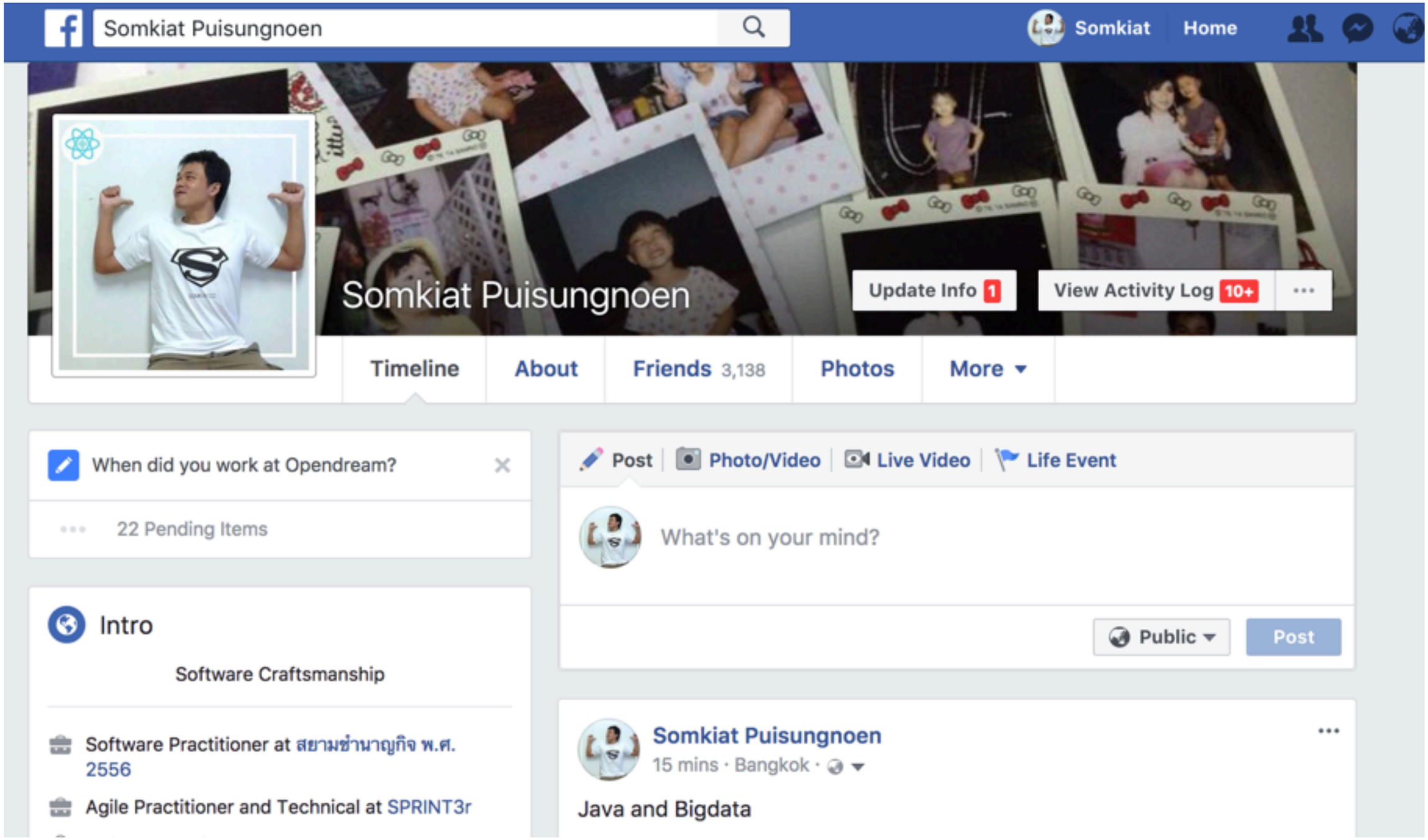
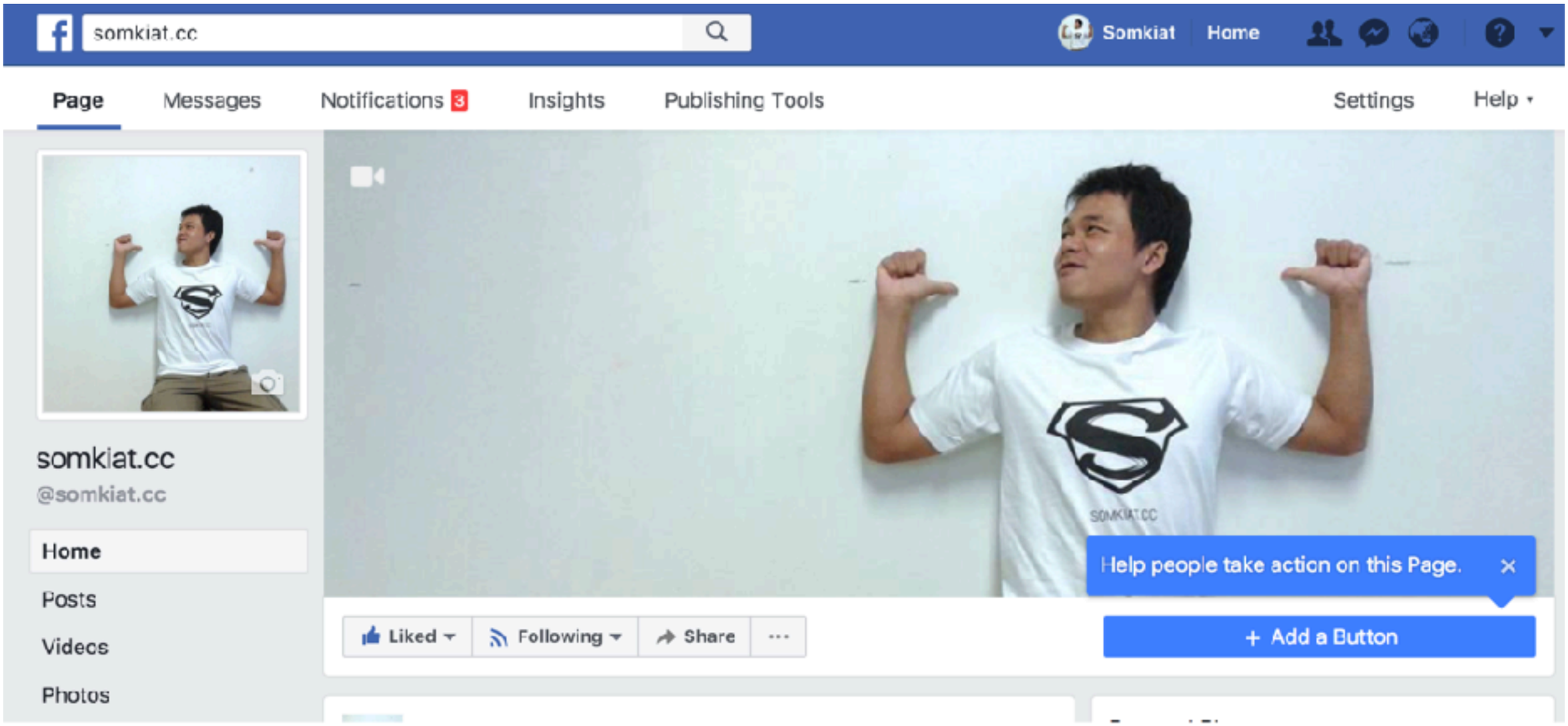
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Automated Testing for Android with Kotlin





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

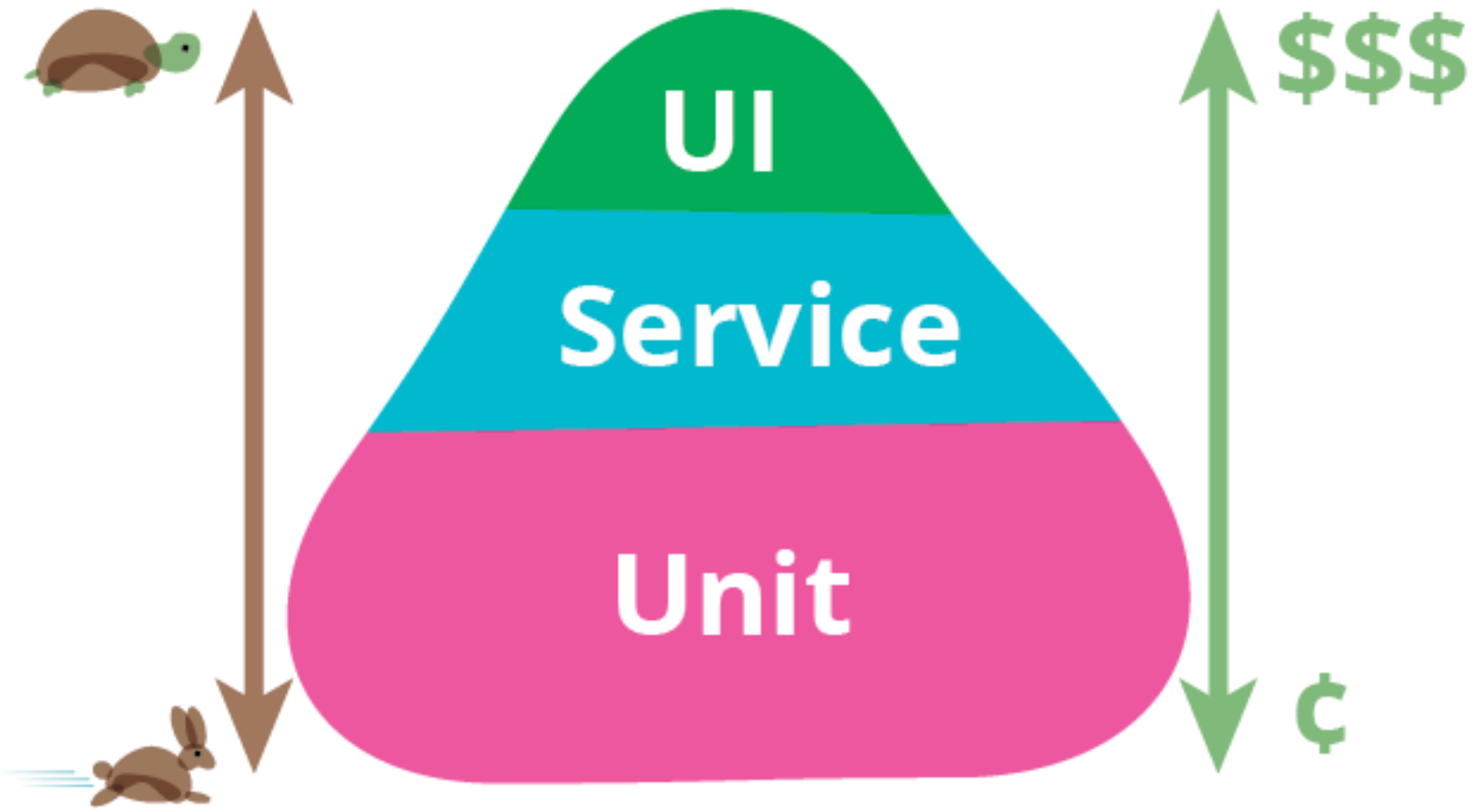
How to **THINK** when and where you should test

What you need to know ?

Android/Kotlin
Android Studio
JUnit 4
Espresso

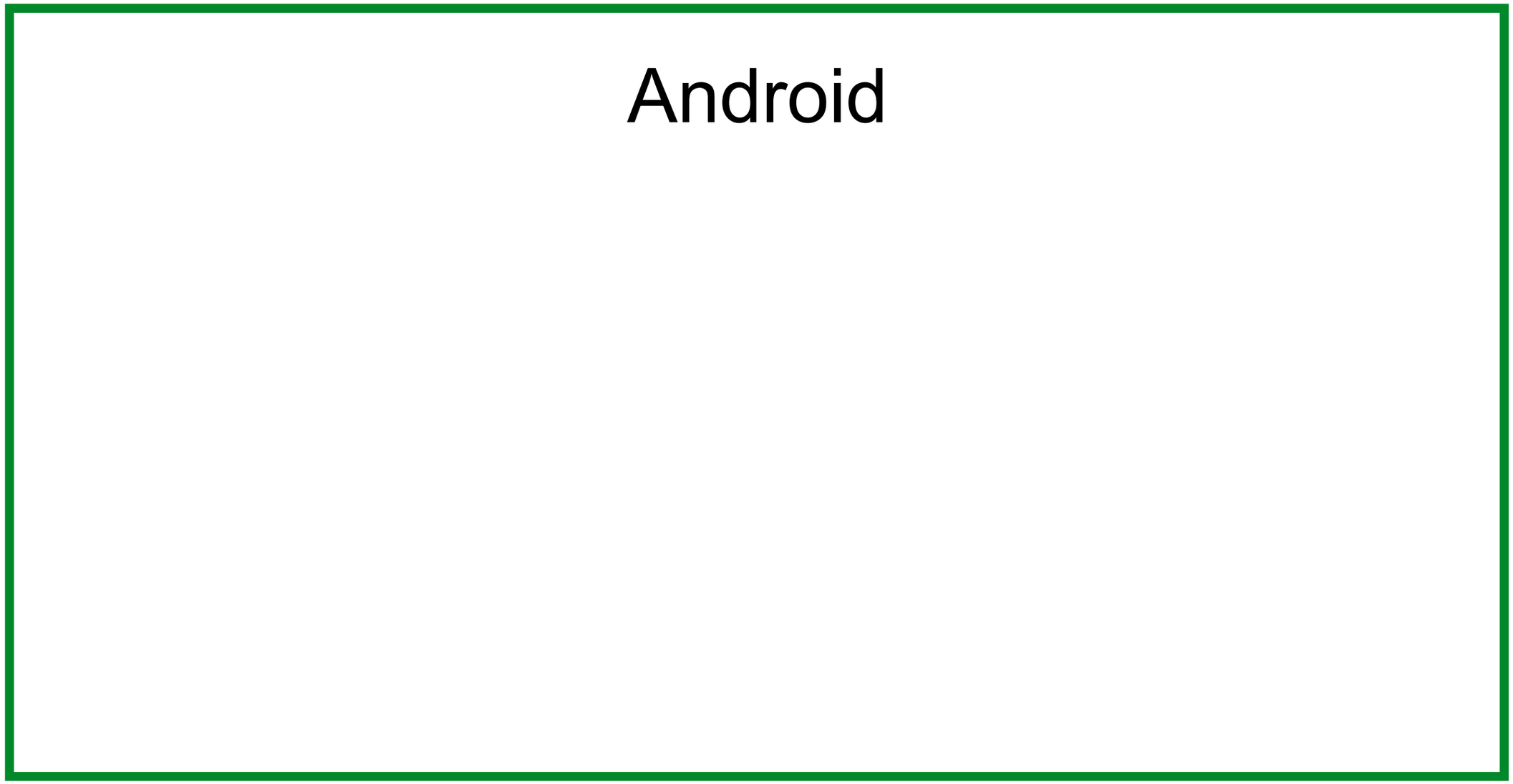
Type of testing

Testing Pyramid

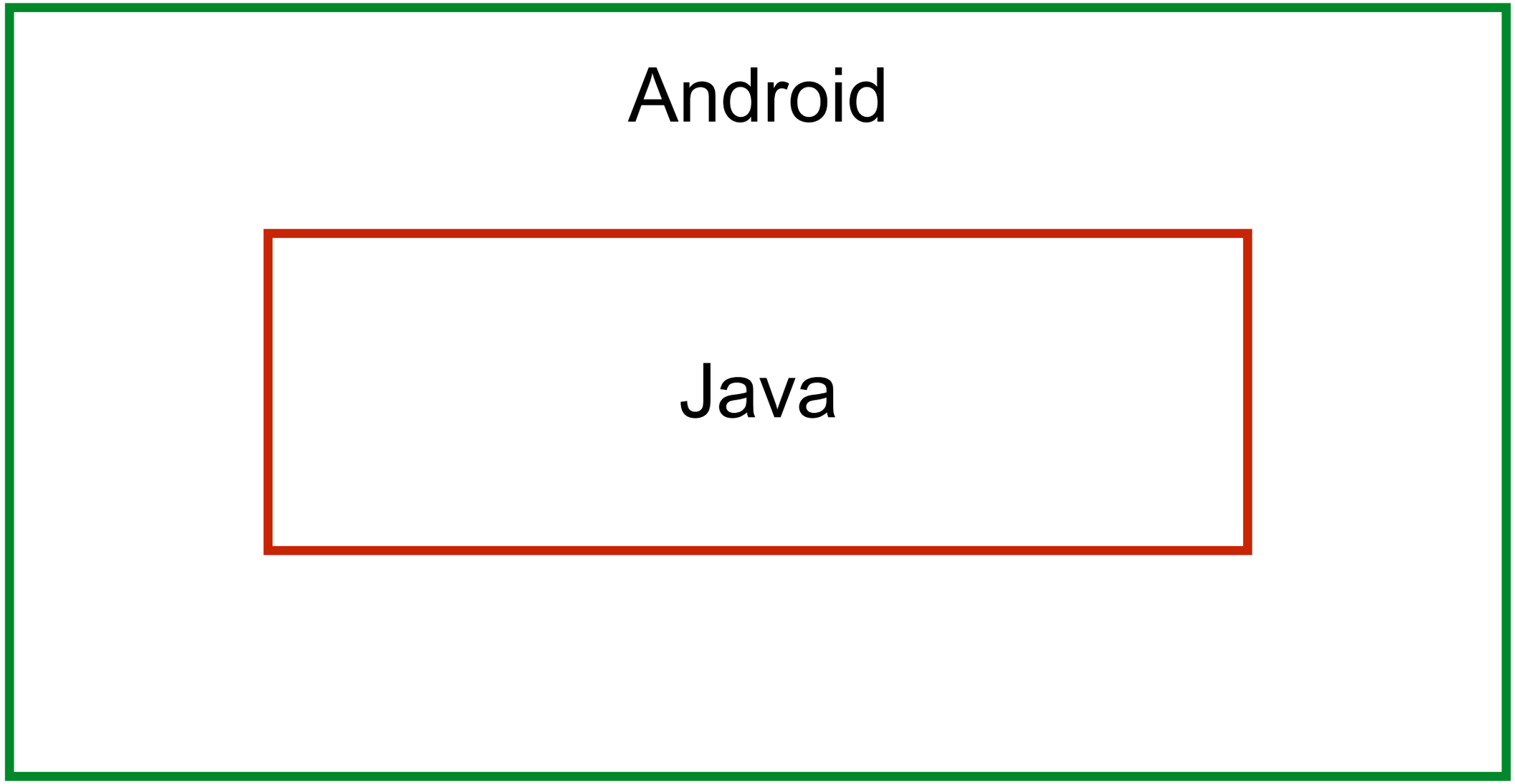


https://martinfowler.com/bliki/TestPyramid.html

Android Testing



Android Testing



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

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI		Instrumentation UI test

/src/test /src/androidTest

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI		Instrumentation UI test

UI TESTING FOR ANDROID
espresso

/src/test /src/androidTest

Resources

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

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

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI	? ? ?	Instrumentation UI test

/src/test /src/androidTest

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI	Robolectric and MVP	Instrumentation UI test

/src/test /src/androidTest

Rule of thumb

Instrumentation tests are slower than JVM tests.

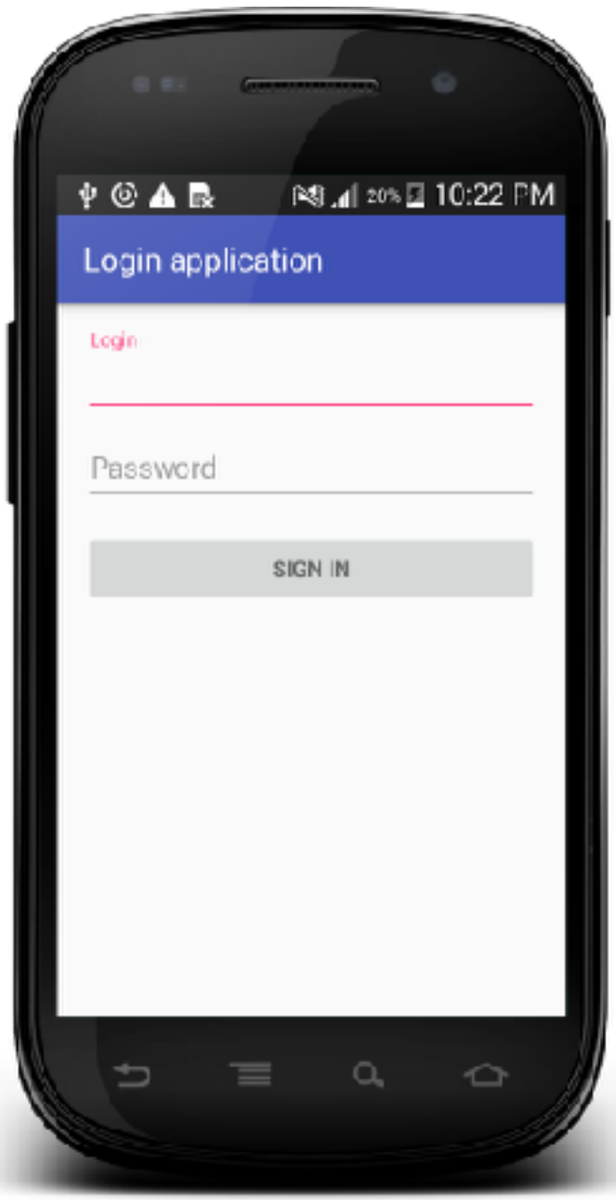
Try to separate the standard Java code from Android-dependent code.

Workshop with Testing

Hello Testing with Kotlin

Login application

Login Page



Result

Success

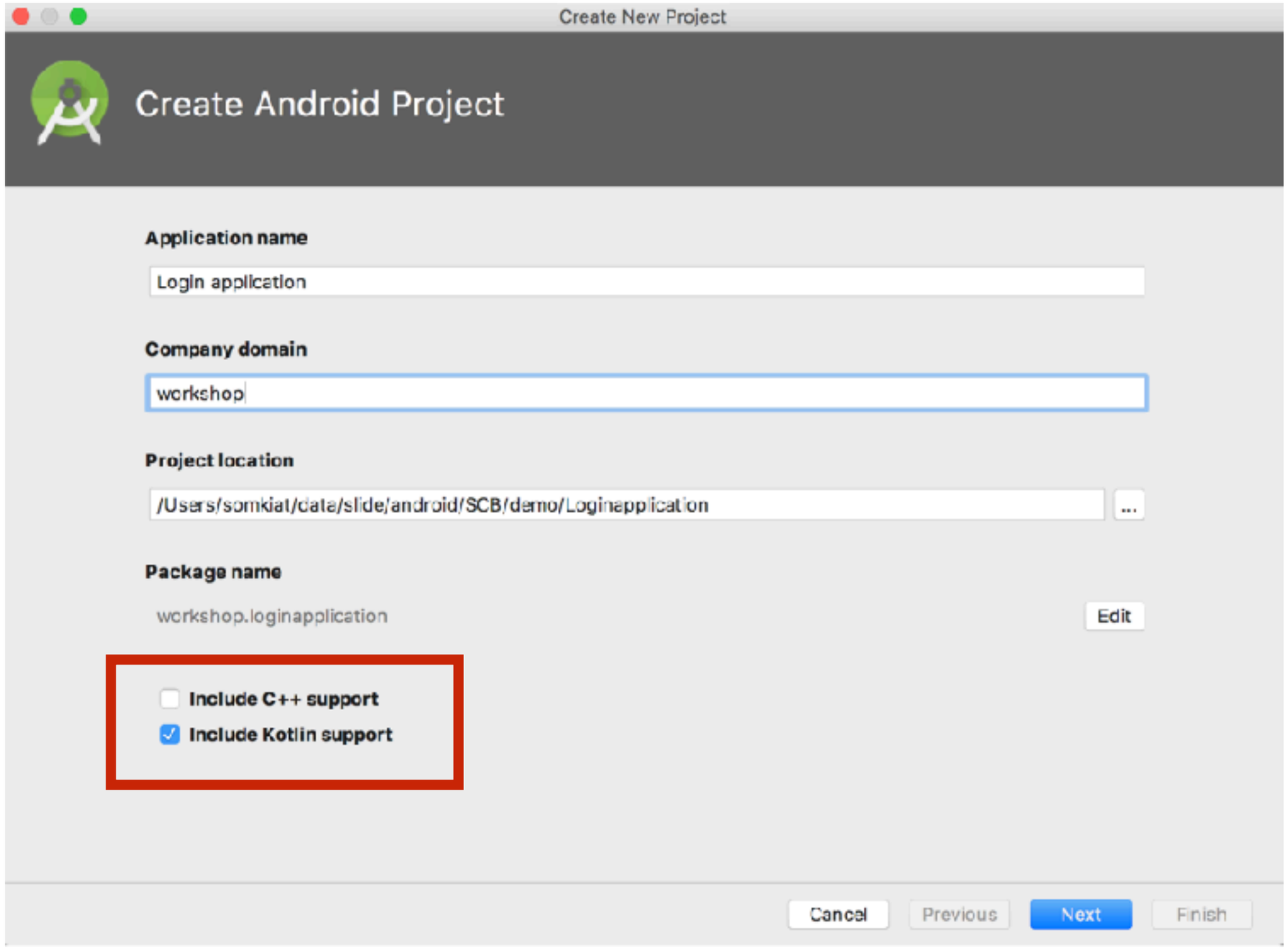
Failure

Test cases

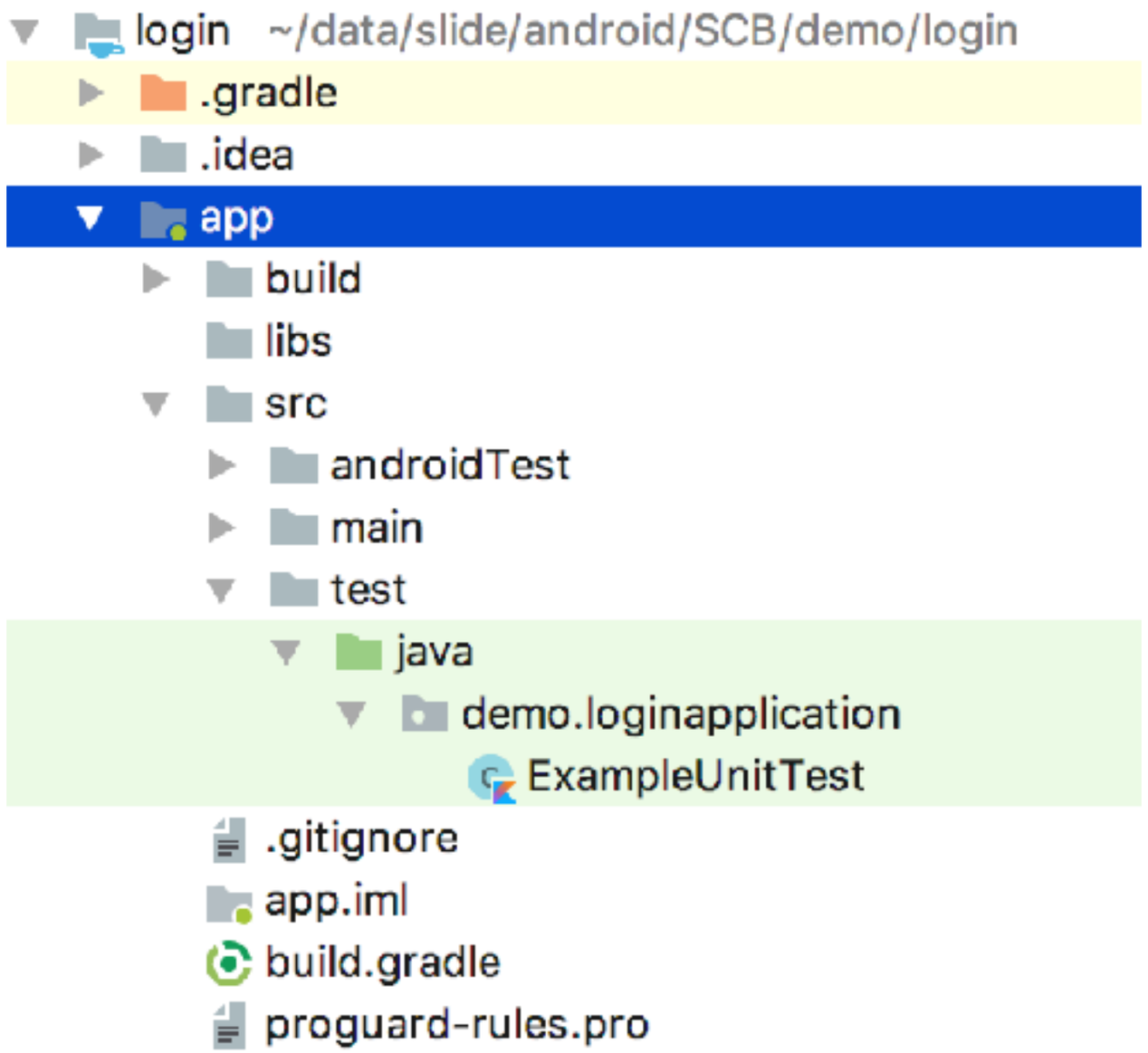
Test Case Name	Email	Password	Expected Result
Login success	somkiat@xxx.com		Show **success** message
Login fail			Show **fail** message

Let’ 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

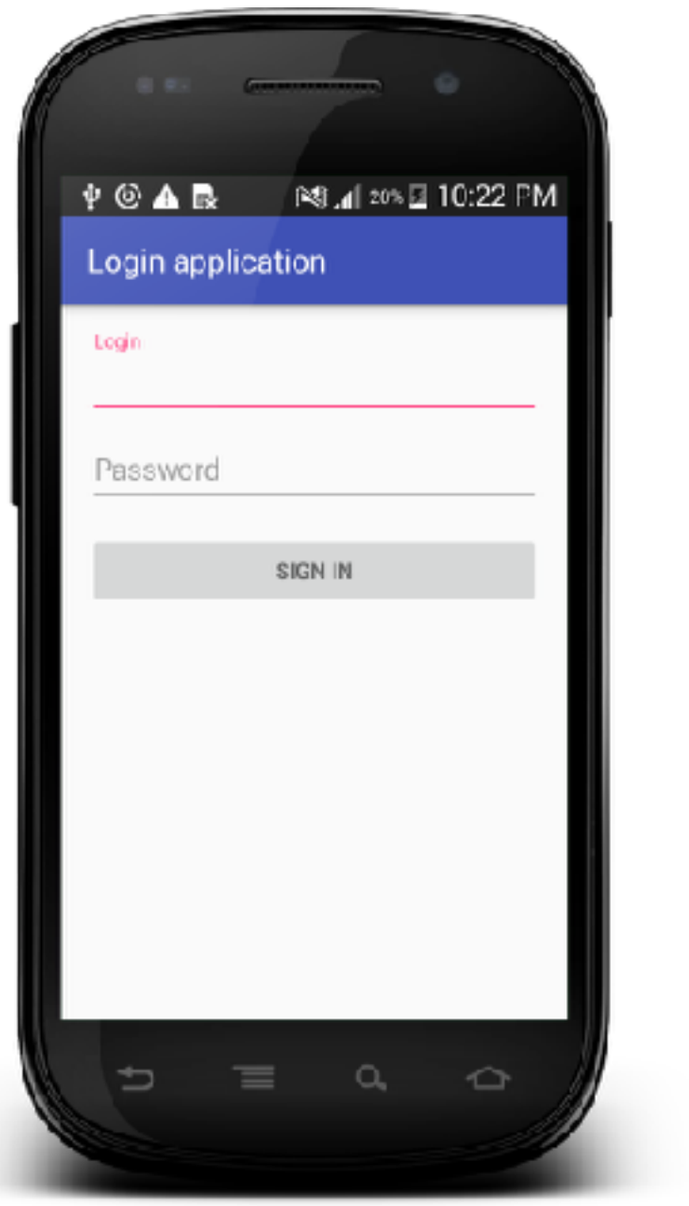
Open file **/app/build.gradle**

```
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'
}
```

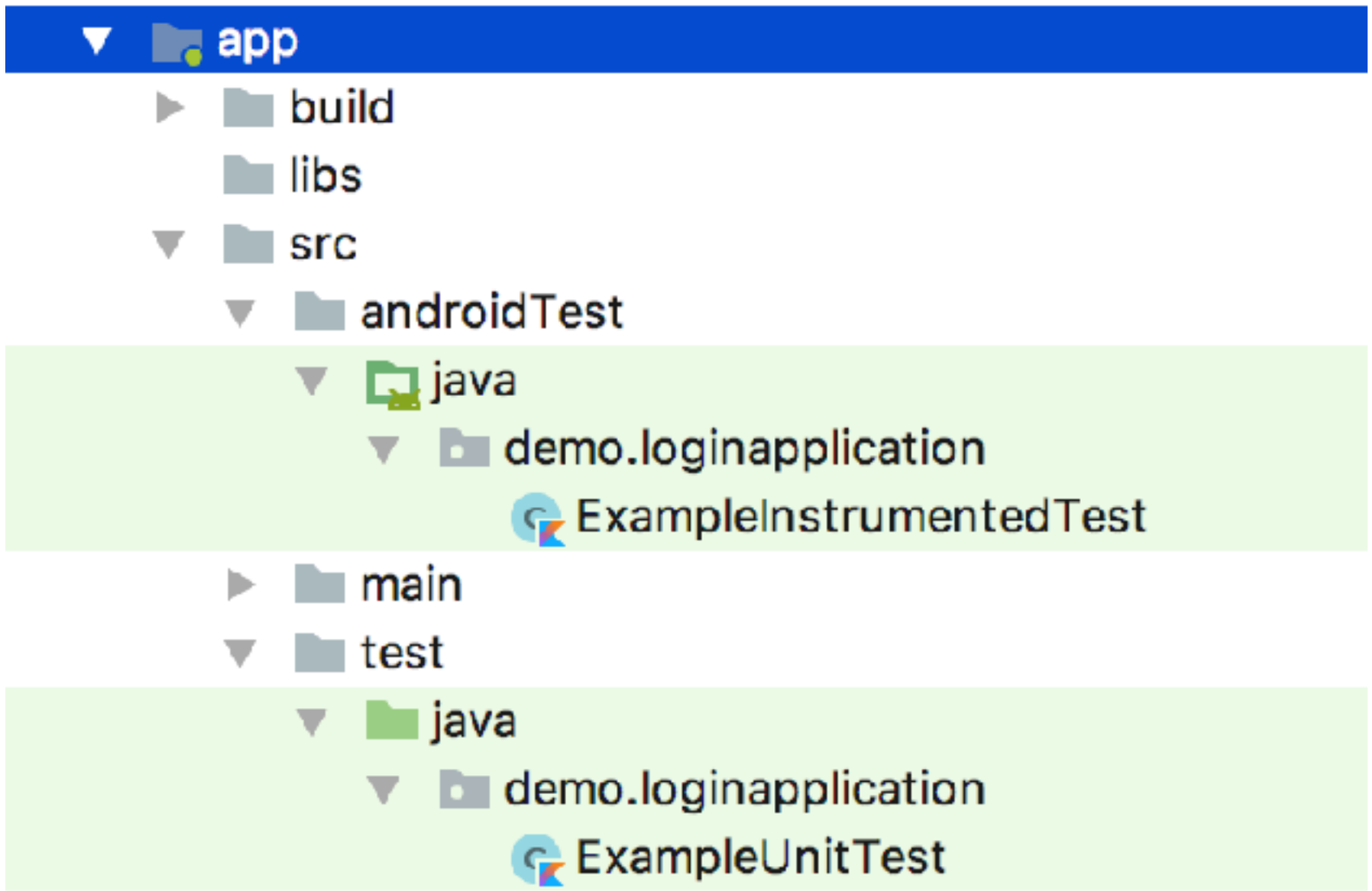
4. Develop login application

Import from Demo project

5. Run app

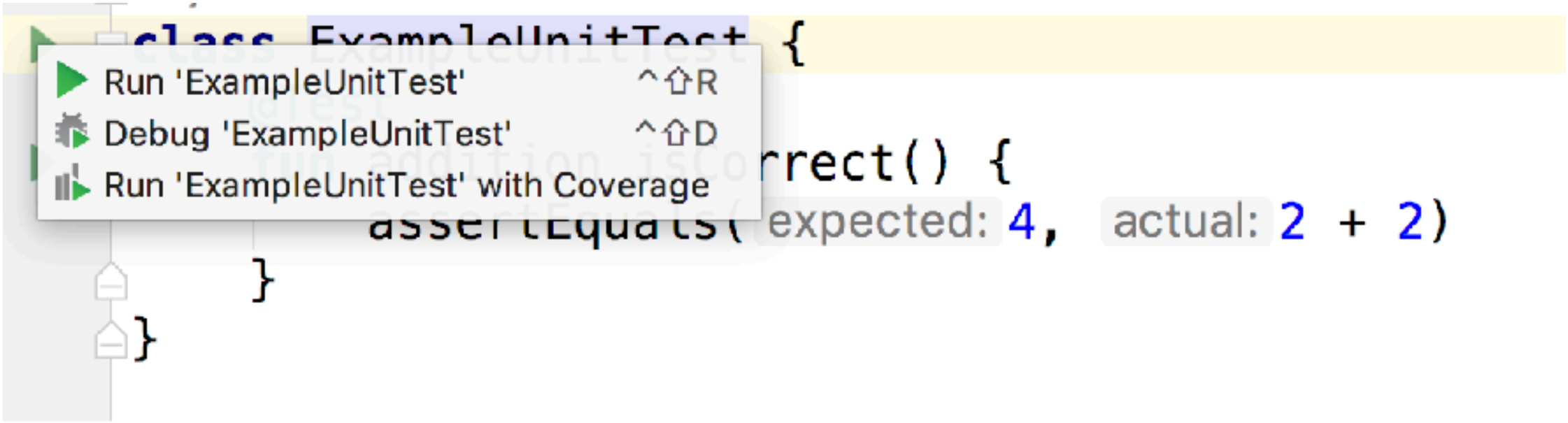


6. Structure of tests



7. Run tests

$gradlew test



8. See test result

/app/build/reports/tests/testDebugUnitTest

Test Summary

1	0	0	0.001s
tests	failures	ignored	duration

100% successful

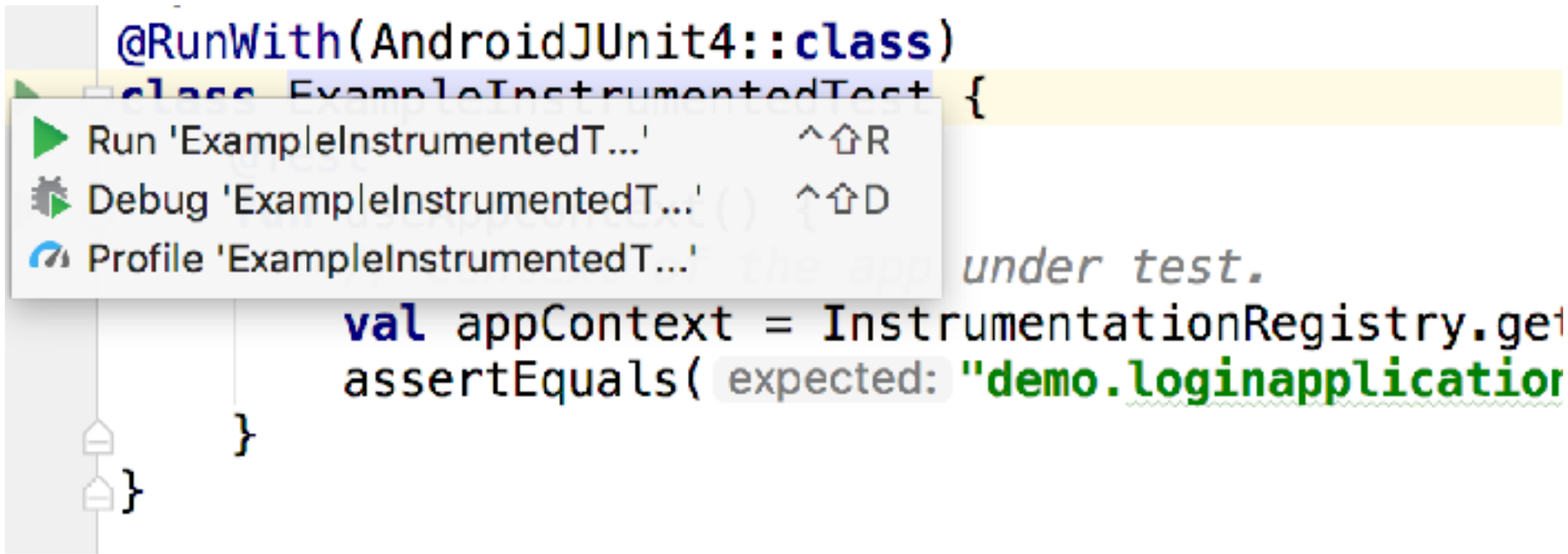
Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
demo.loginapplication	1	0	0	0.001s	100%

9. Run androidTest

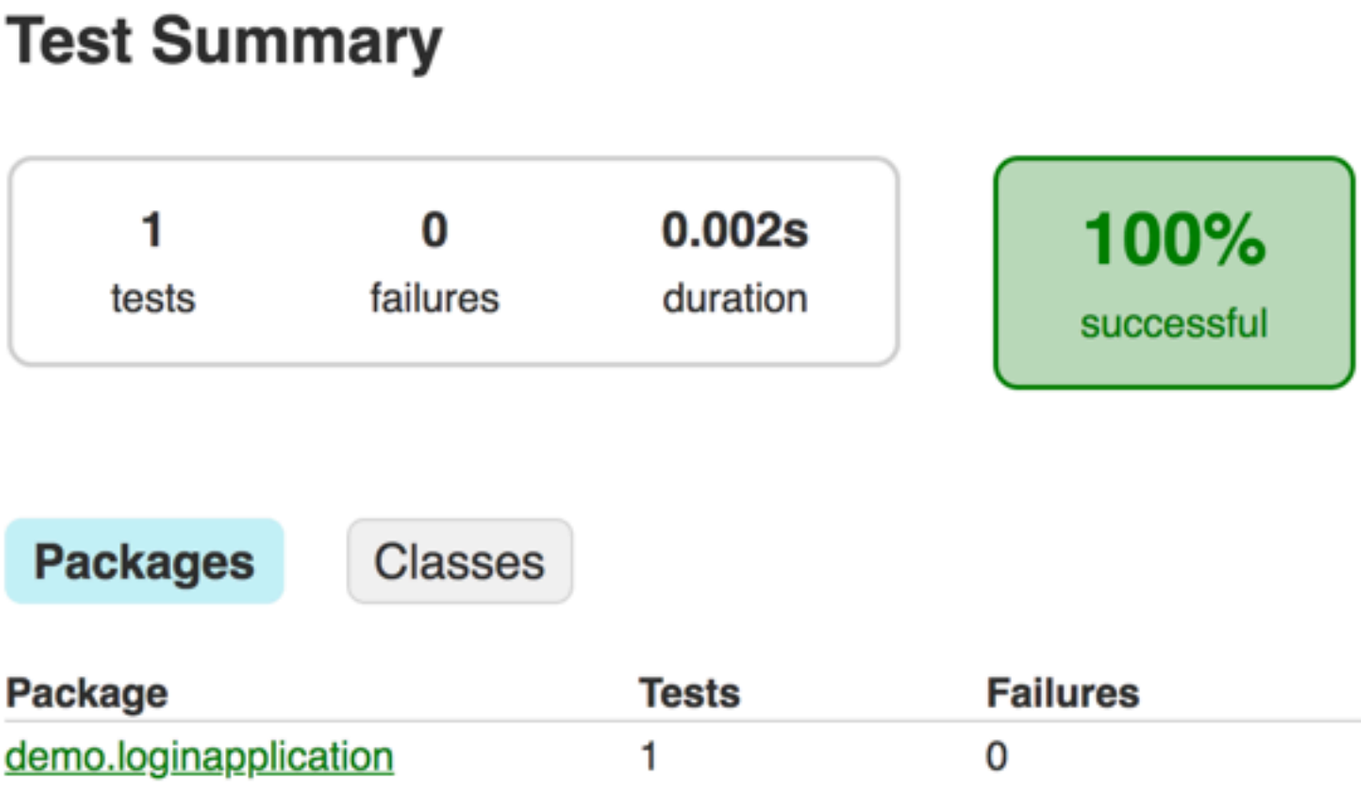
$gradlew **c**onnected**A**ndroid**T**est

$gradlew cAT



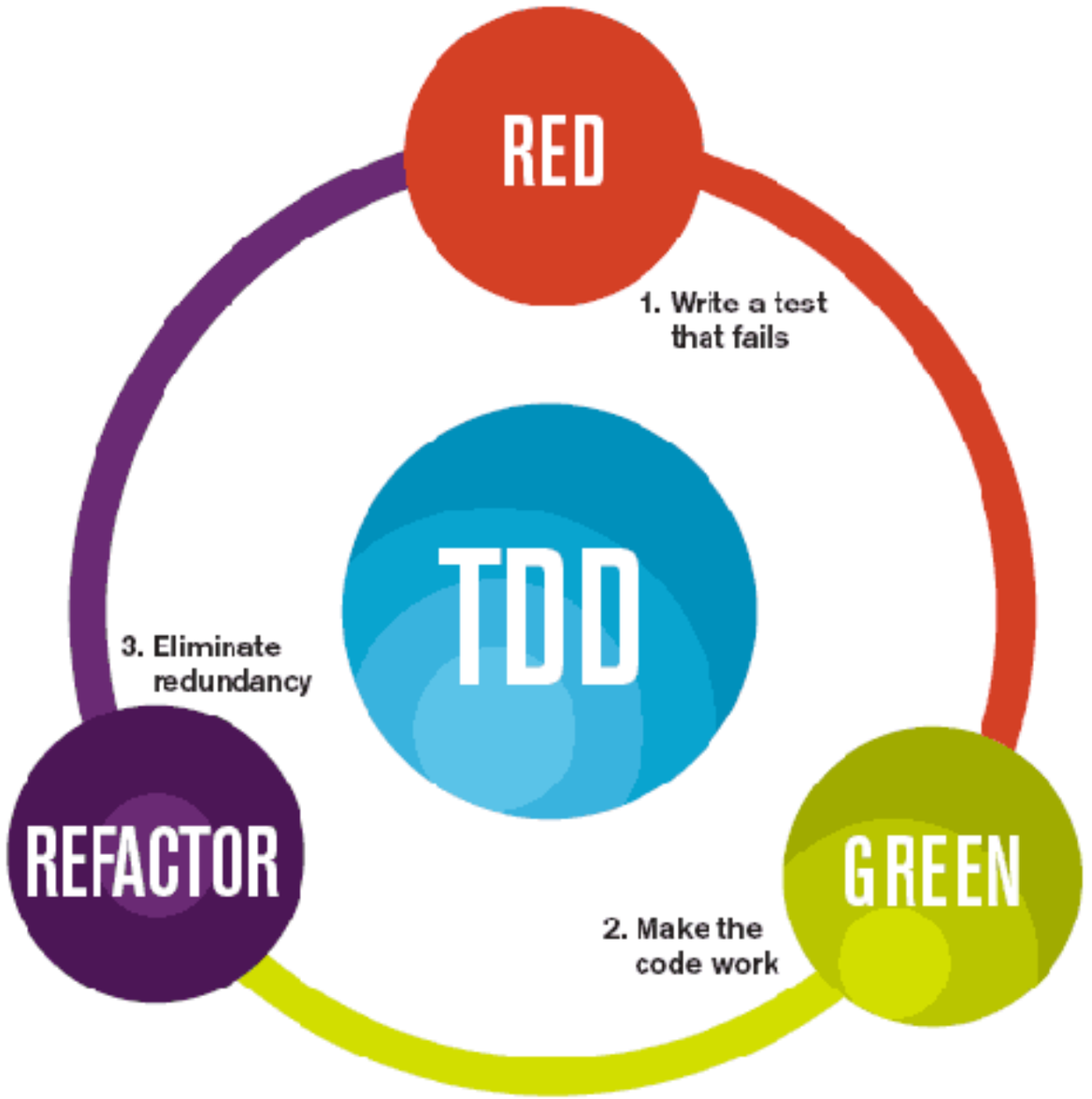
10. See test result

/app/build/reports/androidTests/connected



11. Write your first test case

Unit test or Android Test ?

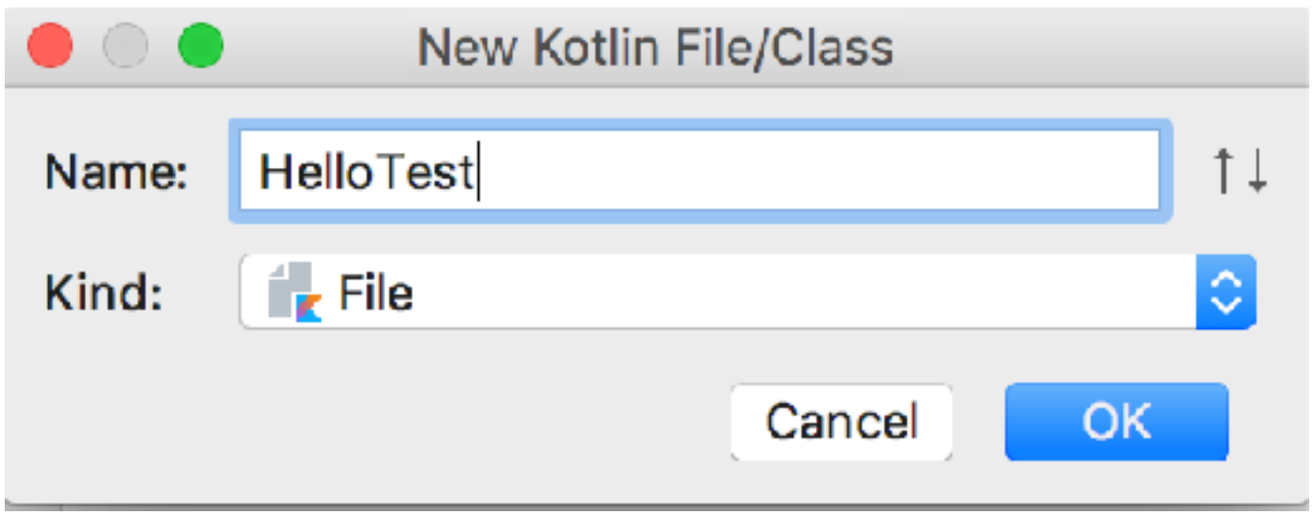


The mantra of Test-Driven Development (TDD) is "red, green, refactor."

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

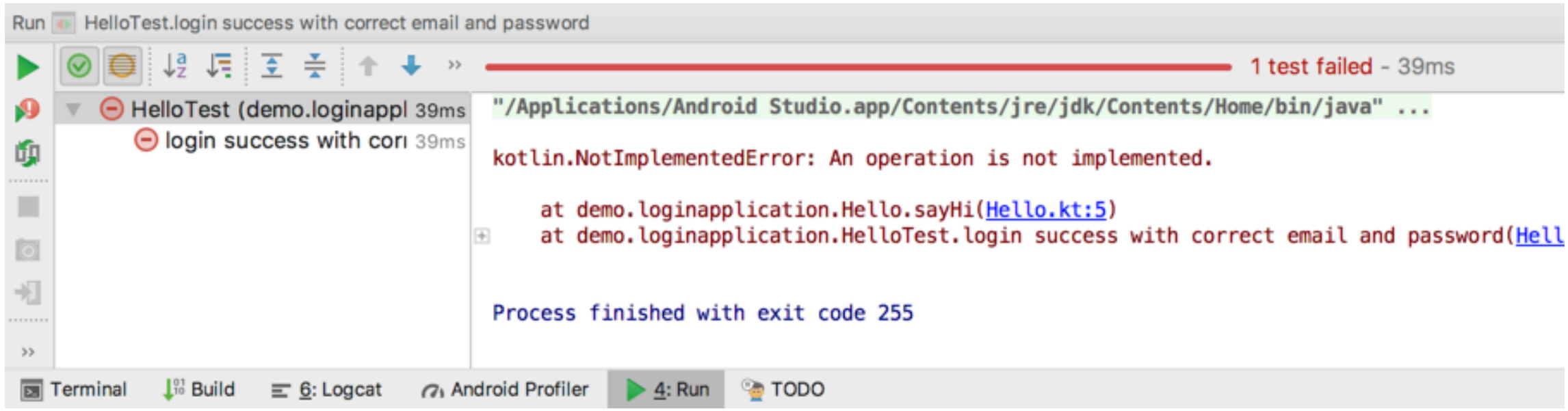
```
class HelloTest {

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

        // Act or When

        // Assert or Then
    }

}
```

Design as a Test

```kotlin
@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)
}
```

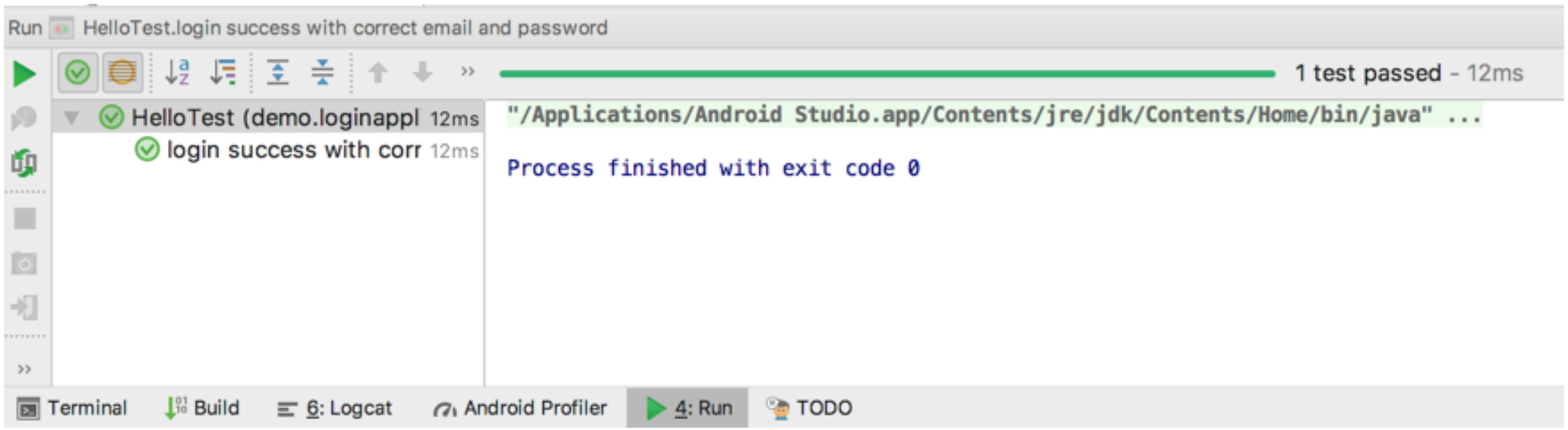
Run first test case !!!

$gradlew test



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

2	0	0	0.004s
tests	failures	ignored	duration

100%
successful

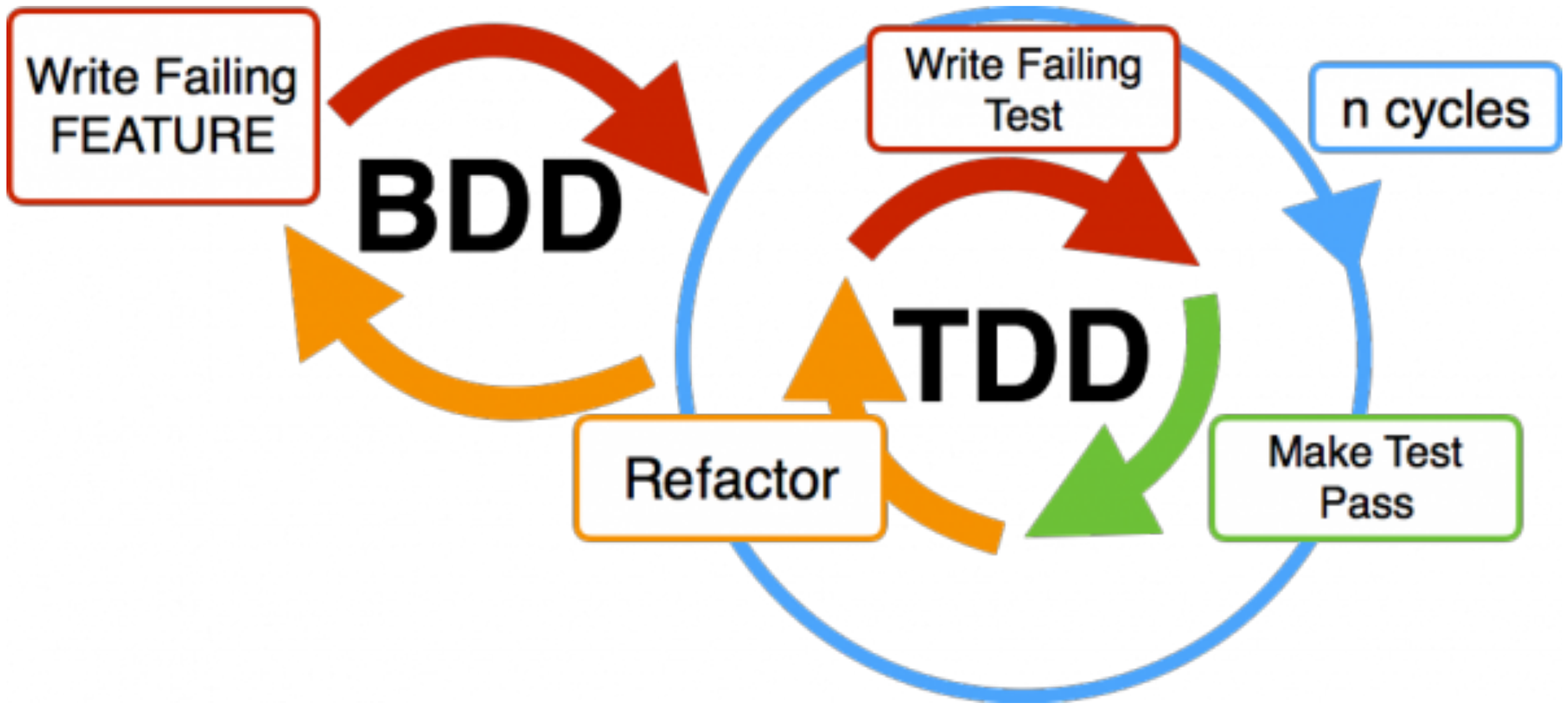
Classes

Class	Tests	Failures	Ignored	Duration	Success rate
ExampleUnitTest	1	0	0	0.001s	100%
HelloTest	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()))

}
```

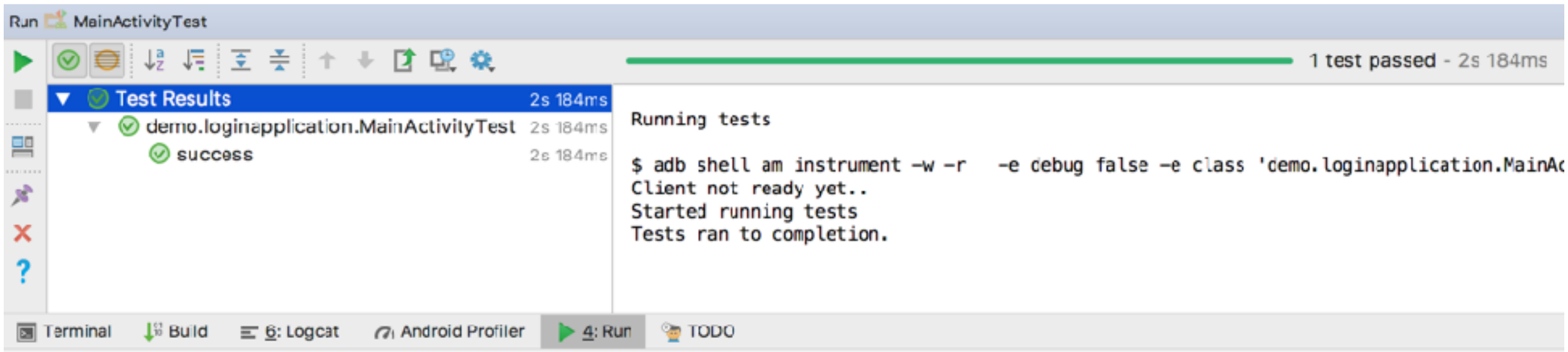
Login success

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



See test result

/app/build/reports/androidTests/connected

Class demo.loginapplication.MainActivityTest

all > demo.loginapplication > MainActivityTest

1	0	1.888s
tests	failures	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

2	0	5.450s
tests	failures	duration

100%
successful

Tests | Devices

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

Android Testing

	JVM	Device
Non-UI	JVM unit test	Instrumentation unit test
UI	Robolectric and MVP	Instrumentation UI test

/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

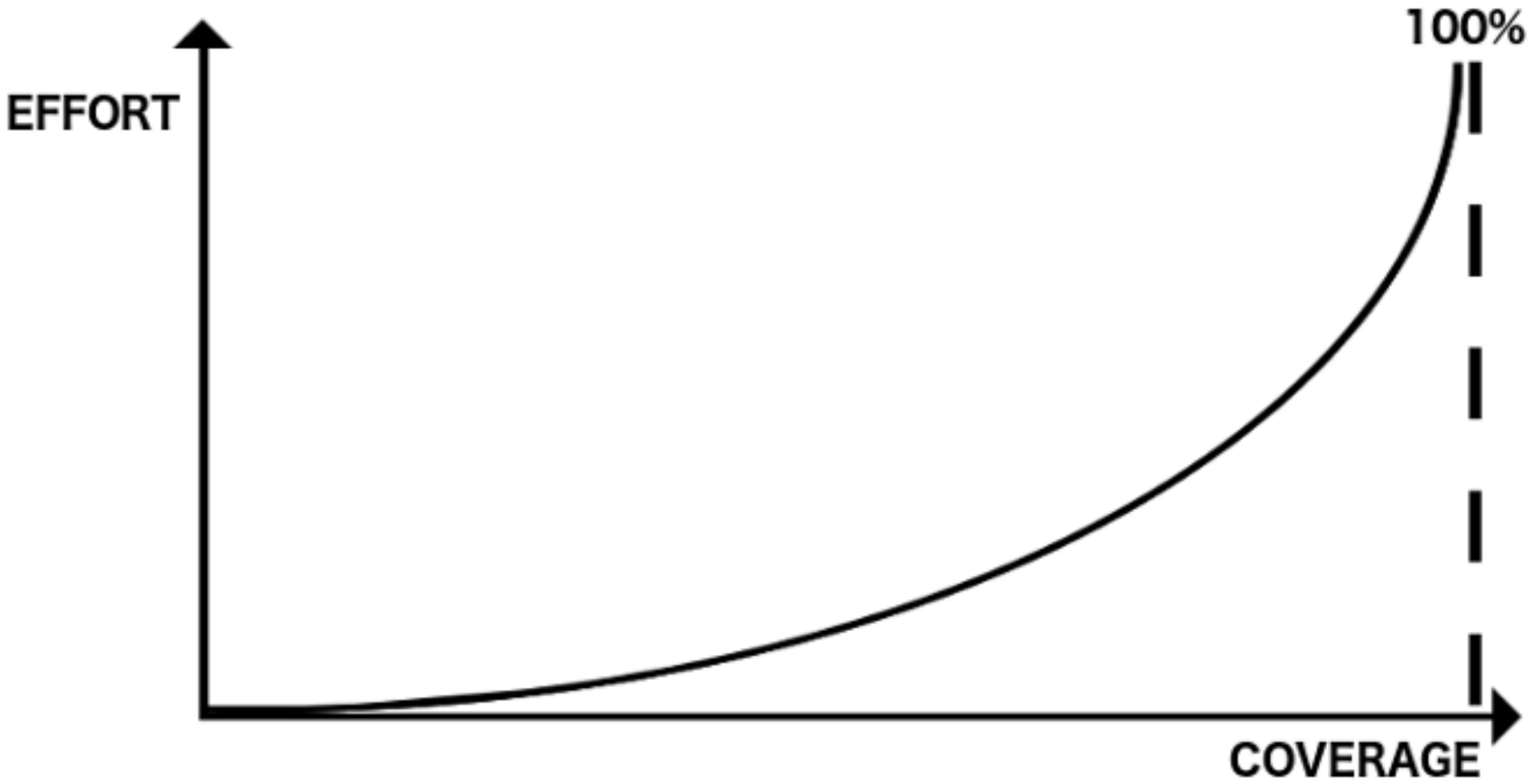
But 100% of code coverage does not mean that your code is 100% correct

Code coverage

Powerful tool to improve the quality of your code

Code coverage != quality of tests

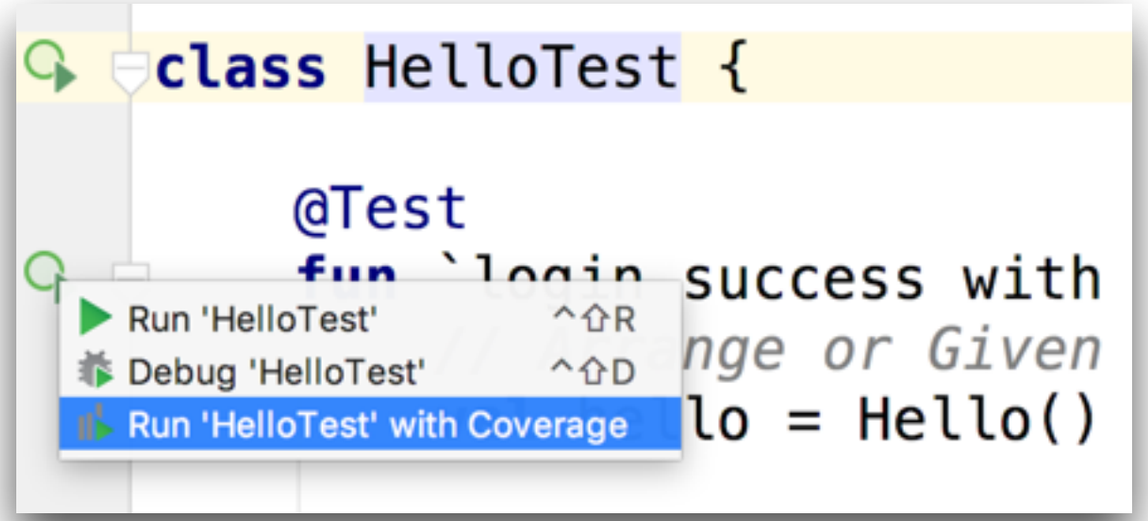
Code coverage 100% ?



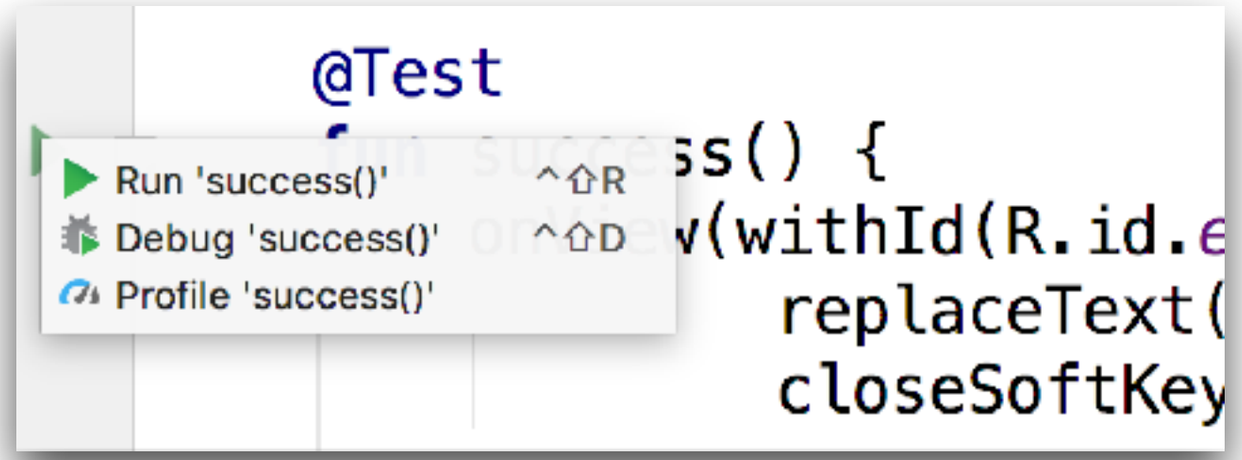
Code coverage for android

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 getDefaultProg
    }
}
```

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/deb
    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/connect
    ])
}
```

4. Run test with code coverage

$gradlew clean jacocoTestReport

Result in /build/reports/jacoco/jacocoTestReport

app > demo.loginapplication

demo.loginapplication

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines
MainActivity		82%		50%	7	13	1	13
Hello		100%		n/a	0	2	0	2
Total	18 of 119	84%	7 of 14	50%	7	15	1	15

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("somkiat@xxx.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. }
```

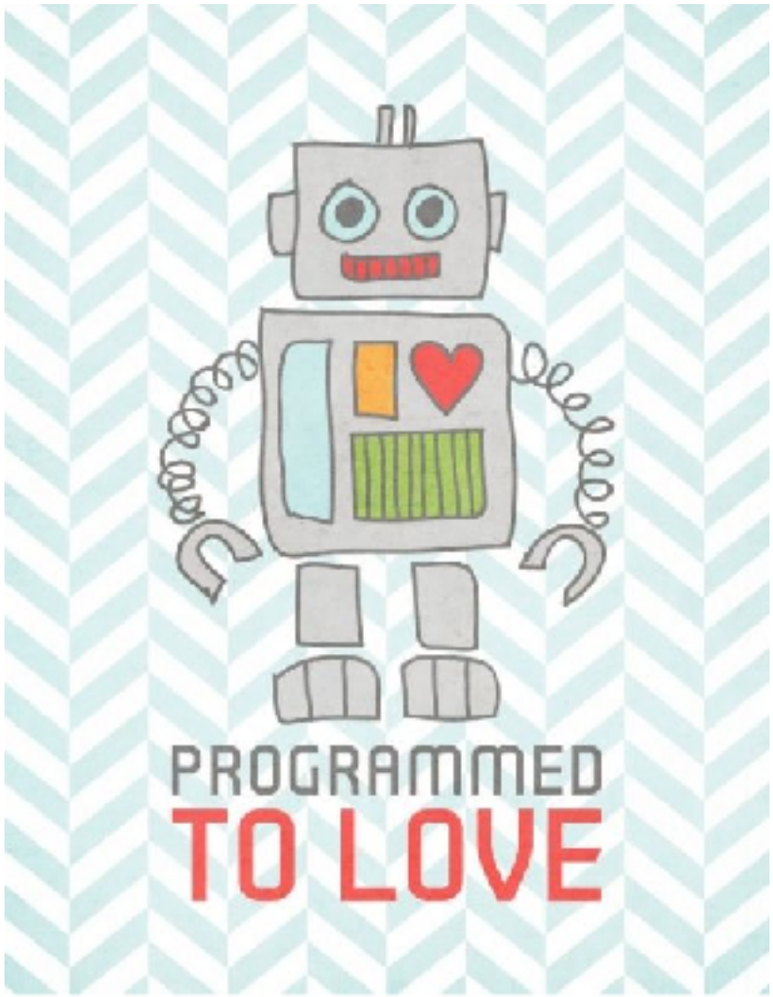
Miss in branch coverage

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

```kotlin
@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 …