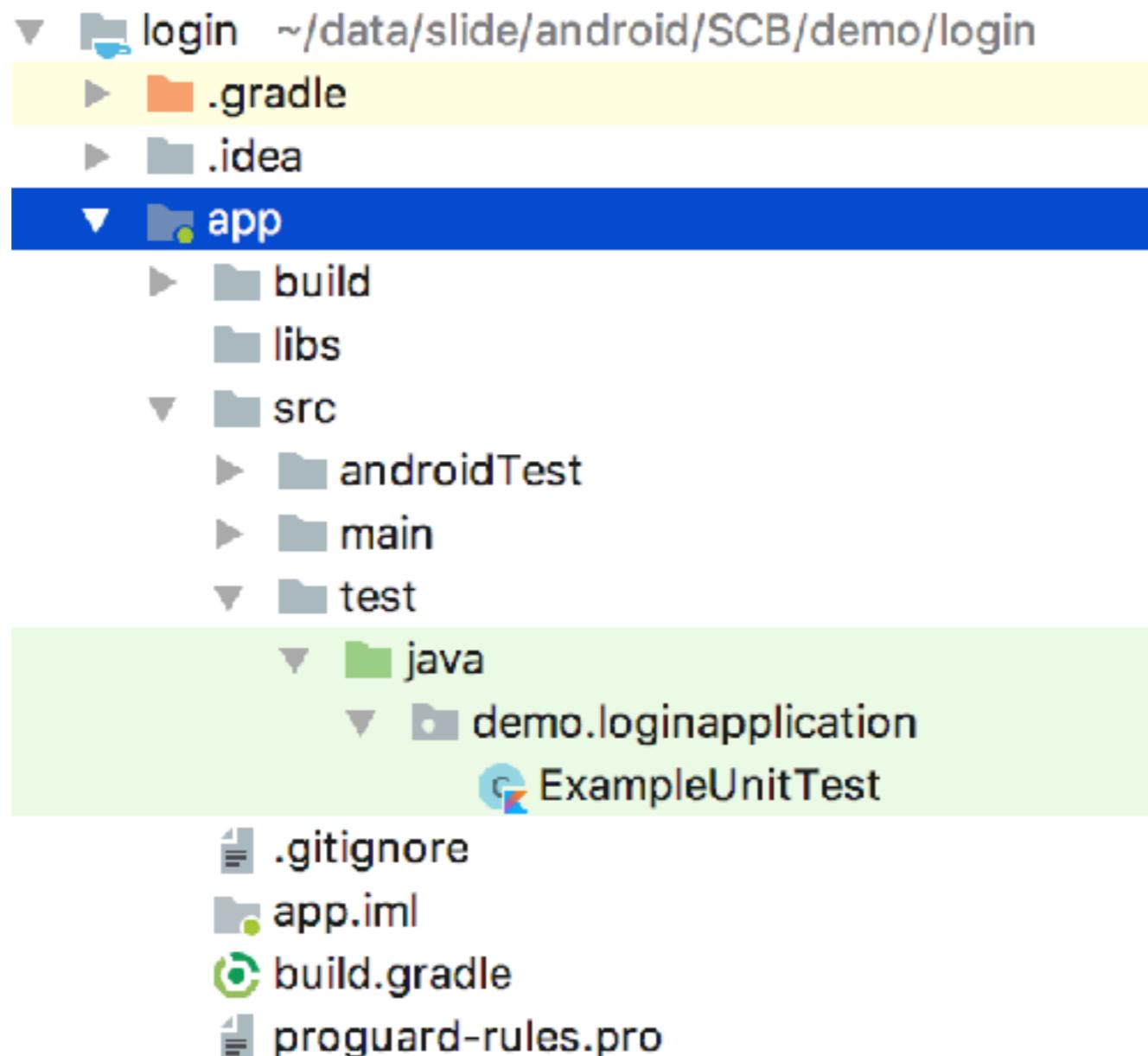
# Let's start to write tests



# 1. Create project with Kotlin



# 2. Project structure



# 3. Supported testing by default

Open file /app/build.gradle

```
android {  
    compileSdkVersion 27  
    defaultConfig {  
        applicationId "demo.loginapplication"  
        minSdkVersion 15  
        targetSdkVersion 27  
        versionCode 1  
        versionName "1.0"  
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
    }  
}
```



# 3. Supported testing by default

Open file /app/build.gradle

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"  
    implementation 'com.android.support:appcompat-v7:27.1.1'  
    implementation 'com.android.support:design:27.1.1'  
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'  
  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'com.android.support.test:runner:1.0.2'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'  
}
```



JUnit

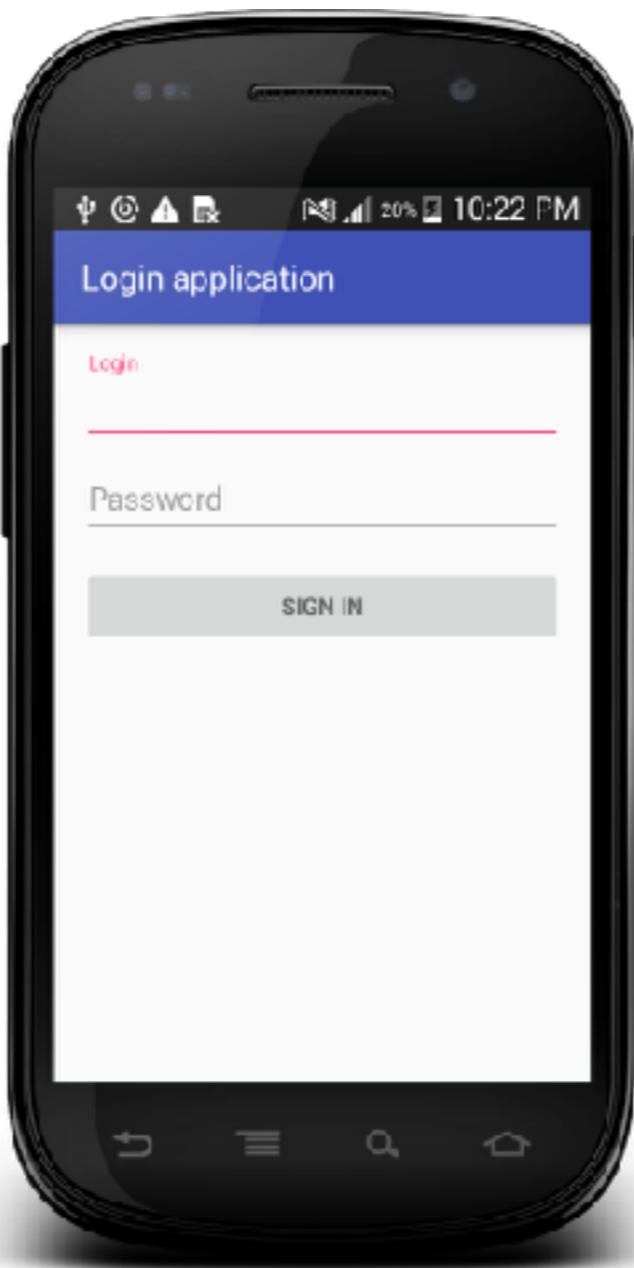


# 4. Develop login application

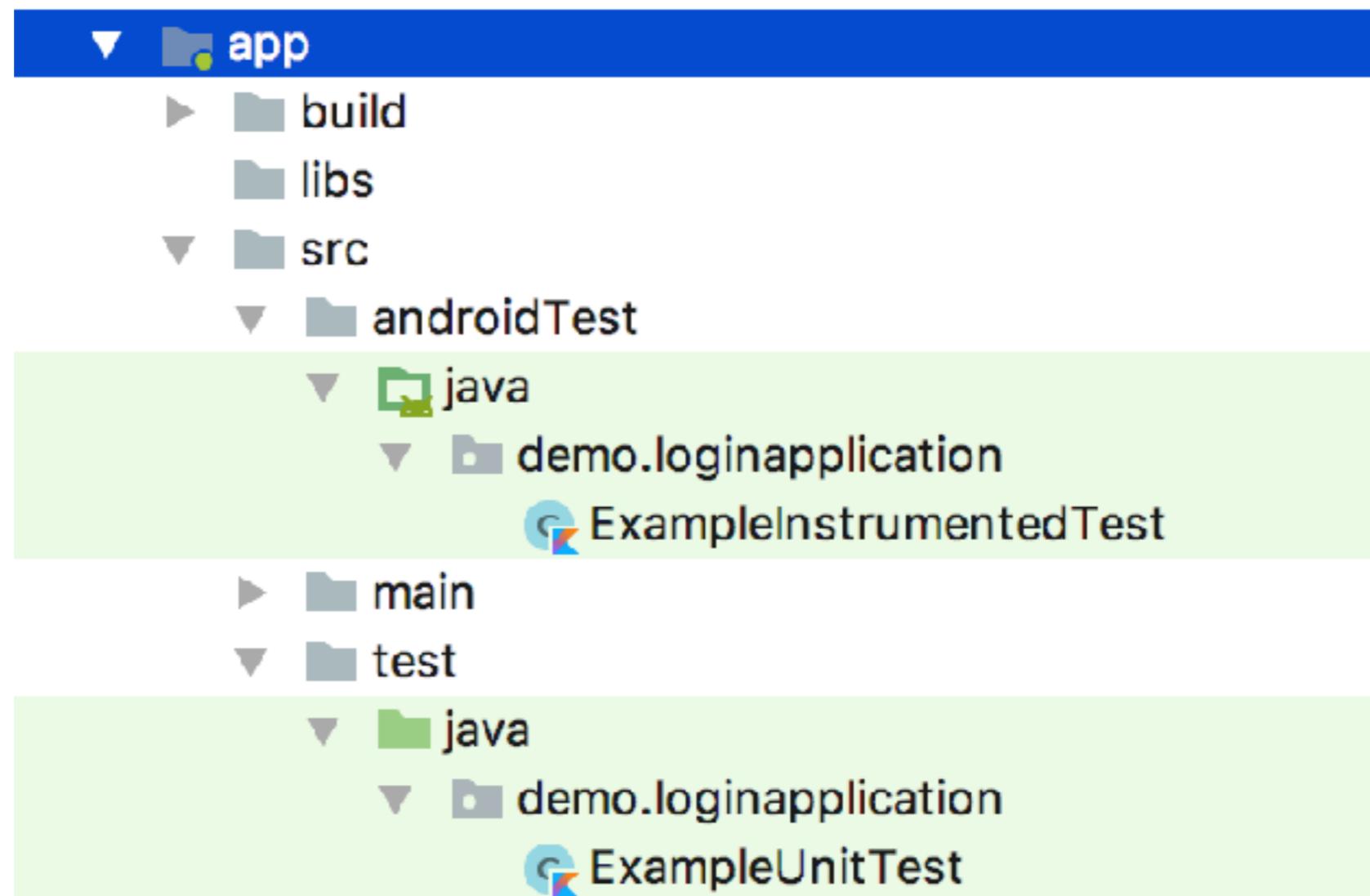
Import from Demo project



# 5. Run app

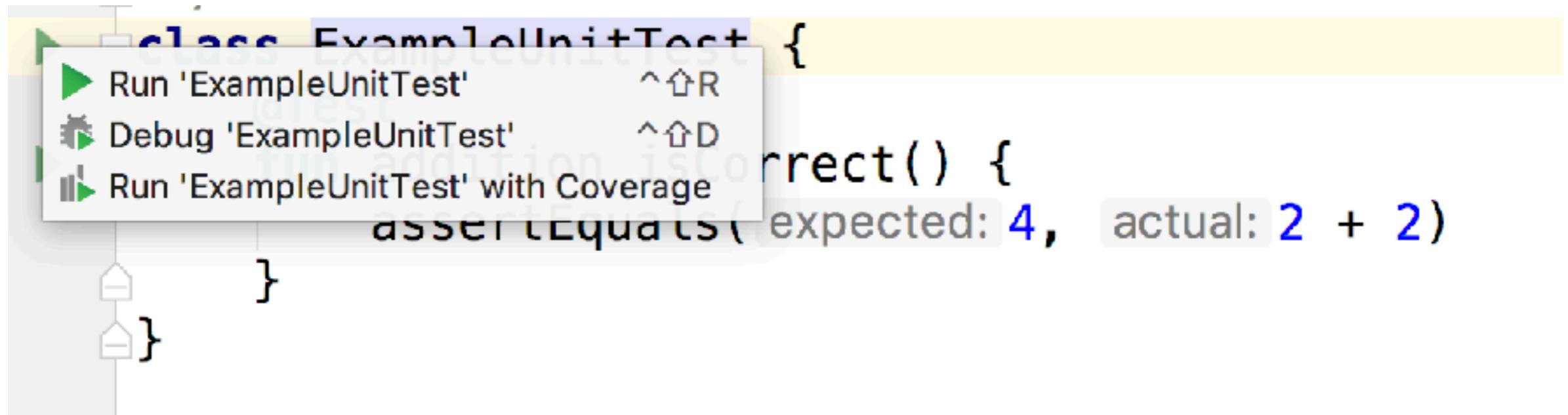


# 6. Structure of tests



# 7. Run tests

\$gradlew test



A screenshot of the Android Studio code editor. A context menu is open over the code for a class named `ExampleUnitTest`. The menu items are: "Run 'ExampleUnitTest'" (with keyboard shortcut `^R`), "Debug 'ExampleUnitTest'" (with keyboard shortcut `^D`), and "Run 'ExampleUnitTest' with Coverage". The code itself shows a `correct()` method containing an `assertEquals` assertion. The assertion line is highlighted with a yellow background.

```
class ExampleUnitTest {
    // ...
    correct() {
        assertEquals(4, 2 + 2)
    }
}
```



# 8. See test result

/app/build/reports/tests/testDebugUnitTest

## Test Summary



Packages

Classes

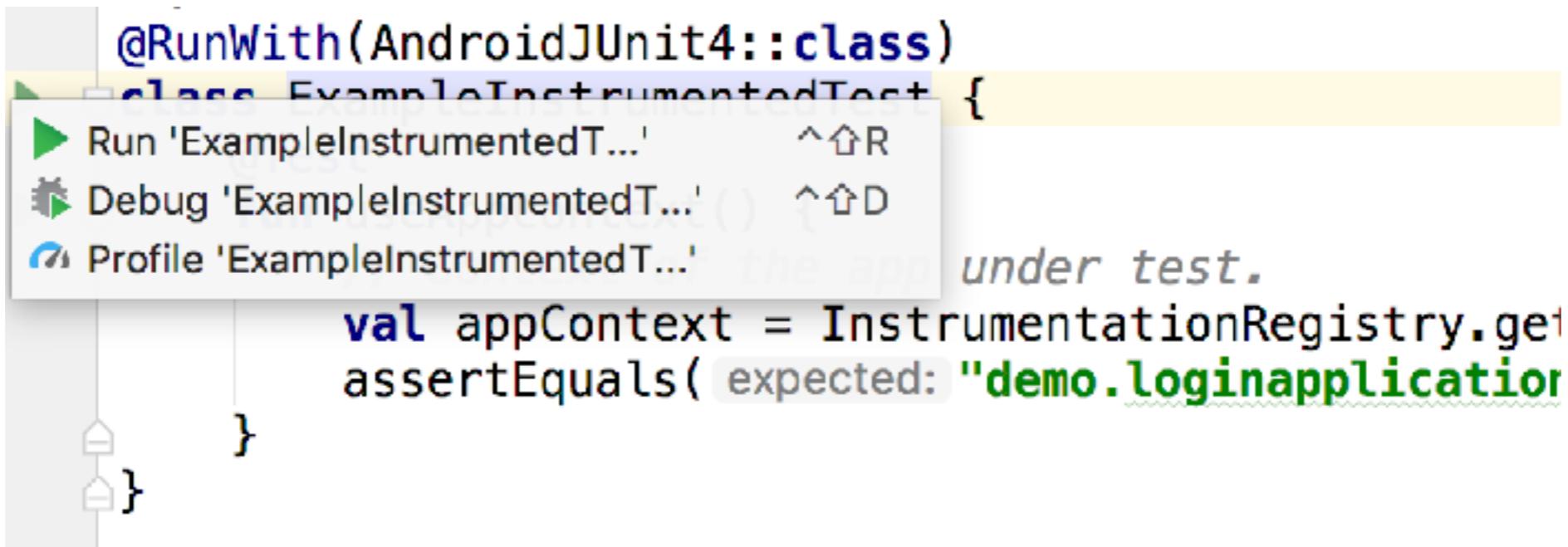
Package	Tests	Failures	Ignored	Duration	Success rate
<a href="#">demo.log application</a>	1	0	0	0.001s	100%



# 9. Run androidTest

\$gradlew connectedAndroidTest

\$gradlew cAT



The screenshot shows an Android Studio code editor with a context menu open over a test class named `ExampleInstrumentedTest`. The menu options are:

- Run 'ExampleInstrumentedT...' (^R)
- Debug 'ExampleInstrumentedT...' (^D)
- Profile 'ExampleInstrumentedT...' (^P)

The text under the menu reads: "under test." Below the menu, the code for the test class is visible:

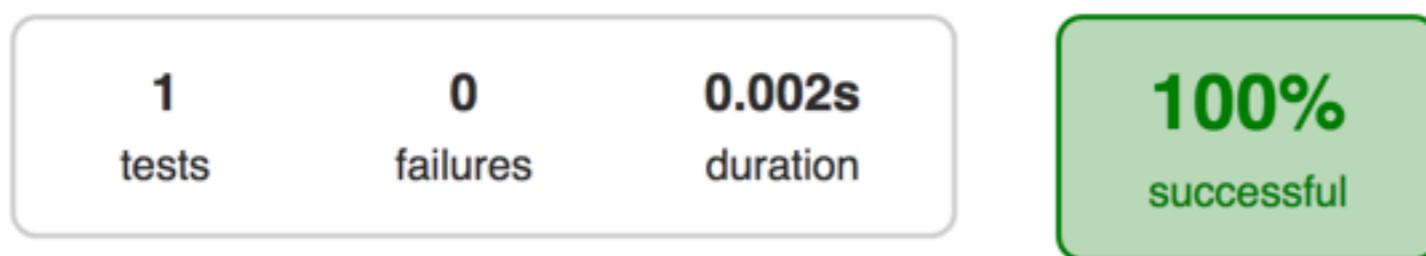
```
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    // Test code
}
```



# 10. See test result

/app/build/reports/androidTests/connected

## Test Summary



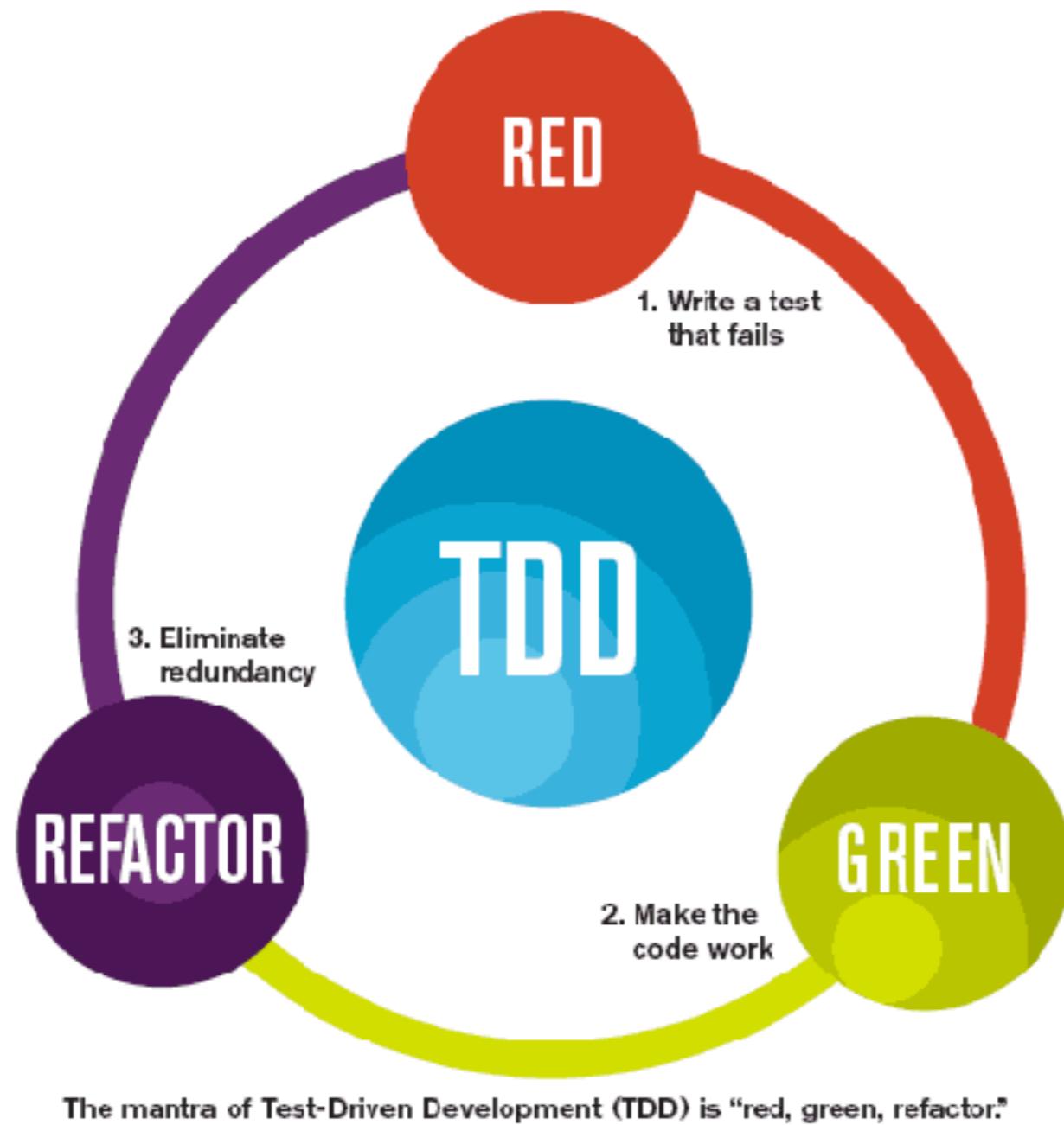
**Packages**      **Classes**

Package	Tests	Failures	Duration	Success rate
<a href="#">demo.loginapplication</a>	1	0	0.002s	100%



# 11. Write your first test case

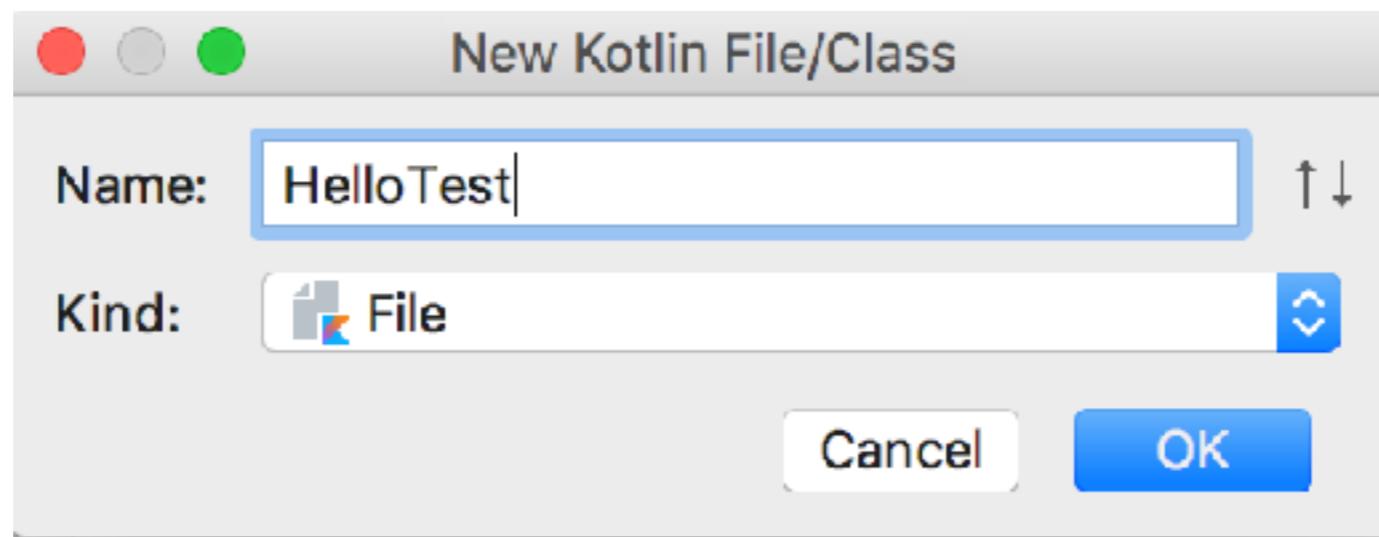
Unit test or Android Test ?



# 12. Business logic testing

Learn how to write unit test with Unit 4.0

Create new class in /app/tests/java



# Hello Test

```
class HelloTest {  
  
    @Test  
    fun `login success with correct email and password`() {  
  
    }  
  
}
```



# Test Structure

```
class HelloTest {  
  
    @Test  
    fun `login success with correct email and password`() {  
        // Arrange or Given  
  
        // Act or When  
  
        // Assert or Then  
    }  
  
}
```



# Design as a Test

```
@Test
fun `login success with correct email and password`() {
    // Arrange or Given
    val hello = Hello()

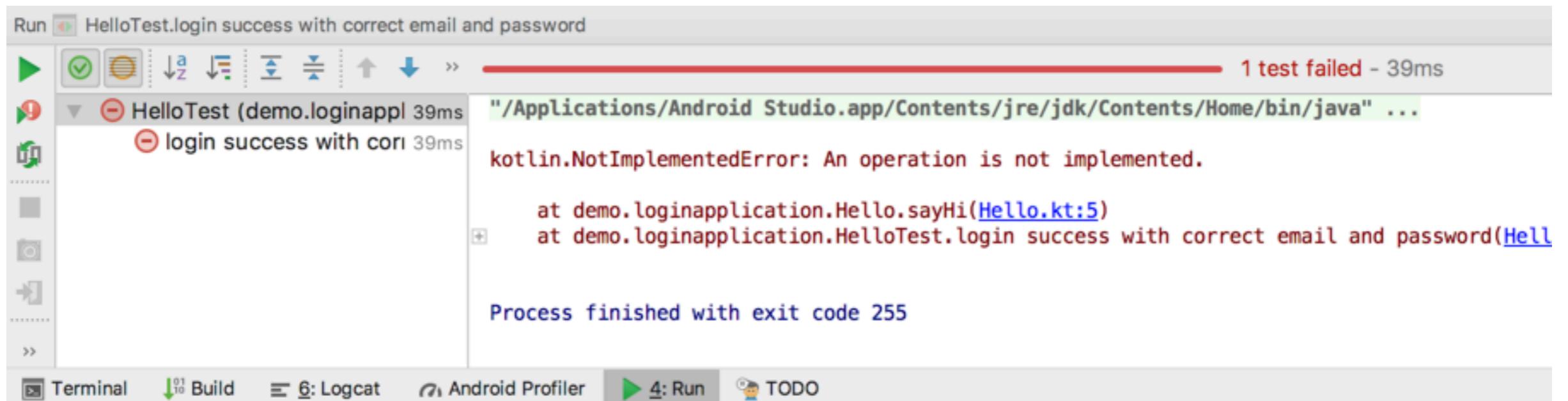
    // Act or When
    val actualResult = hello.sayHi(s: "somkiat")

    // Assert or Then
    assertEquals(expected: "Hi, somkiat", actualResult)
}
```



# Run first test case !!!

\$gradlew test

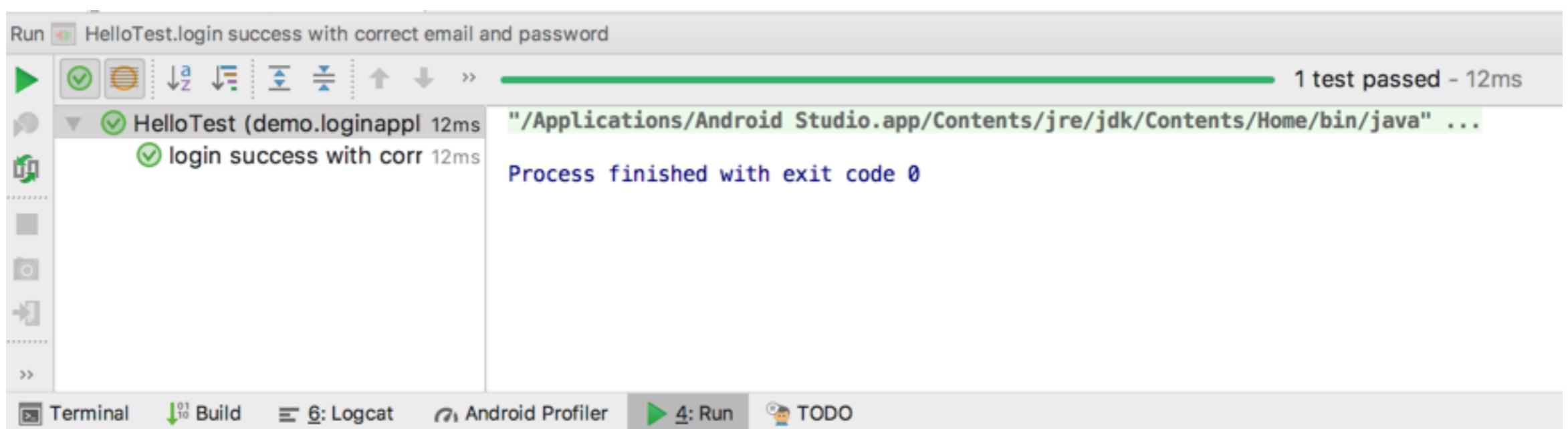


The screenshot shows the Android Studio interface with the 'Run' tab selected. The title bar says 'Run HelloTest.login success with correct email and password'. The status bar indicates '1 test failed - 39ms'. The test list shows a single entry: 'HelloTest (demo.loginapp) 39ms' with a failure icon. Expanding it reveals the error message: 'kotlin.NotImplementedError: An operation is not implemented.' followed by the stack trace: 'at demo.loginapplication.Hello.sayHi(Hello.kt:5)' and 'at demo.loginapplication.HelloTest.login success with correct email and password(Hell...'. Below the test list, the message 'Process finished with exit code 255' is displayed. The bottom navigation bar includes icons for Terminal, Build, Logcat, Android Profiler, Run, and TODO.



# Make test pass

```
class Hello {  
    fun sayHi(name: String): String {  
        return "Hi, $name"  
    }  
}
```



# See test result

/app/build/reports/tests/testDebugUnitTest

## Package demo.loginapplication

[all > demo.loginapplication](#)



### Classes

Class	Tests	Failures	Ignored	Duration	Success rate
<a href="#">ExampleUnitTest</a>	1	0	0	0.001s	100%
<a href="#">HelloTest</a>	1	0	0	0.003s	100%

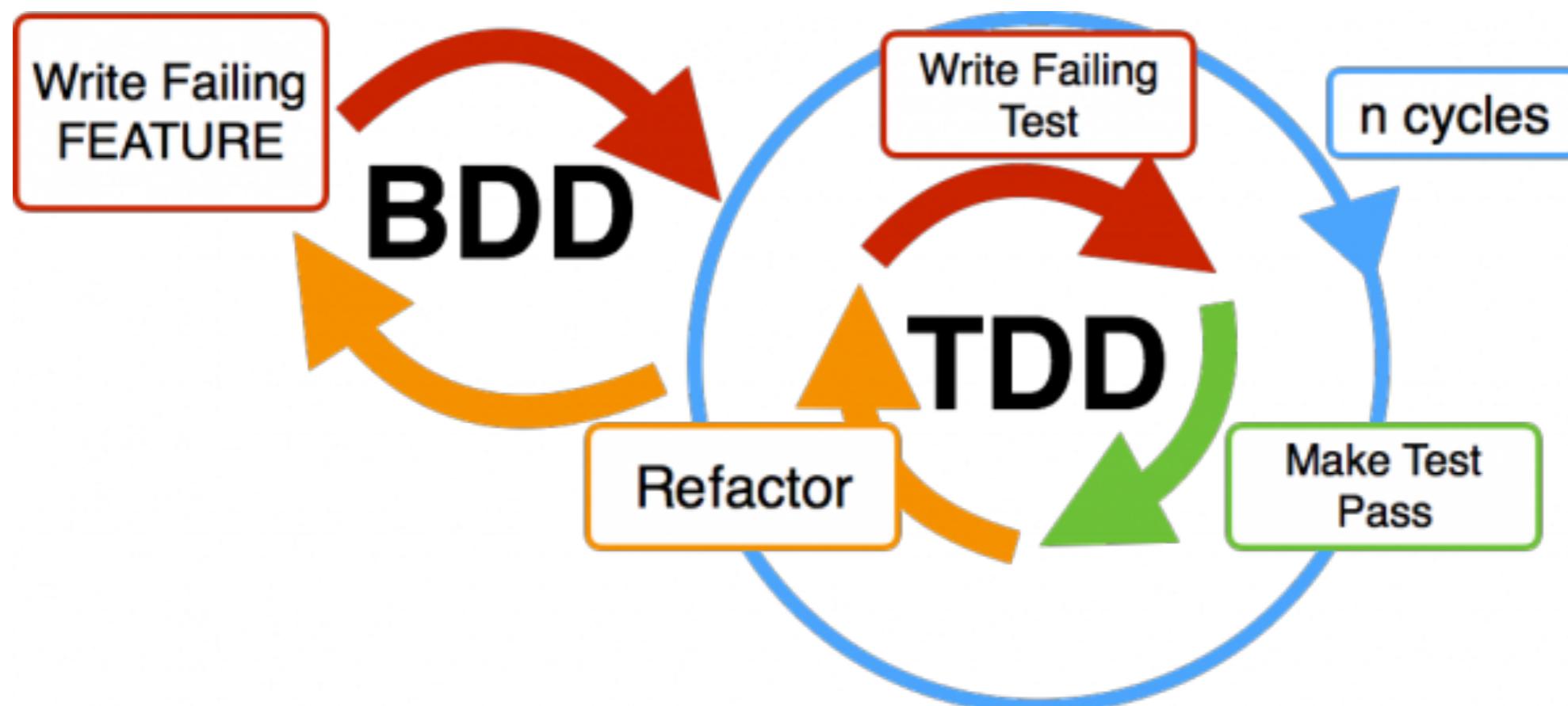


# Write next test case ?



# Start to write UI testing

Using espresso



<https://developer.android.com/training/testing/espresso/>



# Add new dependencies

## File /app/build.gradle

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"  
    implementation 'com.android.support:appcompat-v7:27.1.1'  
    implementation 'com.android.support:design:27.1.1'  
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'  
  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'com.android.support.test:runner:1.0.2'  
    androidTestImplementation 'com.android.support.test:rules:1.0.2'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'  
}
```



# Login success

Using ActivityTestRule to start first activity

```
class MainActivityTest {
```

```
@get:Rule  
val rule: ActivityTestRule<MainActivity>  
    = ActivityTestRule(MainActivity::class.java)
```



# Login success

Fill in data in email and close keyboard

```
@Test
fun success() {
    onView(withId(R.id.email)).perform(
        replaceText(stringToBeSet: "somkiat@xxx.com"),
        closeSoftKeyboard())

    onView(withId(R.id.password)).perform(
        replaceText(stringToBeSet: ""),
        closeSoftKeyboard())

    onView(withId(R.id.email_sign_in_button)).perform(click())

    onView(withText(text: "Success"))
        .inRoot(withDecorView(not(rule.activity.window.decorView)))
        .check(matches(isDisplayed()))
}
```



# Login success

Click login button

```
@Test
fun success() {
    onView(withId(R.id.email)).perform(
        replaceText(stringToBeSet: "somkiat@xxx.com"),
        closeSoftKeyboard())

    onView(withId(R.id.password)).perform(
        replaceText(stringToBeSet: ""),
        closeSoftKeyboard())

    onView(withId(R.id.email_sign_in_button)).perform(click())

    onView(withText(text: "Success"))
        .inRoot(withDecorView(not(rule.activity.window.decorView)))
        .check(matches(isDisplayed()))
}

}
```



# Login success

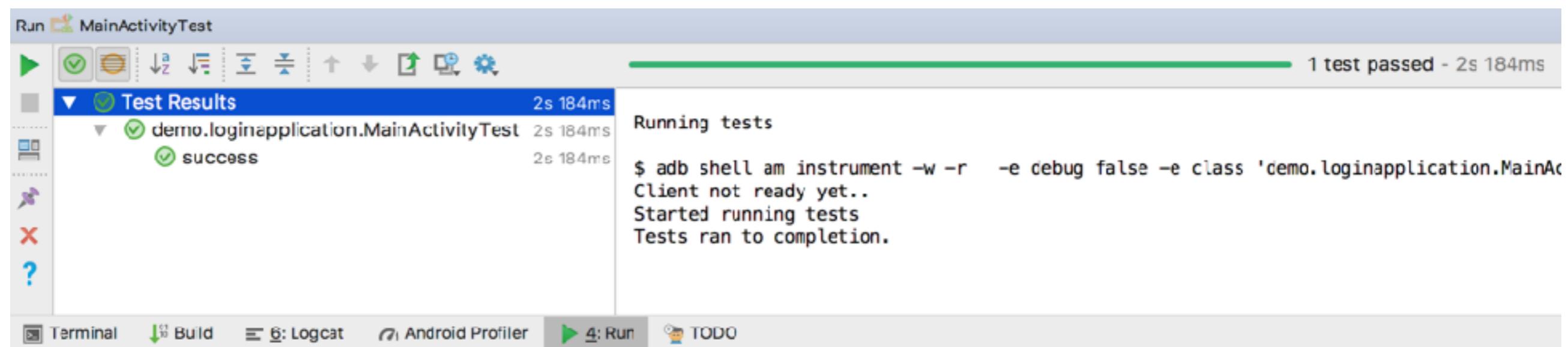
Check result message in Toast

```
@Test  
fun success() {  
    onView(withId(R.id.email)).perform(  
        replaceText(stringToBeSet: "somkiat@xxx.com"),  
        closeSoftKeyboard())  
  
    onView(withId(R.id.password)).perform(  
        replaceText(stringToBeSet: ""),  
        closeSoftKeyboard())  
  
    onView(withId(R.id.email_sign_in_button)).perform(click())  
  
    onView(withText(text: "Success"))  
        .inRoot(withDecorView(not(rule.activity.window.decorView)))  
        .check(matches(isDisplayed()))  
}  
}
```



# Run test

\$gradlew cAT



The screenshot shows the Android Studio interface with the 'Run' tab selected. The title bar says 'Run MainActivityTest'. The main area displays the 'Test Results' section, which shows one test passed ('demo.loginapplication.MainActivityTest') in 2s 184ms. Below this, the terminal output shows the command '\$ adb shell am instrument -w -r -e debug false -e class 'demo.loginapplication.MainActivityTest''. It also shows messages indicating the client is not ready yet, tests are starting, and they have run to completion.

Test	Time
demo.loginapplication.MainActivityTest	2s 184ms

```
Running tests
$ adb shell am instrument -w -r -e debug false -e class 'demo.loginapplication.MainActivityTest'
Client not ready yet..
Started running tests
Tests ran to completion.
```



# See test result

/app/build/reports/androidTests/connected

## Class demo.loginapplication.MainActivityTest

[all](#) > [demo.loginapplication](#) > MainActivityTest

1  
tests

0  
failures

1.888s  
duration

100%  
successful

### Tests

Test

SM-G360HU - 4.4.4

success

passed (1.888s)



# See test result

More than one device

## Class demo.loginapplication.MainActivityTest

[all](#) > [demo.loginapplication](#) > MainActivityTest



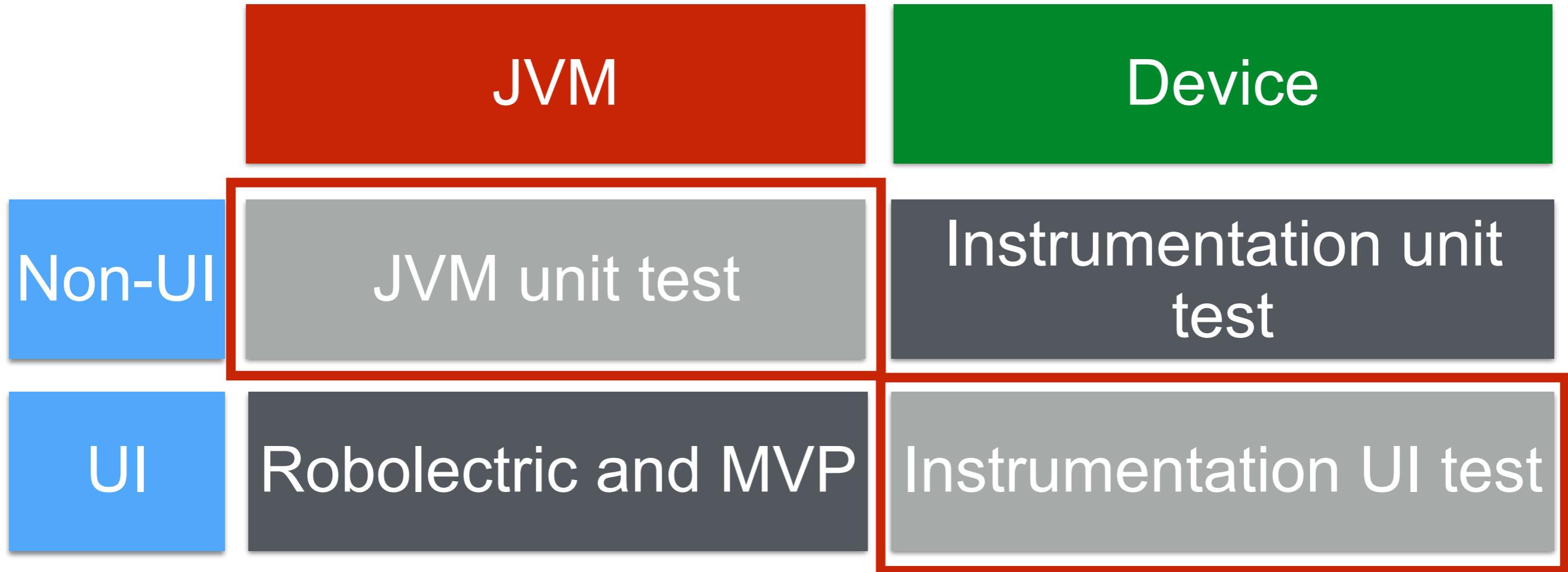
Tests	Devices
<b>Test</b>	<b>Pixel_2_API_P(AVD) - P</b>
success	passed (3.585s)

Test	Pixel_2_API_P(AVD) - P	SM-G360HU - 4.4.4
success	passed (3.585s)	passed (1.865s)



# Android Testing



/src/test

/src/androidTest



# Code coverage



**"Code coverage can show the high risk areas in a program, but never the risk-free."**

Paul Reilly, 2018, Kotlin TDD with Code Coverage



# Code coverage

A tool to measure how much of your code is covered by tests that break down into classes, methods and lines.



# Code coverage

But 100% of code coverage does not mean that your code is 100% correct



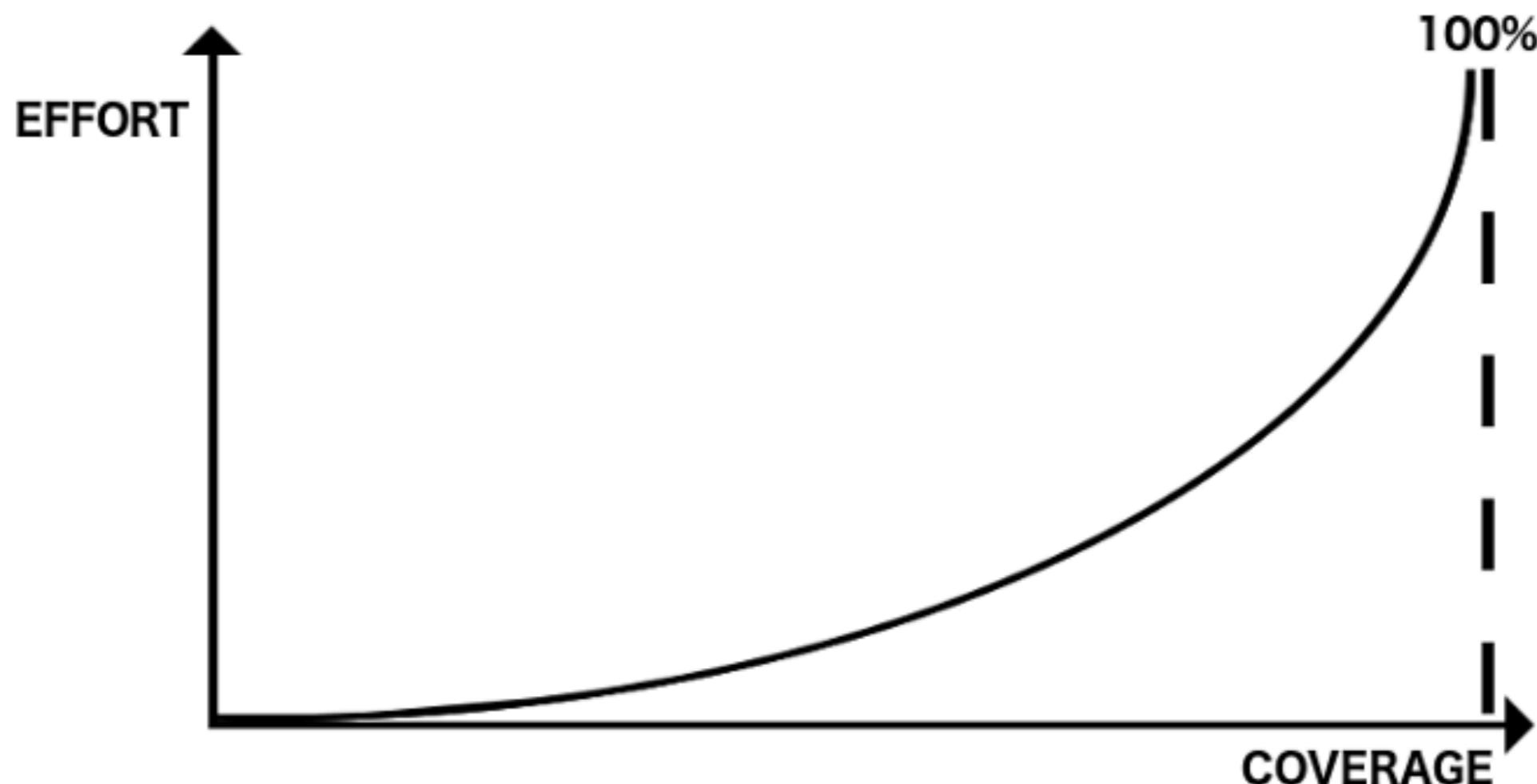
# Code coverage

Powerful tool to improve the quality of your code

**Code coverage != quality of tests**



# Code coverage 100% ?



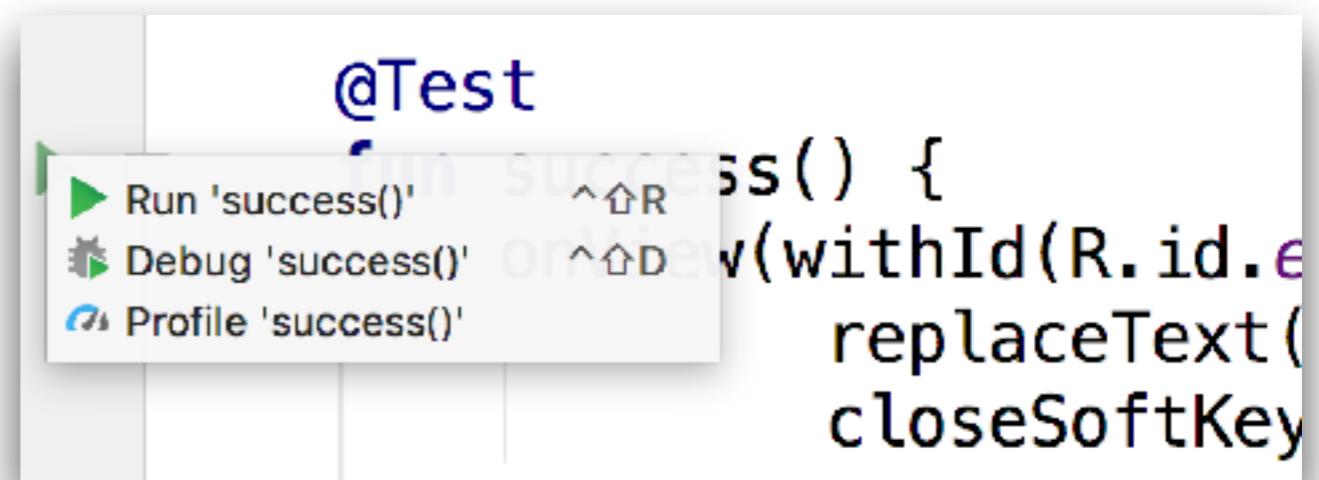
# Code coverage for android

Disabled by default !!

## Unit test



androidTest



# Try to enable code coverage

Use Jacoco library



<https://www.jacoco.org/jacoco/>



# 1. Enable coverage in debug

In file /app/build.gradle

```
buildTypes {  
    debug {  
        testCoverageEnabled true  
    }  
  
    release {  
        minifyEnabled false  
        proguardFiles getDefaultPro  
    }  
}
```



## 2. Use jacobo library

In file /app/build.gradle

```
apply plugin: 'jacoco'

jacoco {
    toolVersion = "0.8.1"
}
```



# 3. Merge result of coverage

Create new grade's task

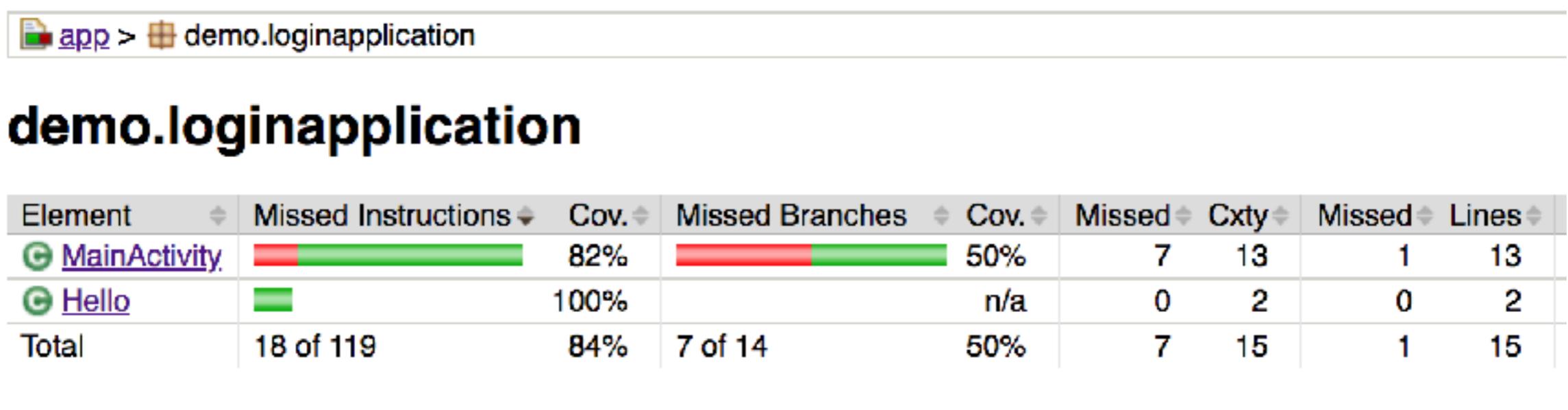
```
task jacocoTestReport(type: JacocoReport, dependsOn: ['testDebugUnitTest',  
    reports {  
        xml.enabled = true  
        html.enabled = true  
    }  
  
    def fileFilter = [ '**/R.class', '**/R*.class', '**/BuildConfig.*', '*  
    def debugTree = fileTree(dir: "$project.buildDir/tmp/kotlin-classes/debug")  
    def mainSrc = "$project.projectDir/src/main/java"  
  
    sourceDirectories = files([mainSrc])  
    classDirectories = files([debugTree])  
    executionData = fileTree(dir: project.buildDir, includes: [  
        'jacoco/testDebugUnitTest.exec', 'outputs/code-coverage/connected'  
    ])  
}
```



# 4. Run test with code coverage

\$gradlew clean jacocoTestReport

Result in /build/reports/jacoco/jacocoTestReport



# More detail of coverage

app > demo.loginapplication > MainActivity.kt

## MainActivity.kt

```
1. package demo.loginapplication
2.
3. import android.support.v7.app.AppCompatActivity
4. import android.os.Bundle
5. import android.view.View
6. import android.widget.Toast
7. import kotlinx.android.synthetic.main.activity_main.*
8.
9. class MainActivity : AppCompatActivity(), View.OnClickListener {
10.
11.     override fun onCreate(savedInstanceState: Bundle?) {
12.         super.onCreate(savedInstanceState)
13.         setContentView(R.layout.activity_main)
14.
15.         email_sign_in_button.setOnClickListener(this)
16.     }
17.
18.     override fun onClick(v: View?) {
19.         when(v?.id) {
20.             R.id.email_sign_in_button -> {
21.                 processLogin()
22.             }
23.         }
24.     }
25.
26.     private fun processLogin() {
27.         if(email.text.toString().equals("somkiatexxx.com")) {
28.             Toast.makeText(this@MainActivity, "Success", Toast.LENGTH_SHORT).show()
29.         } else {
30.             Toast.makeText(this@MainActivity, "Fail", Toast.LENGTH_SHORT).show()
31.         }
32.     }
33. }
```

Miss in branch coverage

Miss in line coverage



# Improve UI Testing with Espresso



# Improve test structure

Robot or Page Object pattern



# Easy to read and understand

## Robot or Page Object pattern

```
@Test  
fun success_with_robot() {  
    robot  
        .setEmail("somkiat@xxx.com")  
        .setPassword("")  
        .clickLogin()  
        .mustShow( text: "Success")  
}
```



# Use library/DSL :: Kakao



<https://github.com/agoda-com/Kakao>



# Add dependencies

In file /app/build.gradle

```
dependencies {  
    androidTestImplementation 'com.agoda.kakao:kakao:1.3.0'  
}
```



# Easy to read and understand

```
@Test  
fun success_with_kakao() {  
    screen { this: TestLoginScreen  
        email { this: KEditText  
            replaceText( text: "somkiat@xxx.com" )  
        }  
        password { this: KEditText  
            replaceText( text: "" )  
        }  
        login { this: KButton  
            click()  
        }  
    }  
}
```



# Add more features ...

