Unit Testing

JUnit, Android Unit tests, and Mockito

Patrick Wood

Why do I need unit testing?

- Tests can (but don't have to) be run in a local environment rather than on an emulated android device
- Scales better than manual testing
- More thorough than manual testing
- Can be designed before the app is even programmed (test driven development)

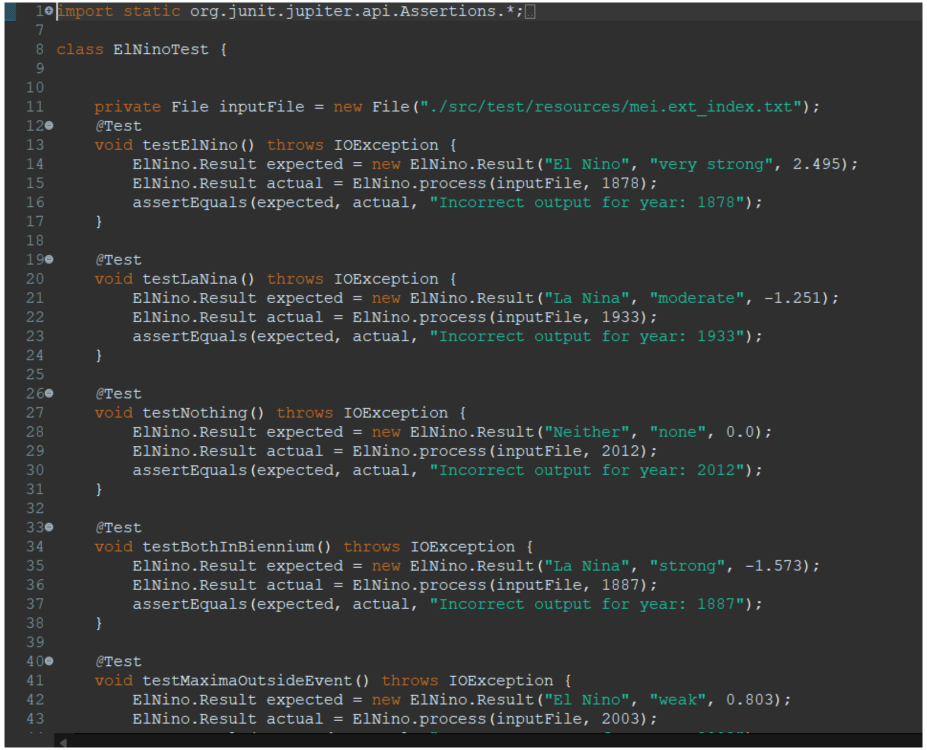
But, like, do I really need to?

- Yes
- Logcat and print statements are great, but they are not nearly as versatile as unit tests
- They have to be observed during the running of the application
- They are not scalable AT ALL

```
com....pets_javat                  40 frames!  The applicati                  on its main t
com....pets_java                   t has changed
com....pets_ja                       is not null
com....pets_                           mes and files
com....pets                              lder: position=0
com....pet                                  tion=0
com....pet                            /raw.githubusercontent.com/CNUC         ster/p
com....pe          tuden                w.githubusercontent.com/CNUCla        ter/p
com....p          student                    tion=0
com....p         _student  D
com....          _student  I  Qu                  usercontent.com/CNUClass    er/p
com....          _student  I  Quer                 ercontent.com/CNUClass     er/p
com....          _student  D  onBind
com....          _student  D  ERROR IMG
com....p         _student  I  Querying: ht             t.com/CNUClass         er/p
com....p          student  D  New Data
com....p          student  D  onBindViewHolder:
com....pe          tudent  D      BINDING: positi
com....pe            dent  D  New Data
com....pets           ent  D  onBindViewHolder: positi
com....pets_            t  D      BINDING: position=0
com....pets_j              I  Querying: https://raw.githubus             /master/p
com....pets_ja                onBindViewHolder: position=0
com....pets_java                  BINDING: position=0
com....pets_javat
com....pets_javathre
com....pets_javathread
```

What is a test?

- A test is a section of code that exists to test another specific section of code
- These can test other small sections of code in isolation or entire systems working together



```java
import static org.junit.jupiter.api.Assertions.*;

class ElNinoTest {


    private File inputFile = new File("./src/test/resources/mei.ext_index.txt");
    @Test
    void testElNino() throws IOException {
        ElNino.Result expected = new ElNino.Result("El Nino", "very strong", 2.495);
        ElNino.Result actual = ElNino.process(inputFile, 1878);
        assertEquals(expected, actual, "Incorrect output for year: 1878");
    }

    @Test
    void testLaNina() throws IOException {
        ElNino.Result expected = new ElNino.Result("La Nina", "moderate", -1.251);
        ElNino.Result actual = ElNino.process(inputFile, 1933);
        assertEquals(expected, actual, "Incorrect output for year: 1933");
    }

    @Test
    void testNothing() throws IOException {
        ElNino.Result expected = new ElNino.Result("Neither", "none", 0.0);
        ElNino.Result actual = ElNino.process(inputFile, 2012);
        assertEquals(expected, actual, "Incorrect output for year: 2012");
    }

    @Test
    void testBothInBiennium() throws IOException {
        ElNino.Result expected = new ElNino.Result("La Nina", "strong", -1.573);
        ElNino.Result actual = ElNino.process(inputFile, 1887);
        assertEquals(expected, actual, "Incorrect output for year: 1887");
    }

    @Test
    void testMaximaOutsideEvent() throws IOException {
        ElNino.Result expected = new ElNino.Result("El Nino", "weak", 0.803);
        ElNino.Result actual = ElNino.process(inputFile, 2003);
```

Test classification

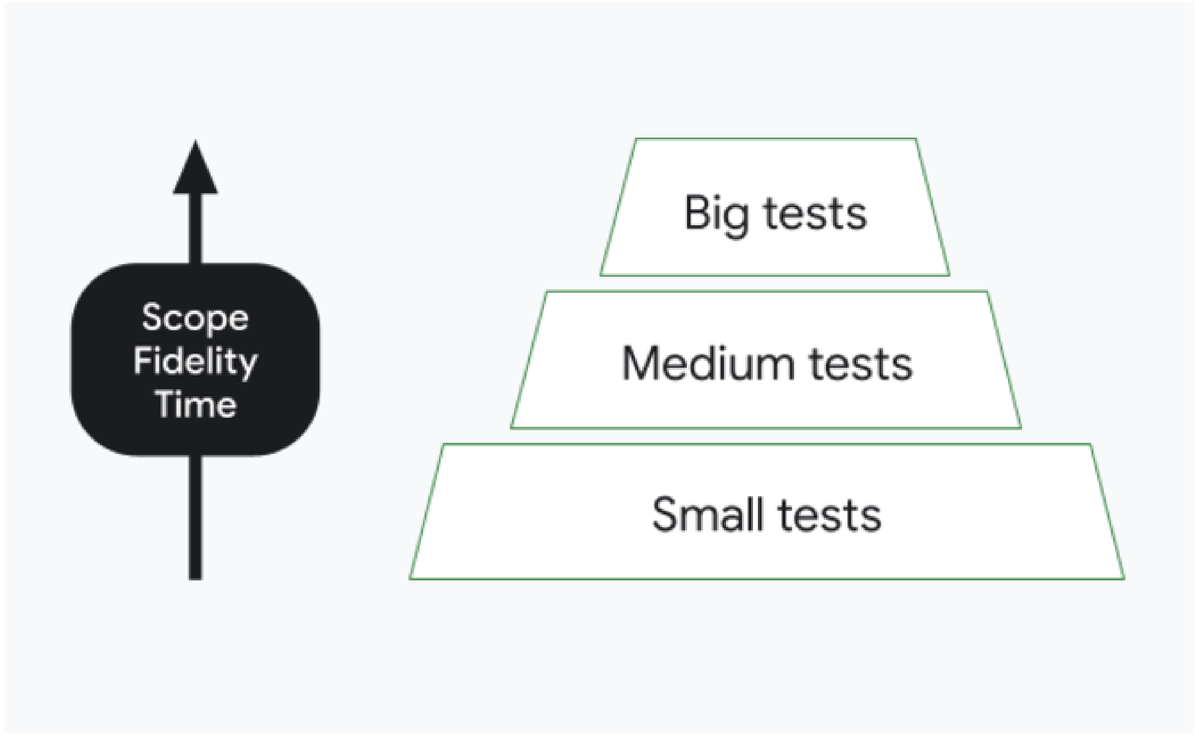
Subject:

- Functionality: does it work like it’s supposed to?
- Performance: does it work smoothly and quickly?
- Accessibility: is it easy to use?
- Compatibility: does it work on different devices?

Scope:

- Unit (Small) Tests
- Integration (Medium) Tests
- End-to-End (Big) Tests

How it all fits together



Testing strategies (23 Oct. 2024) https://developer.android.com/training/testing/fundamentals/strategies

Types of Android Test

Local Tests

- Run through the Java Virtual Machine (JVM)
- Allows for evaluation of internal logic much more quickly
- (Usually logic tests)

Instrumented Tests

- Run on an Android device, either physical or emulated
- Builds the app alongside a **test app** that interfaces with the application
- (Usually UI Tests)

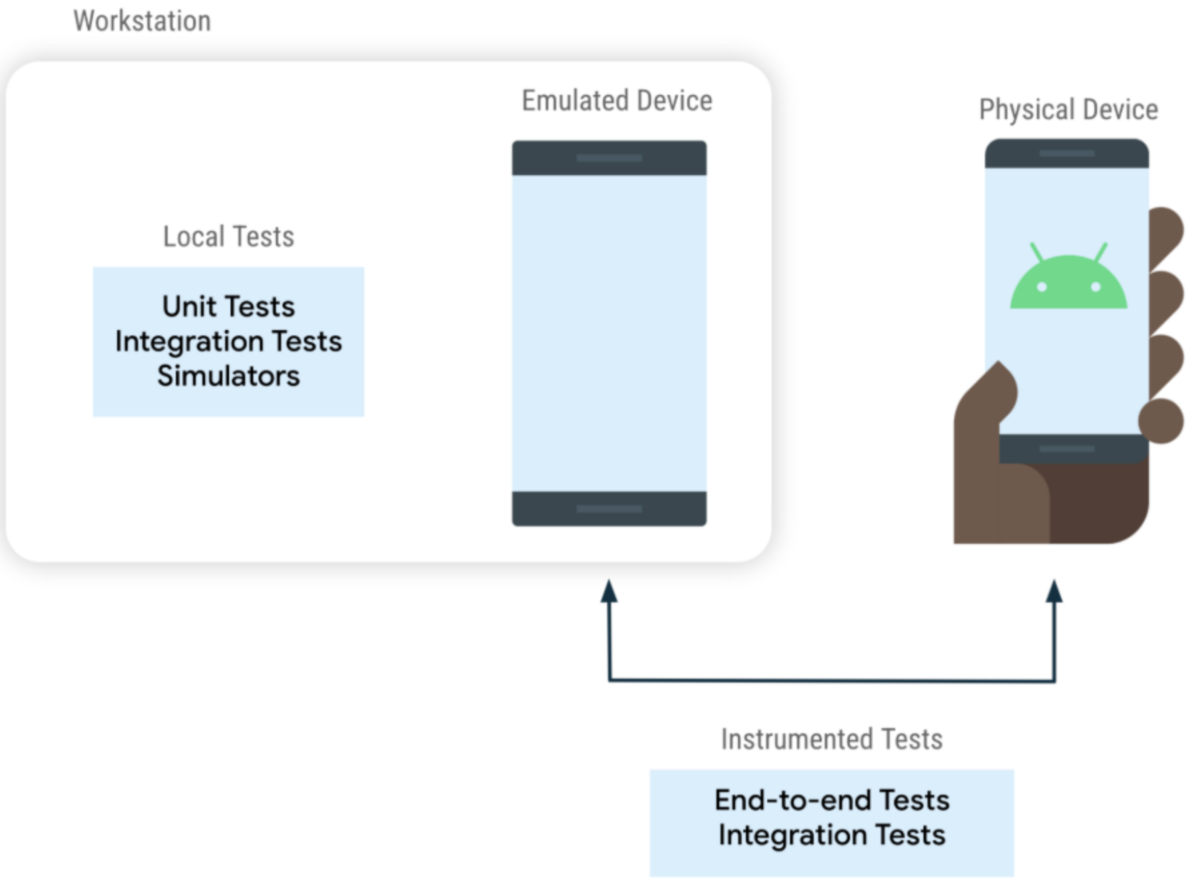


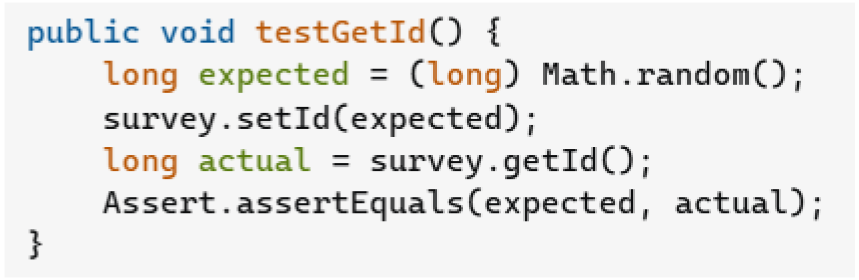
***Figure 2**: Different types of tests depending on where they run.*

Fundamentals of testing Android apps, (23 Oct, 2024) https://developer.android.com/training/testing/fundamentals#espresso

Local Tests

Local tests are typically done with JUnit, the same way testing is done on other applications

JUnit tests are typically paired with mocks (more on them later)



```
public void testGetId() {
    long expected = (long) Math.random();
    survey.setId(expected);
    long actual = survey.getId();
    Assert.assertEquals(expected, actual);
}
```

Instrument Tests

Instrument tests are typically conducted with an external library like Espresso

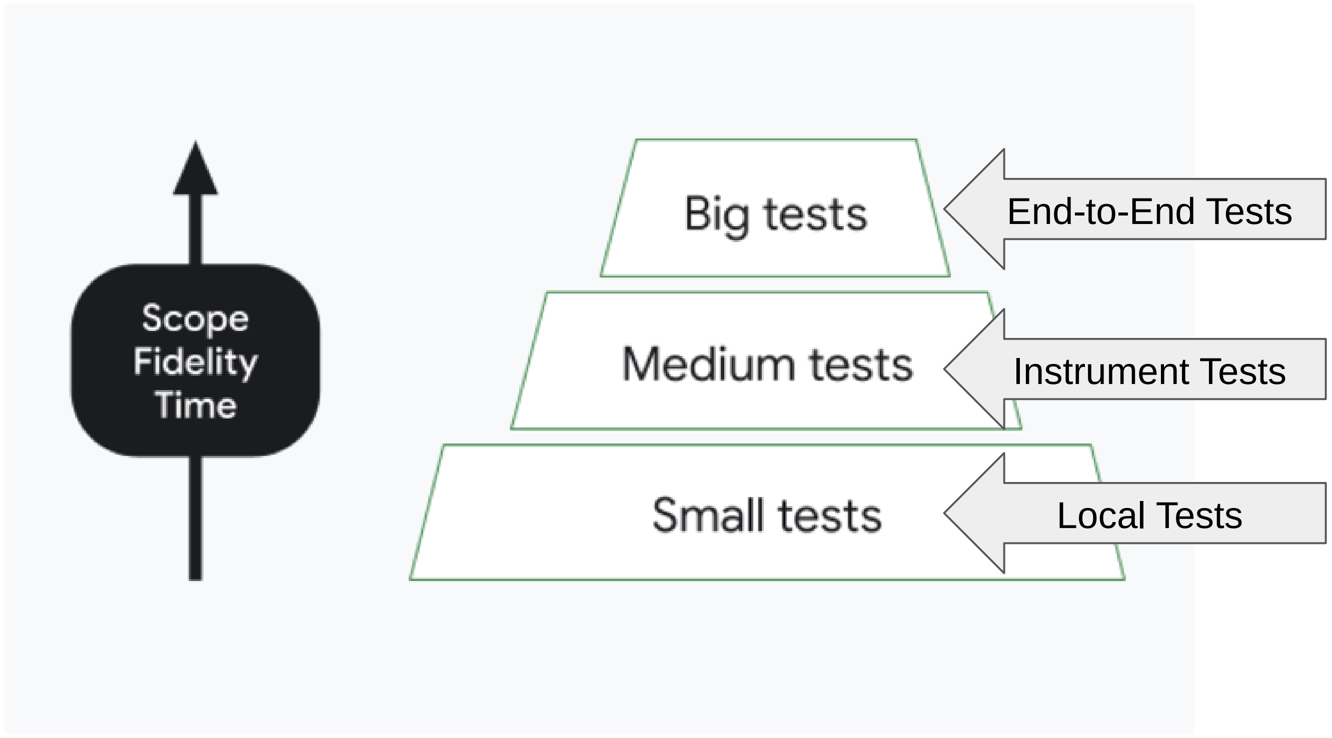
Espresso is used to automatically interact with the application

```
@Test
public void greeterSaysHello() {
    onView(withId(R.id.name_field)).perform(typeText("Steve"));
    onView(withId(R.id.greet_button)).perform(click());
    onView(withText("Hello Steve!")).check(matches(isDisplayed()));
}
```

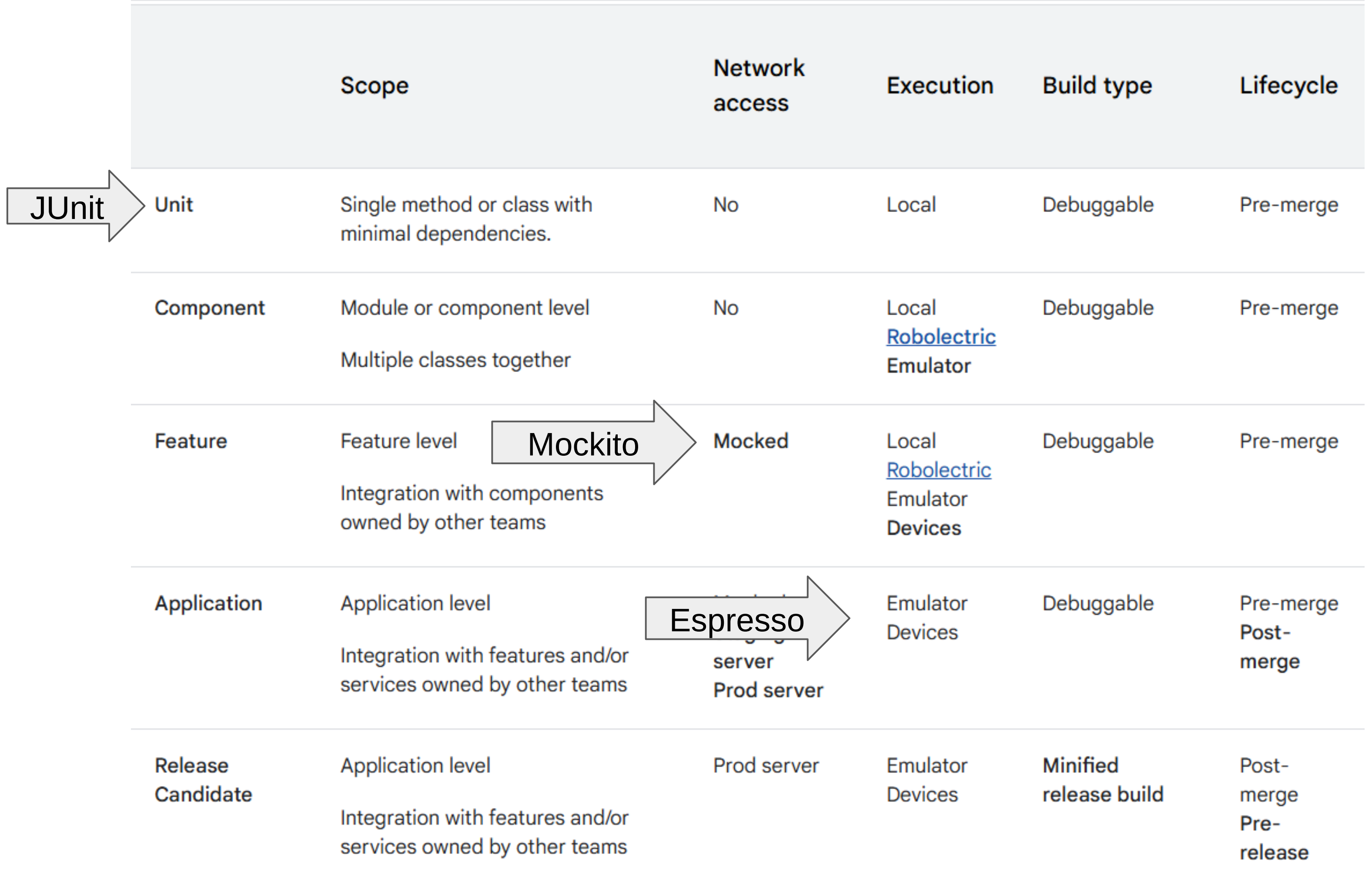
Mocks (Mockito)

- Mocks are used to simulate dependencies in Unit testing
- Typically paired with local JUnit tests
- Simulate things like context so that the local tests can interact with the android logic
- This allows you to test just your objects in isolation

How it all fits together



Testing strategies (23 Oct. 2024) https://developer.android.com/training/testing/fundamentals/strategies


JUnit
Mockito
Espresso

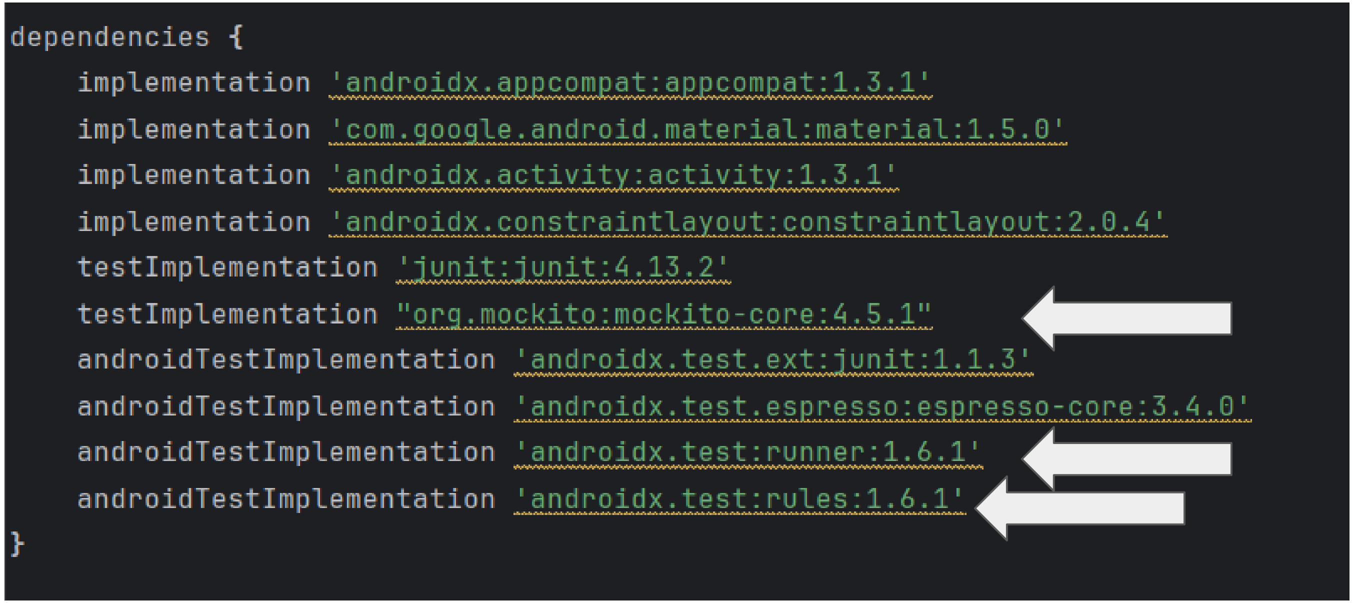
	Scope	Network access	Execution	Build type	Lifecycle
Unit	Single method or class with minimal dependencies.	No	Local	Debuggable	Pre-merge
Component	Module or component level Multiple classes together	No	Local Robolectric **Emulator**	Debuggable	Pre-merge
Feature	Feature level Integration with components owned by other teams	**Mocked**	Local Robolectric Emulator **Devices**	Debuggable	Pre-merge
Application	Application level Integration with features and/or services owned by other teams	Mocked **Staging server** **Prod server**	Emulator Devices	Debuggable	Pre-merge **Post-merge**
Release Candidate	Application level Integration with features and/or services owned by other teams	Prod server	Emulator Devices	**Minified release build**	Post-merge **Pre-release**

Testing strategies (23 Oct. 2024) https://developer.android.com/training/testing/fundamentals/strategies

Ok, but how do I actually do it?

Set Up Dependencies

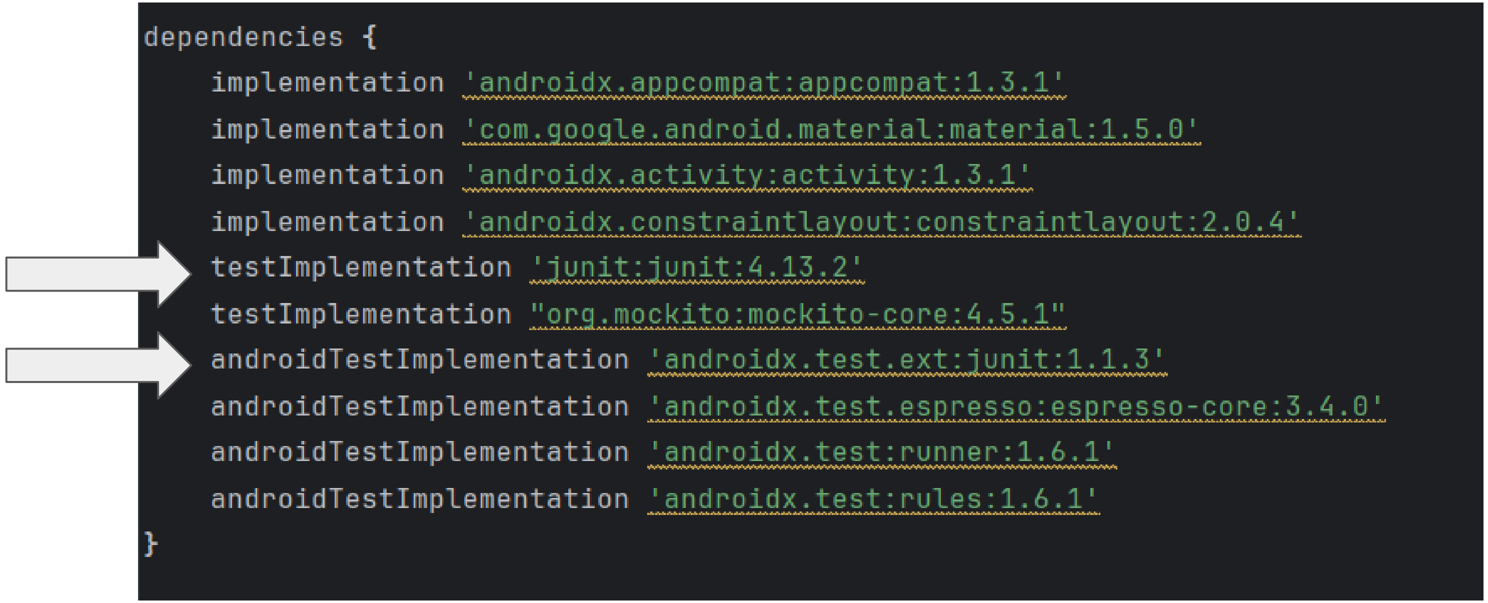
Most “New Activity” formats already have most of the testing dependencies set up



```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.activity:activity:1.3.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.13.2'
    testImplementation "org.mockito:mockito-core:4.5.1"
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    androidTestImplementation 'androidx.test:runner:1.6.1'
    androidTestImplementation 'androidx.test:rules:1.6.1'
}
```

Set Up Dependencies

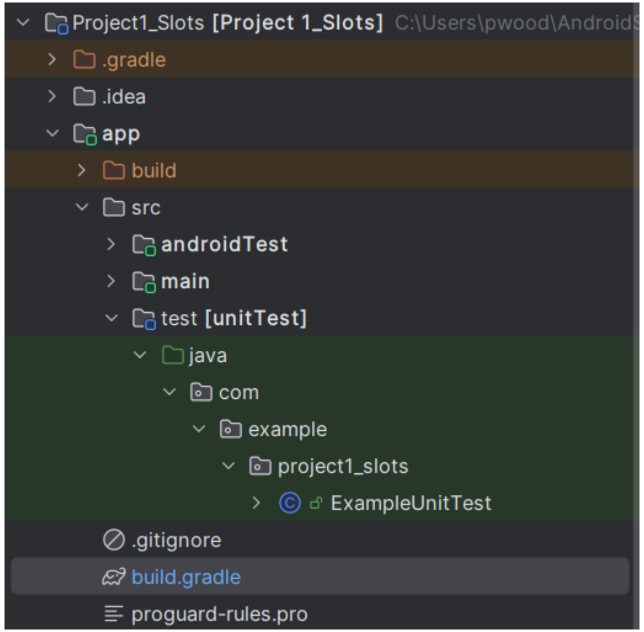
Note the Difference between “testImplementation” and “androidTestImplementation”



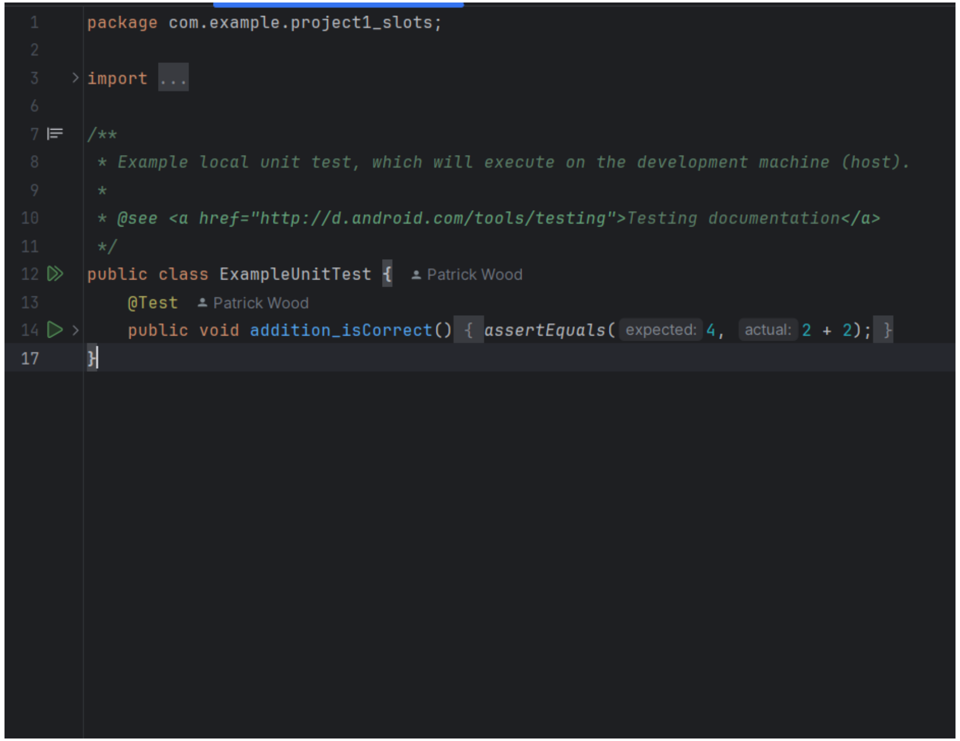
```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.activity:activity:1.3.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.13.2'
    testImplementation "org.mockito:mockito-core:4.5.1"
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    androidTestImplementation 'androidx.test:runner:1.6.1'
    androidTestImplementation 'androidx.test:rules:1.6.1'
}
```

Create a test file

Android studio is nice and does this for us most of the time



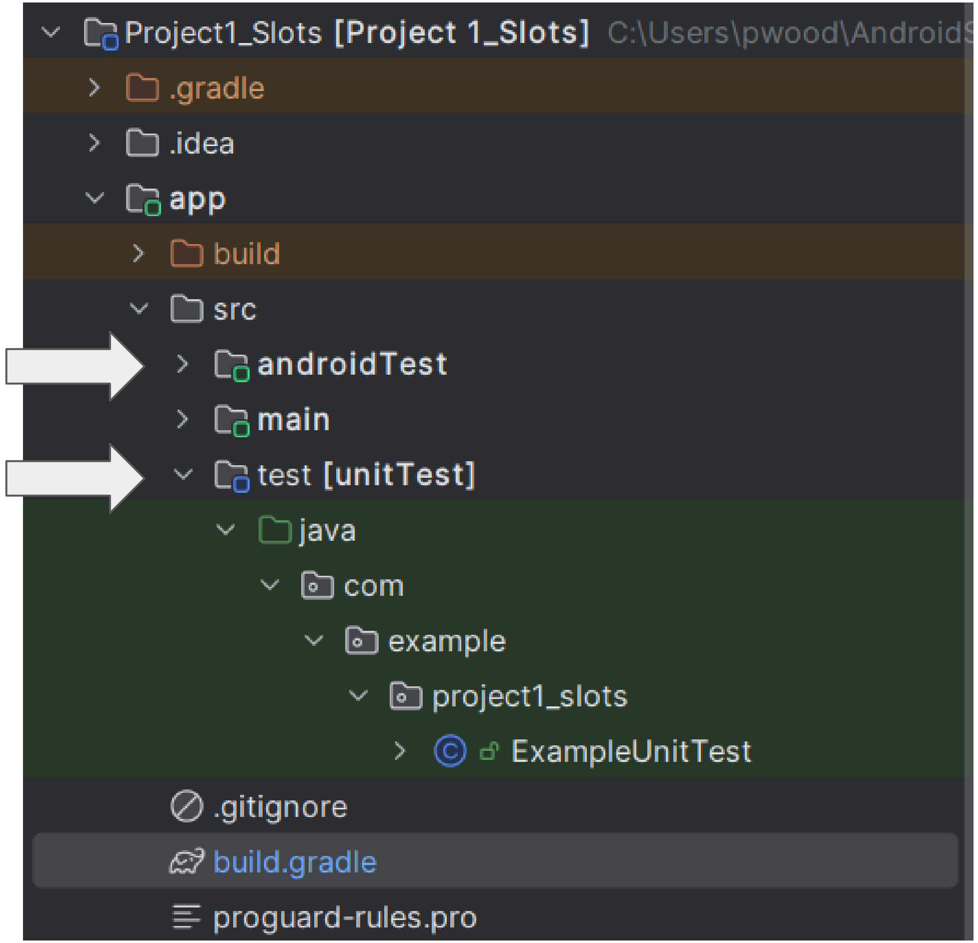
```
Project1_Slots [Project 1_Slots] C:\Users\pwood\Android
  .gradle
  .idea
  app
    build
    src
      androidTest
      main
      test [unitTest]
        java
          com
            example
              project1_slots
                ExampleUnitTest
    .gitignore
    build.gradle
    proguard-rules.pro
```



```java
package com.example.project1_slots;

import ...

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
public class ExampleUnitTest {  Patrick Wood
    @Test  Patrick Wood
    public void addition_isCorrect() { assertEquals( expected: 4,  actual: 2 + 2); }
}
```

Create a test file

Again, note the difference between “androidTest” and “test [unitTest]”

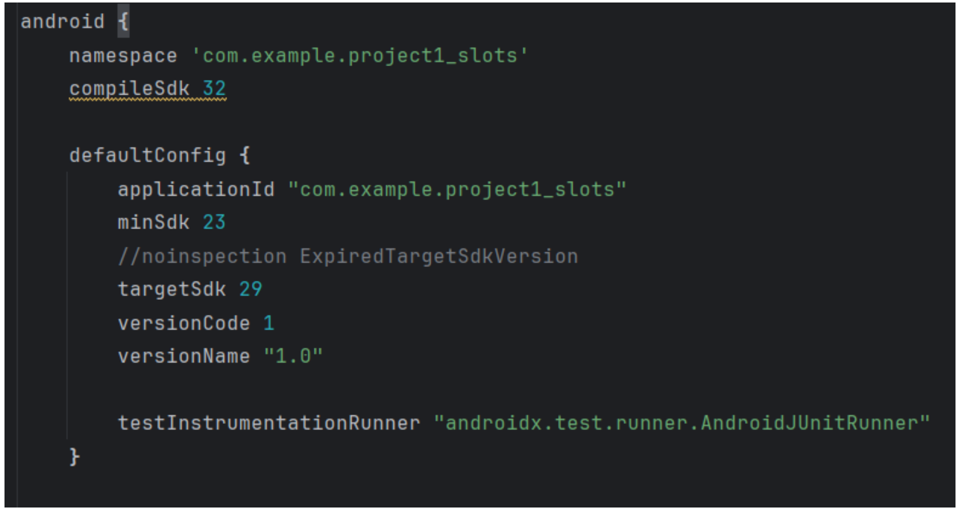


```
Project1_Slots [Project 1_Slots] C:\Users\pwood\Android
  .gradle
  .idea
  app
    build
    src
      androidTest
      main
      test [unitTest]
        java
          com
            example
              project1_slots
                ExampleUnitTest
    .gitignore
    build.gradle
    proguard-rules.pro
```

Set up Espresso runner

You may also have to edit the defaultConfig property to include a testInstrumentationRunner

Sometimes this will be generated for you (it was for me)

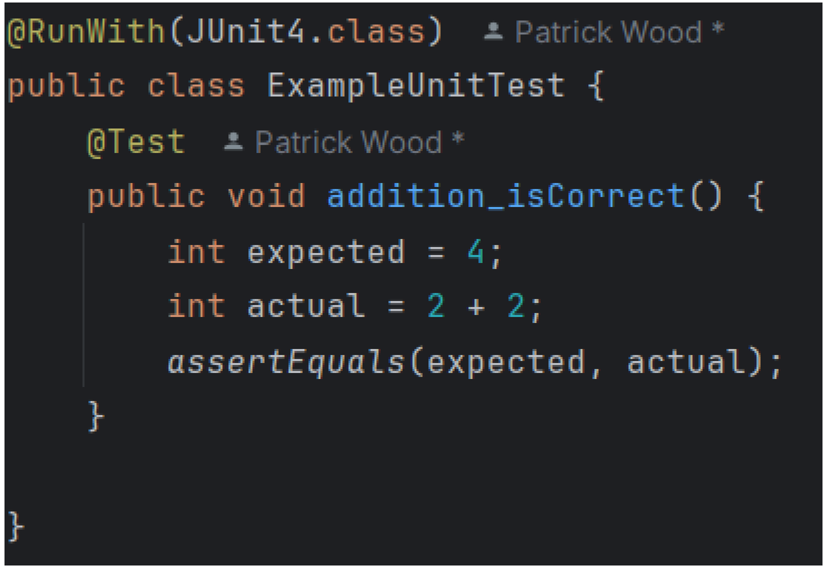


```
android {
    namespace 'com.example.project1_slots'
    compileSdk 32

    defaultConfig {
        applicationId "com.example.project1_slots"
        minSdk 23
        //noinspection ExpiredTargetSdkVersion
        targetSdk 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
```

Local Test Structure

Local tests are typically made up of three stages:

- Create/Declare expected value
- Calculate actual value
- Assert equals

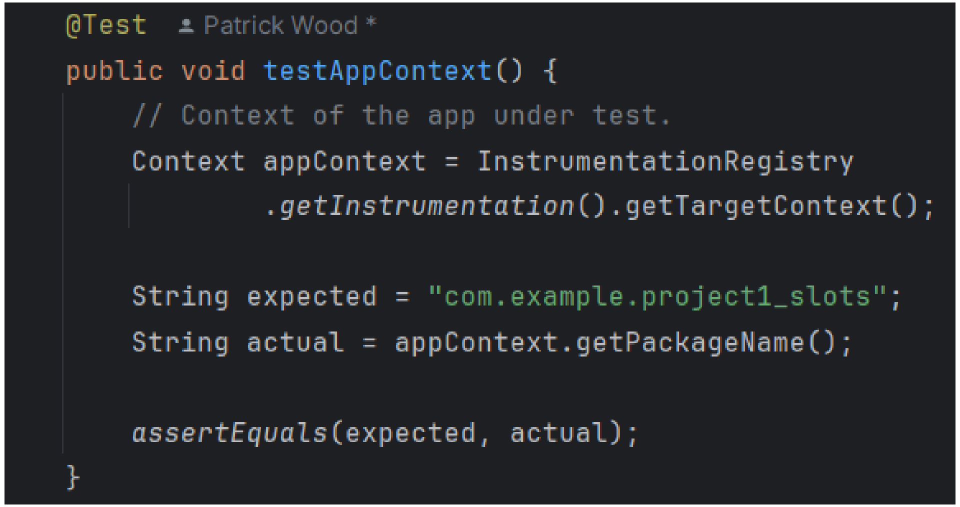


```
@RunWith(JUnit4.class)  Patrick Wood *
public class ExampleUnitTest {
    @Test  Patrick Wood *
    public void addition_isCorrect() {
        int expected = 4;
        int actual = 2 + 2;
        assertEquals(expected, actual);
    }

}
```

Instrumented Test Structure

Instrumented tests are similar, but are done using behaviors instead

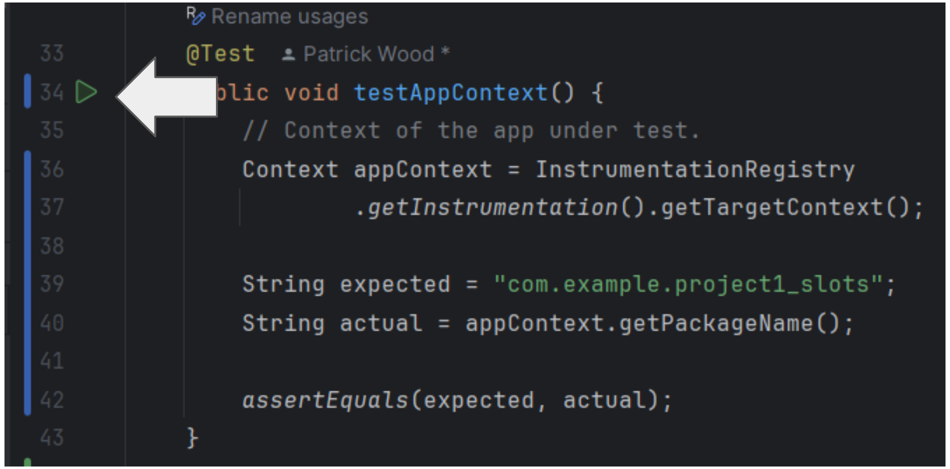
- Determine desired behavior
- Use the testing application to simulate an action and capture the result from the action
- Check to see if the resulting behavior matches the desired behavior



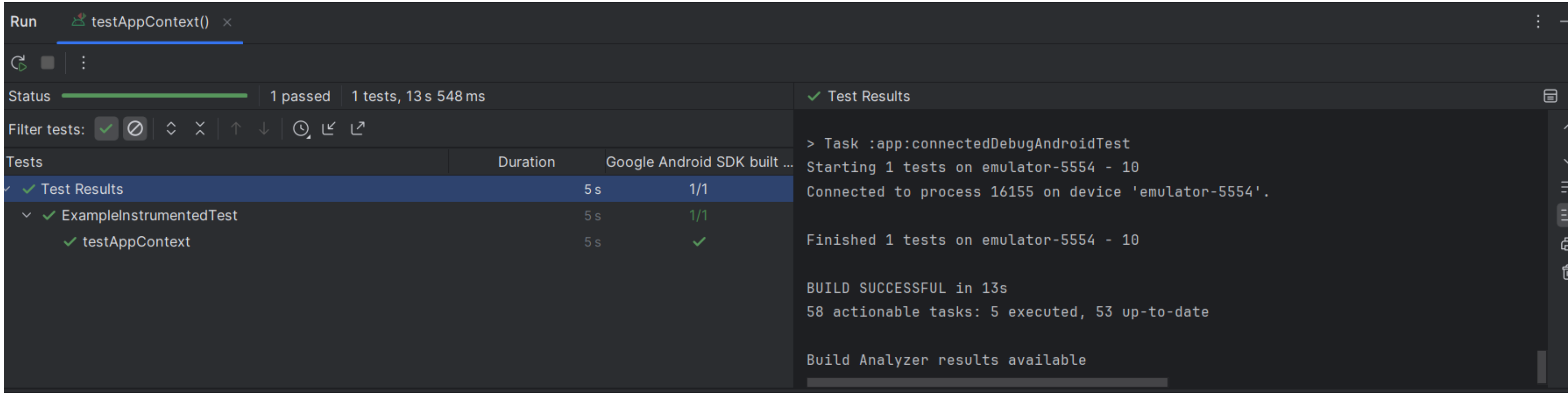
```java
@Test  Patrick Wood *
public void testAppContext() {
    // Context of the app under test.
    Context appContext = InstrumentationRegistry
            .getInstrumentation().getTargetContext();

    String expected = "com.example.project1_slots";
    String actual = appContext.getPackageName();

    assertEquals(expected, actual);
}
```

Running Tests

Tests can either be run through the terminal or through the GUI



```
@Test
public void testAppContext() {
    // Context of the app under test.
    Context appContext = InstrumentationRegistry
            .getInstrumentation().getTargetContext();

    String expected = "com.example.project1_slots";
    String actual = appContext.getPackageName();

    assertEquals(expected, actual);
}
```



Tests	Duration	Google Android SDK built ...
Test Results	5 s	1/1
ExampleInstrumentedTest	5 s	1/1
testAppContext	5 s	✓

```
> Task :app:connectedDebugAndroidTest
Starting 1 tests on emulator-5554 - 10
Connected to process 16155 on device 'emulator-5554'.

Finished 1 tests on emulator-5554 - 10

BUILD SUCCESSFUL in 13s
58 actionable tasks: 5 executed, 53 up-to-date

Build Analyzer results available
```

Live Demonstration!

Sources

Build Local Unit Tests (23 Oct. 2024) https://developer.android.com/training/testing/local-tests#java

Testing strategies (23 Oct. 2024) https://developer.android.com/training/testing/fundamentals/strategies

Fundamentals of testing Android apps, (23 Oct, 2024)

https://developer.android.com/training/testing/fundamentals#espresso

Espresso (23 Oct, 2024) https://developer.android.com/training/testing/espresso

Test apps on Android (23 Oct 2024) https://developer.android.com/training/testing