CSE2MAD

Mobile Application Development

Lecture 10

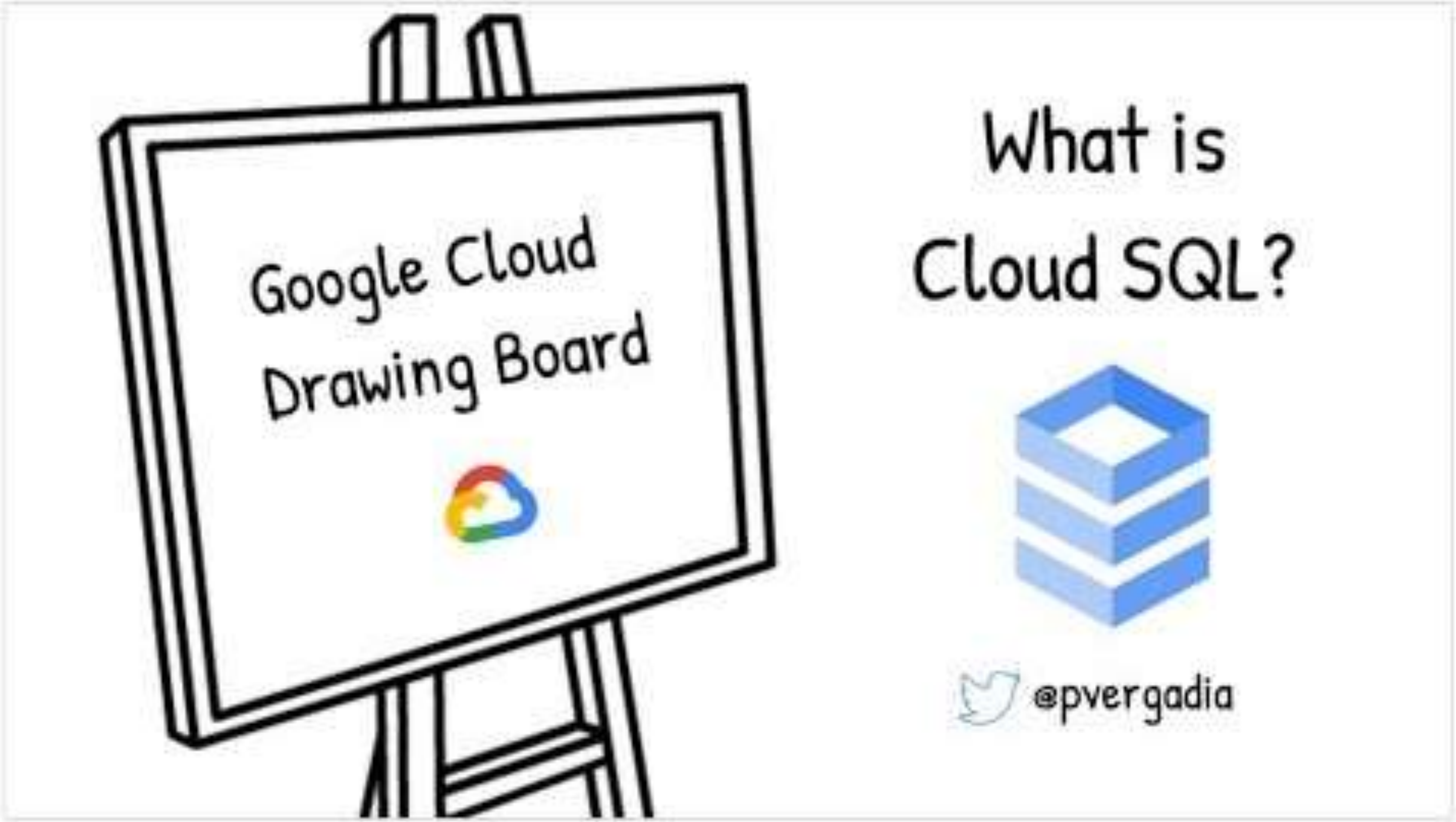
Outline

- Cloud SQL
- Testing in Android Studio
- Runtime Debugging
- Performance Profiling
- Android Privacy
- Usability Testing

Cloud SQL

- If you really need a relational database
- Google Cloud has this capability
  - ➢ MySQL
  - ➢ PostgreSQL
  - ➢ SQL Server
- Can be utilised by your Android App

https://cloud.google.com/sql/docs/



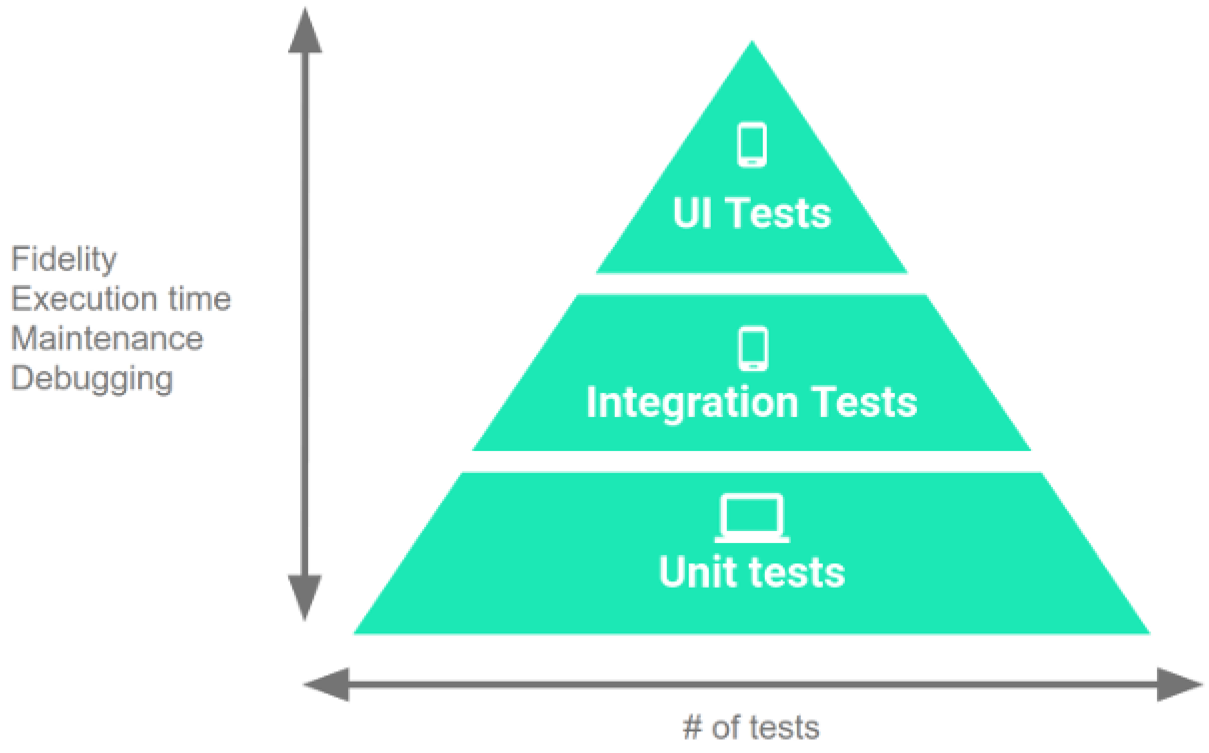
https://youtu.be/Kl8ig2BtLAY

Testing in Android Studio

- Important and necessary for all applications to undergo code-level testing
- Overview: https://developer.android.com/studio/test
- Unit Tests, Instrumental Tests, Interface Tests
  - Let’s see what they are and how we can execute them

Fundamentals of Testing

- Small tests are **unit tests** that validate your app's behavior one class at a time.
- Medium tests are **integration tests** that validate either interactions between levels of the stack within a module, or interactions between related modules.
- Large tests are end-to-end tests that validate user journeys spanning multiple modules of your app.



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

Write small tests

Unit Test: A unit is the smallest testable part of your system; this may be a small as a method in your Java classes. Testing is conducted to validate the unit functions as expected.

Unit Testing is performed by software developers, as they are most familiar with the code and the expected functionality.

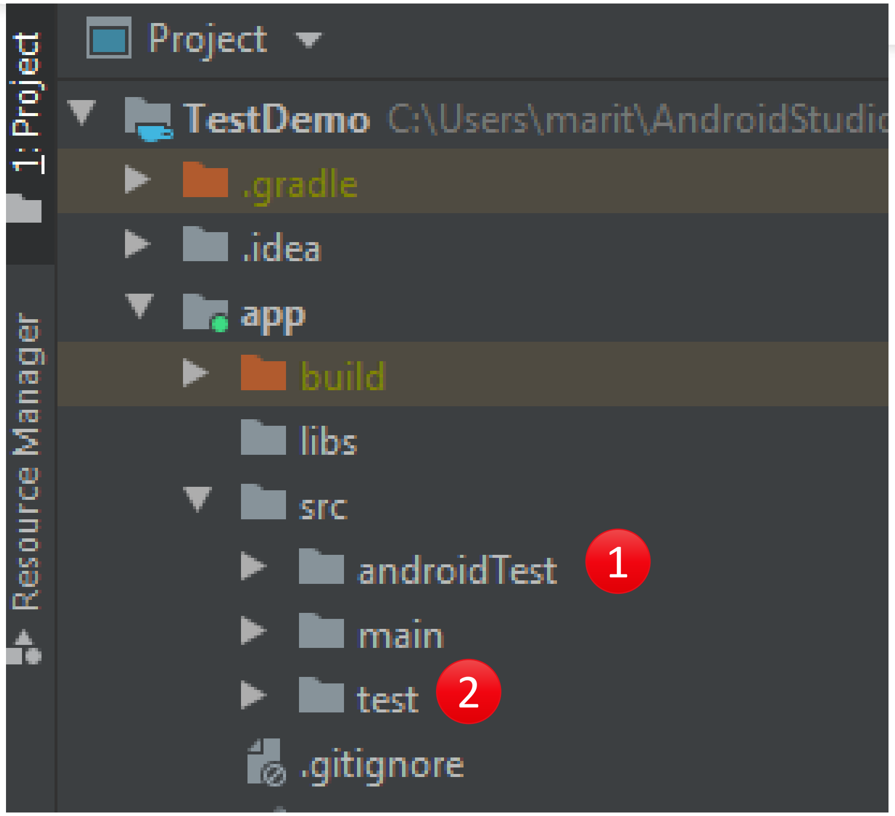
This is known as the white-box method.

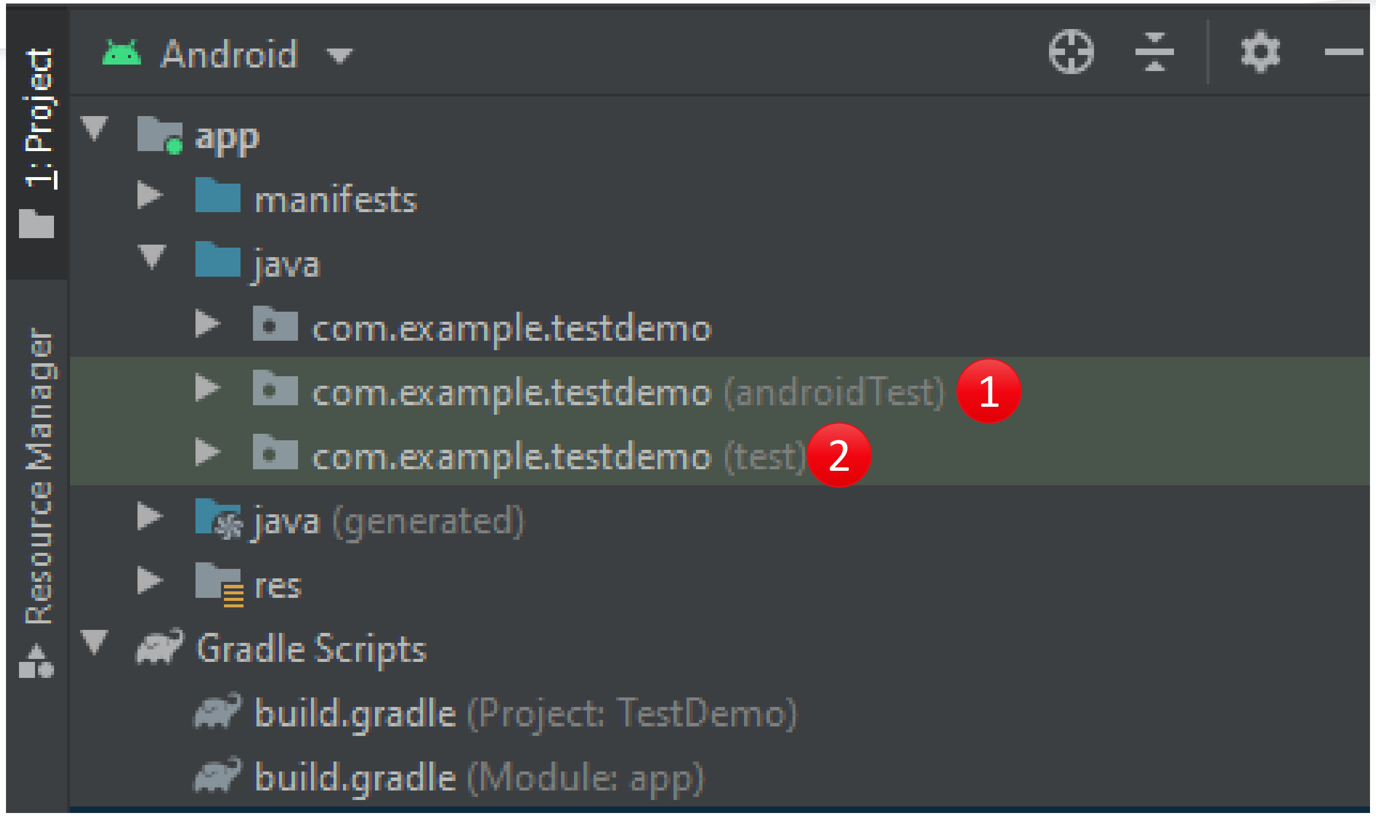
- Logical stages involve
    - Unit Test Plan
        - Understand what needs to be tested, what inputs and outputs are expected
- Unit Test Case
    - Code the test
- Unit Test
    - Perform the test

Unit Testing in Android Studio

- Do I need to unit test everything?
  - ➢ Generally No. Focus on portions of your code that deliver key functional outcomes to your app
- Make sure your unit tests don’t overlap
- Exercise major control flow branch conditions in your code
- Separate the dev environment from your code
  - ➢ No machine specific or OS specific dependencies
- To make your code easier to test, develop your code in terms of modules, where each module represents a specific task that users complete within your app.

Unit Testing in Android Studio




1: Project
Resource Manager
Android
app
manifests
java
com.example.testdemo
com.example.testdemo (androidTest)
1
com.example.testdemo (test)
2
java (generated)
res
Gradle Scripts
build.gradle (Project: TestDemo)
build.gradle (Module: app)

Your project's (1) instrumented tests and (2) local JVM tests are visible in either the Project view (left) or Android view (right).

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

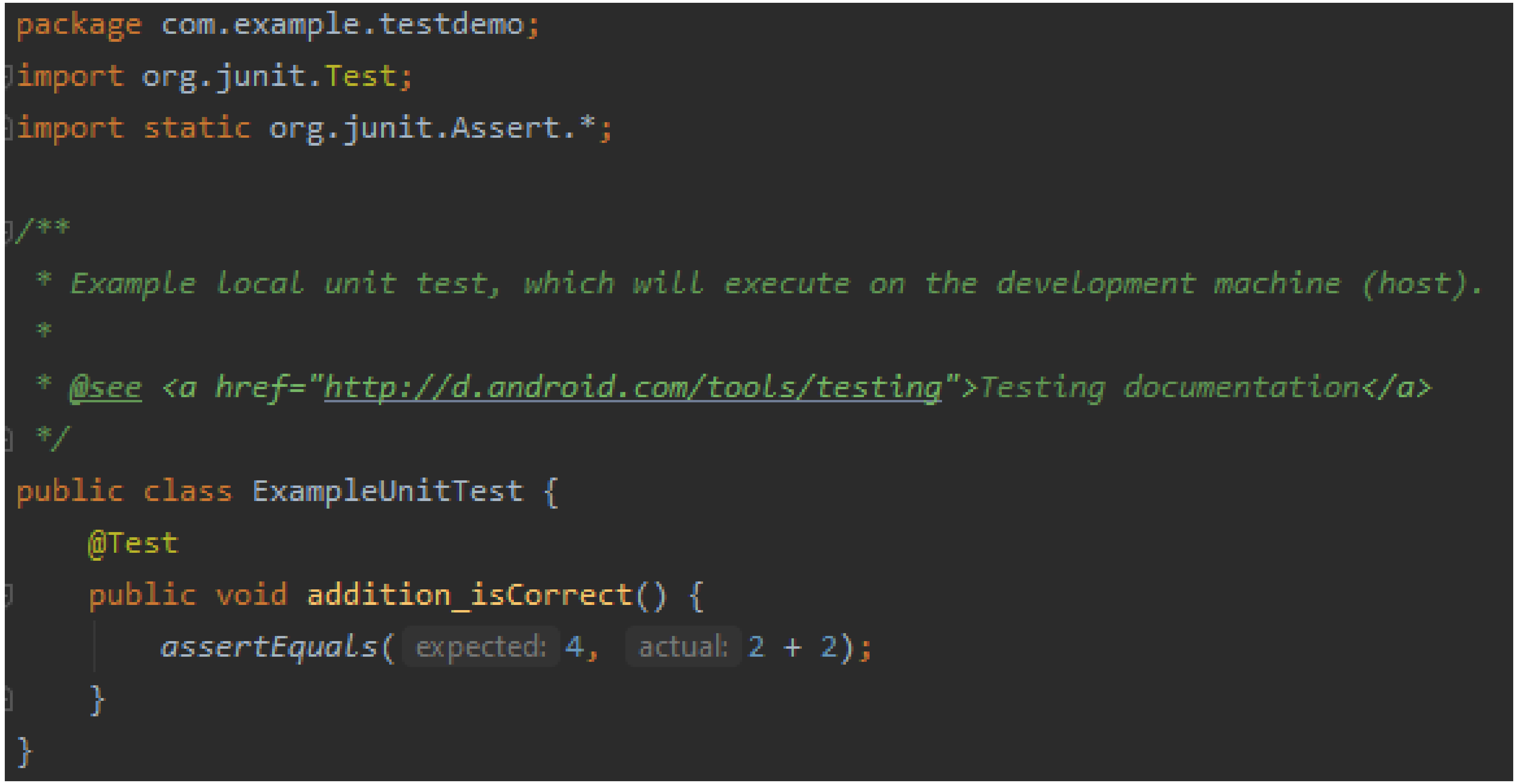
Create a unit test class

1. Open the Java file containing the code you want to test.
2. Click the class or method you want to test, then press Ctrl+Shift+T (⇧⌘T).
3. In the menu that appears, click Create New Test.
4. In the Create Test dialog, edit any fields and select any methods to generate, and then click OK.
5. In the Choose Destination Directory dialog, click the source set corresponding to the type of test you want to create: **androidTest** for an instrumented test or **test** for a local unit test. Then click OK.

Android indicates that these tests should be run on an android device

Default Unit Test

Default Unit Test

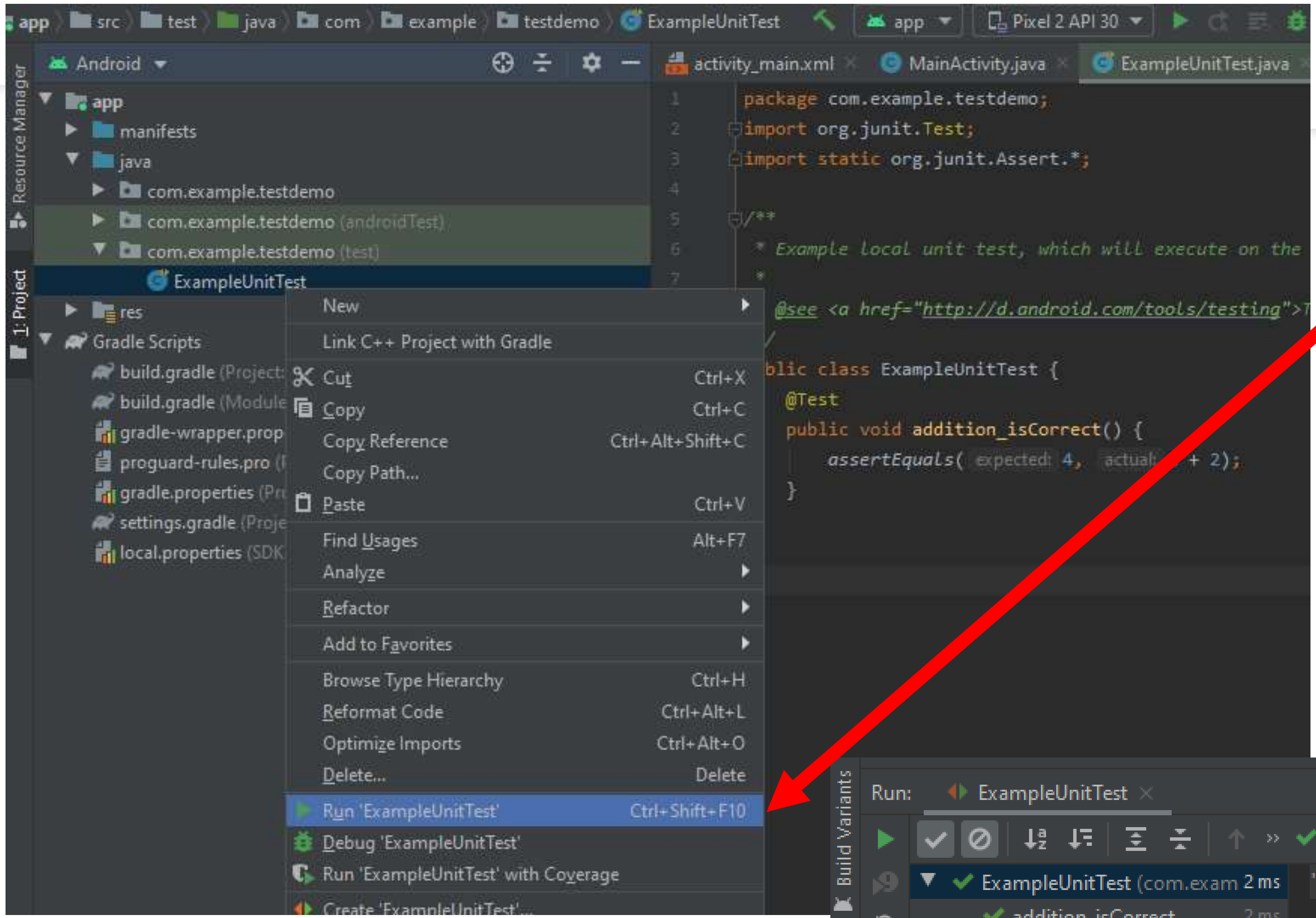


```java
package com.example.testdemo;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * 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 {
    @Test
    public void addition_isCorrect() {
        assertEquals( expected: 4,  actual: 2 + 2);
    }
}
```

https://developer.android.com/training/testing/unit-testing/local-unit-tests#build https://junit.org/junit4/

Run a local unit test class

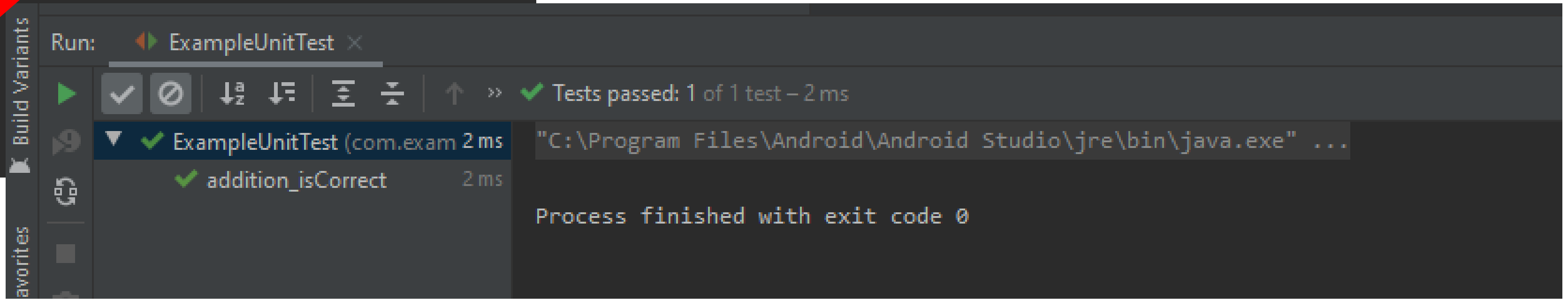
JUnit



```
package com.example.testdemo;
import org.junit.Test;
import static org.junit.Assert.*;

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

The test is run on the local JVM

Result



Run:
ExampleUnitTest
Tests passed: 1 of 1 test – 2 ms
ExampleUnitTest (com.exam 2 ms
addition_isCorrect 2 ms

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...

Process finished with exit code 0
```

https://developer.android.com/training/testing/unit-testing/local-unit-tests#build https://junit.org/junit4/

Run a local unit test class

JUnit

Let’s see what the Junit Test Framework gives us in Android Studio

- Assertion based

assertEquals(4, 2 + 2); → **Evaluates true**

Expected

Actual

Statement	**Description**
fail([message])	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)	Checks that the boolean condition istrue.
assertFalse([message,] boolean condition)	Checks that the boolean condition isfalse.
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be thesame.
assertNull([message,] object)	Checks that the object is null.
assertNotNull([message,] object)	Checks that the object is not null.
assertSame([message,] expected, actual)	Checks that both variables refer to the same object.
assertNotSame([message,] expected, actual)	Checks that both variables refer to different objects.

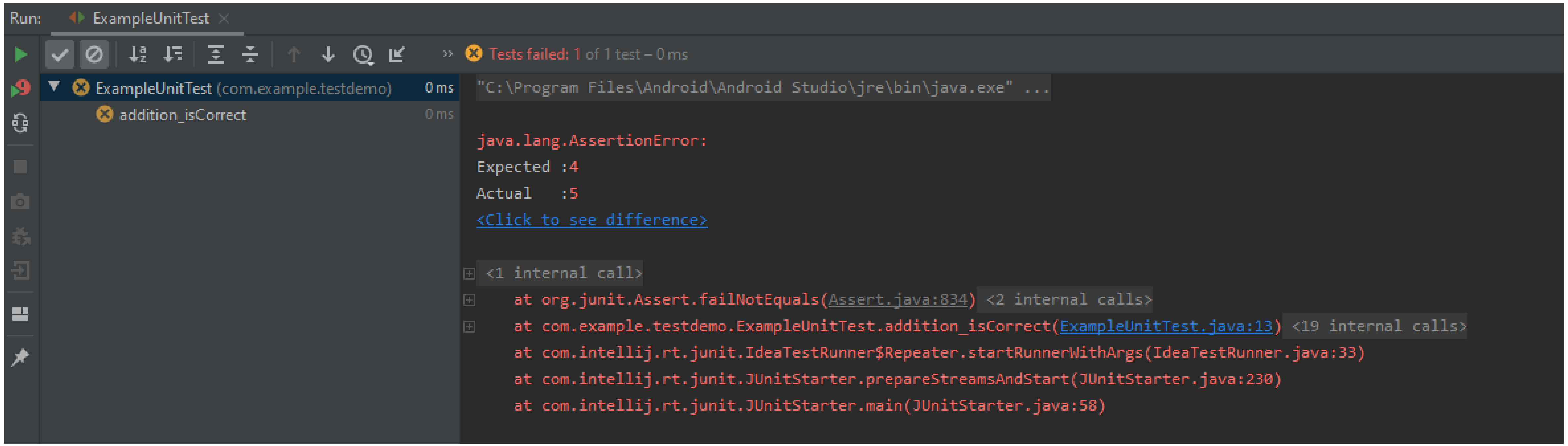
https://www.vogella.com/tutorials/JUnit/article.html#junit_intro

Run a local unit test class

JUnit

Let’s break it!

assertEquals(4, 3 + 2);

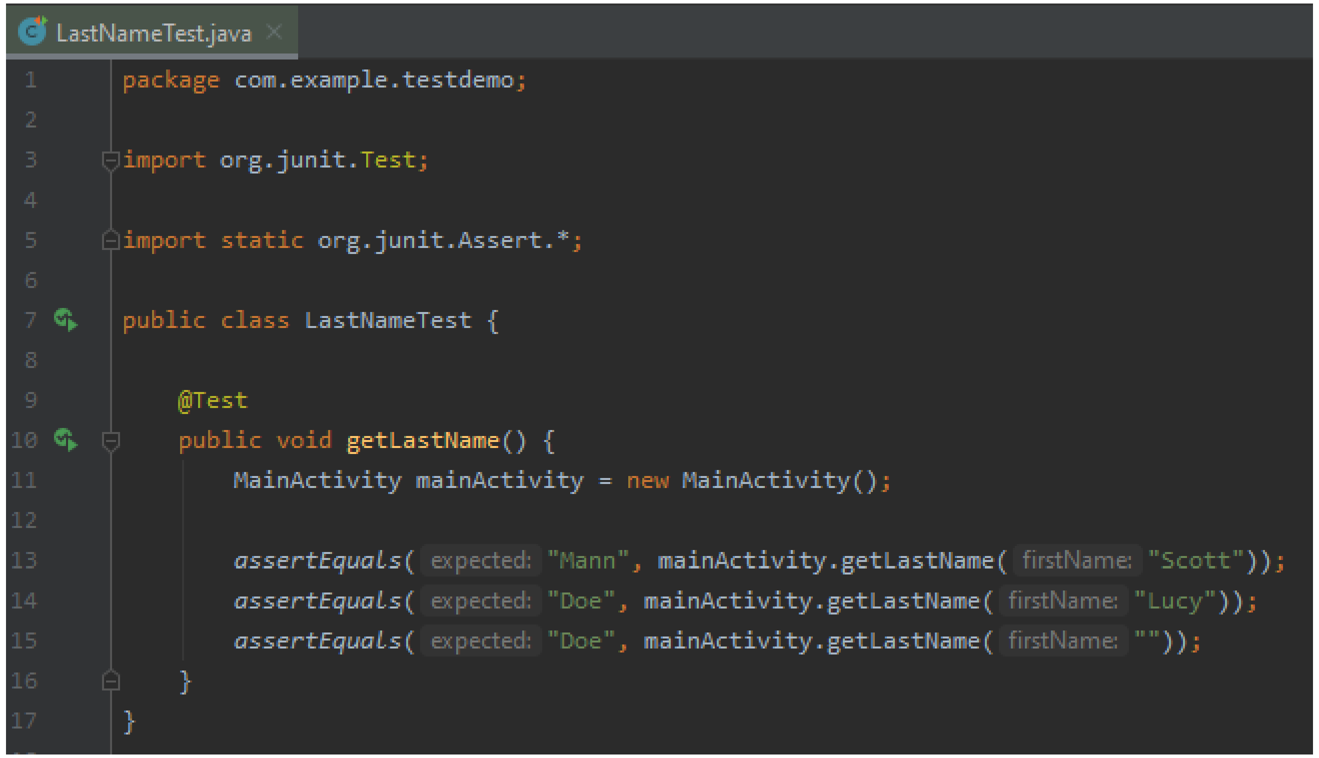

Run: ExampleUnitTest
Tests failed: 1 of 1 test – 0 ms
ExampleUnitTest (com.example.testdemo) 0 ms
addition_isCorrect 0 ms

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...

java.lang.AssertionError:
Expected :4
Actual   :5
<Click to see difference>

<1 internal call>
    at org.junit.Assert.failNotEquals(Assert.java:834) <2 internal calls>
    at com.example.testdemo.ExampleUnitTest.addition_isCorrect(ExampleUnitTest.java:13) <19 internal calls>
    at com.intellij.rt.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:33)
    at com.intellij.rt.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:230)
    at com.intellij.rt.junit.JUnitStarter.main(JUnitStarter.java:58)
```

Unit Testing in Android Studio - Coverage

JUnit

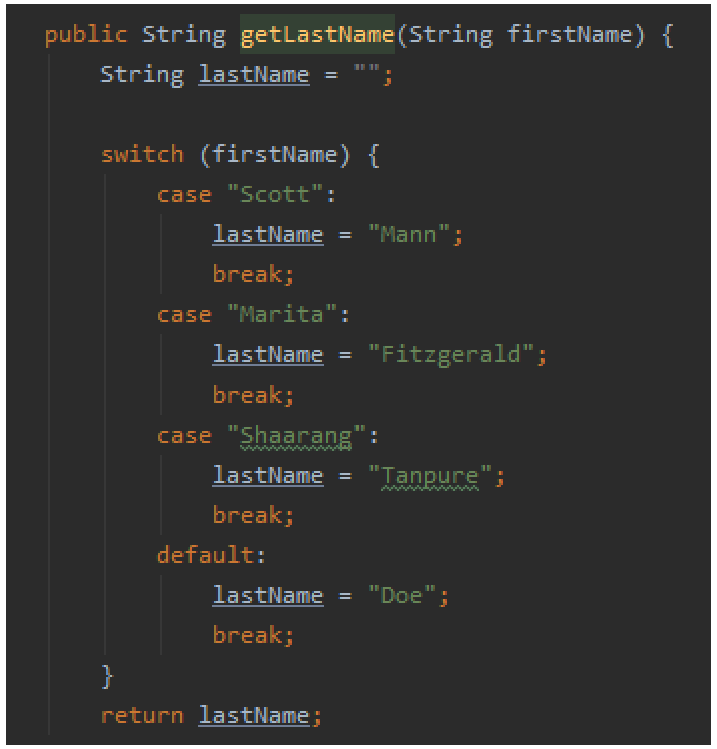
LastNameTest.java



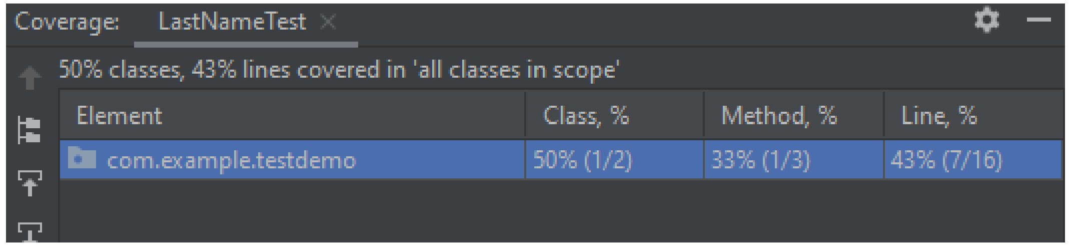
LastNameTest.java

```java
package com.example.testdemo;

import org.junit.Test;

import static org.junit.Assert.*;

public class LastNameTest {

    @Test
    public void getLastName() {
        MainActivity mainActivity = new MainActivity();

        assertEquals( expected: "Mann", mainActivity.getLastName( firstName: "Scott"));
        assertEquals( expected: "Doe", mainActivity.getLastName( firstName: "Lucy"));
        assertEquals( expected: "Doe", mainActivity.getLastName( firstName: ""));
    }
}
```

MainActivity.java



```java
public String getLastName(String firstName) {
    String lastName = "";

    switch (firstName) {
        case "Scott":
            lastName = "Mann";
            break;
        case "Marita":
            lastName = "Fitzgerald";
            break;
        case "Shaarang":
            lastName = "Tanpure";
            break;
        default:
            lastName = "Doe";
            break;
    }
    return lastName;
```



Coverage: LastNameTest

50% classes, 43% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
com.example.testdemo	50% (1/2)	33% (1/3)	43% (7/16)

https://developer.android.com/studio/test#view_test_coverage https://www.jetbrains.com/help/idea/running-test-with-coverage.html

Instrumented Unit Tests in Android Studio

- Instrumented unit tests are tests that run on physical devices and emulators, and they can take advantage of the Android framework APIs and supporting APIs, such as AndroidX Test.
- Instrumented tests provide more fidelity than local unit tests, but they run much more slowly.
- Therefore, it is recommend to use instrumented unit tests only in cases where you must test against the behavior of a real device.

https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests

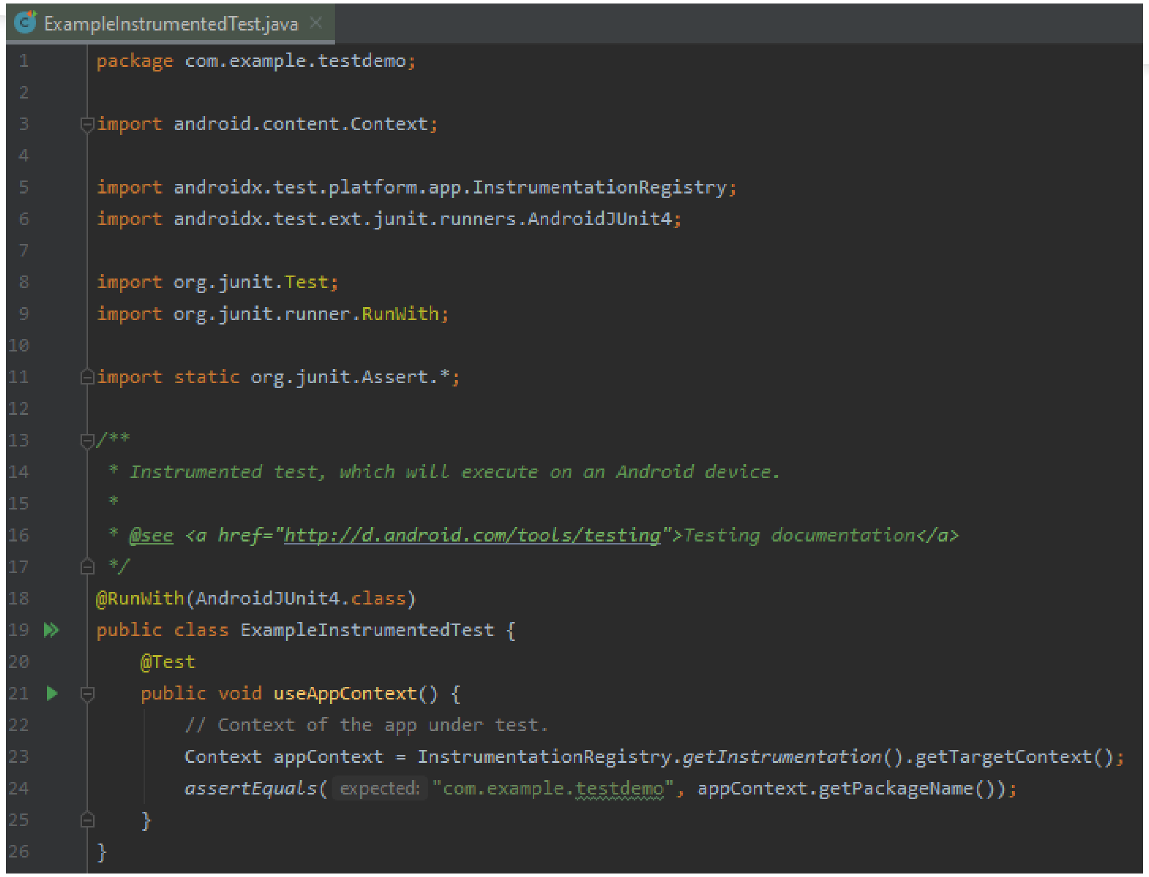
Testing Devices

When running your tests on a device, you can choose among the following types:

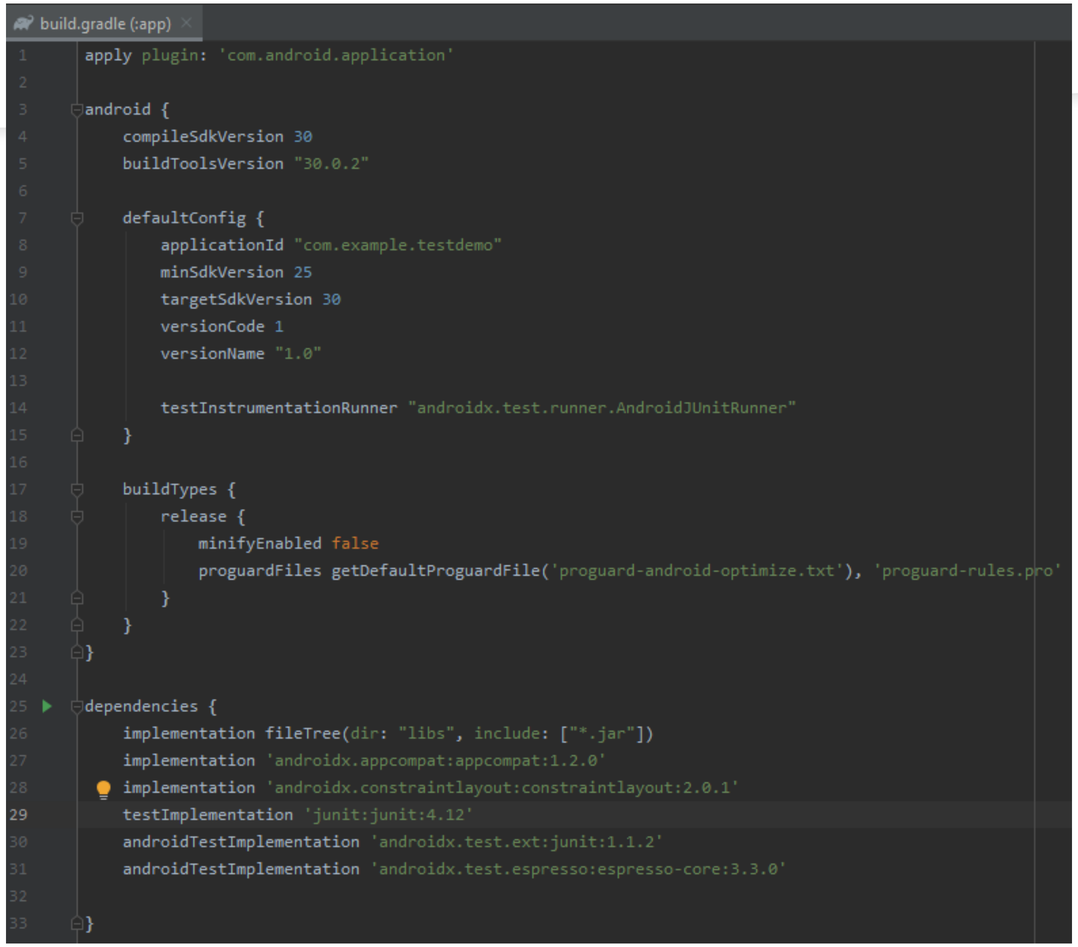
- **Real devices** offer the highest fidelity but also take the most time to run your tests.
- **Simulated devices (Robolectric)**, on the other hand, provide improved test speed at the cost of lower fidelity.
- **Virtual devices(ie Android Studio emulator)** offer a balance of fidelity and speed.

https://developer.android.com/training/testing/fundamentals http://robolectric.org/

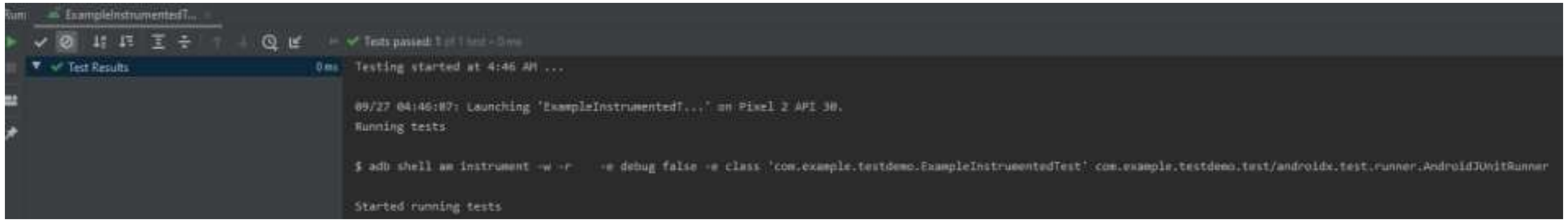
Instrumented Tests in Android Studio


ExampleInstrumentedTest.java

```java
package com.example.testdemo;

import android.content.Context;

import androidx.test.platform.app.InstrumentationRegistry;
import androidx.test.ext.junit.runners.AndroidJUnit4;

import org.junit.Test;
import org.junit.runner.RunWith;

import static org.junit.Assert.*;

/**
 * Instrumented test, which will execute on an Android device.
 *
 * @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
    @Test
    public void useAppContext() {
        // Context of the app under test.
        Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
        assertEquals( expected: "com.example.testdemo", appContext.getPackageName());
    }
}
```


build.gradle (:app)

```groovy
apply plugin: 'com.android.application'

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId "com.example.testdemo"
        minSdkVersion 25
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

}
```


Run: ExampleInstrumentedT...
Tests passed: 1 of 1 test - 0 ms
Test Results 0 ms

```
Testing started at 4:46 AM ...

09/27 04:46:07: Launching 'ExampleInstrumentedT...' on Pixel 2 API 30.
Running tests

$ adb shell am instrument -w -r    -e debug false -e class 'com.example.testdemo.ExampleInstrumentedTest' com.example.testdemo.test/androidx.test.runner.AndroidJUnitRunner

Started running tests
```

UI Testing & Espresso

Use Espresso to write concise, beautiful, and reliable Android UI tests.

https://developer.android.com/training/testing/ui-testing/espresso-testing

Let's test the UI to see if interacting with it will show a view that we expect

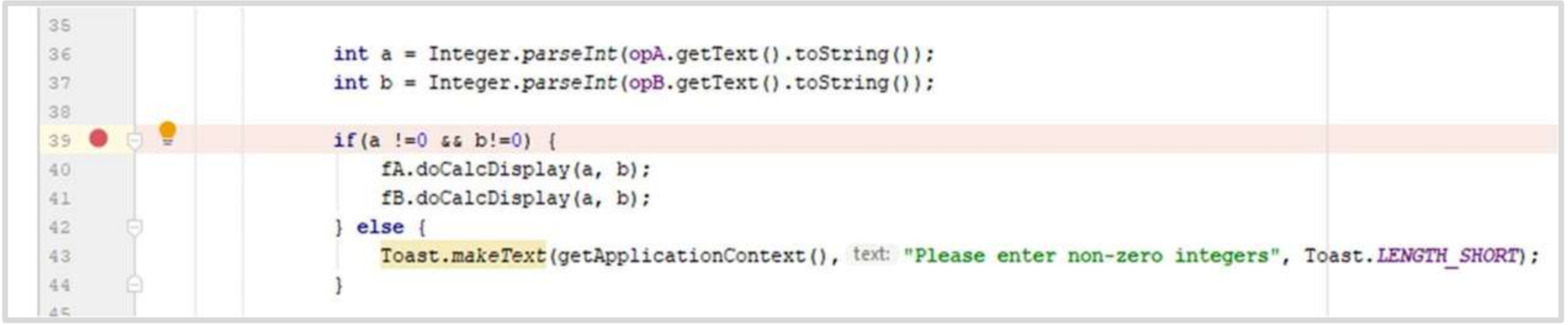
Testing Examples

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture10_TestingDemo

https://youtu.be/JRkDVvB106k https://developer.android.com/studio/test/espresso-test-recorder

Runtime Debugging

- Used to explore runtime behavior
- Interrogate values of variables
- Click in the margin of your code, a red ‘breakpoint’ appears



```
int a = Integer.parseInt(opA.getText().toString());
int b = Integer.parseInt(opB.getText().toString());

if(a !=0 && b!=0) {
    fA.doCalcDisplay(a, b);
    fB.doCalcDisplay(a, b);
} else {
    Toast.makeText(getApplicationContext(), text: "Please enter non-zero integers", Toast.LENGTH_SHORT);
}
```

https://developer.android.com/studio/debug

Runtime Debugging

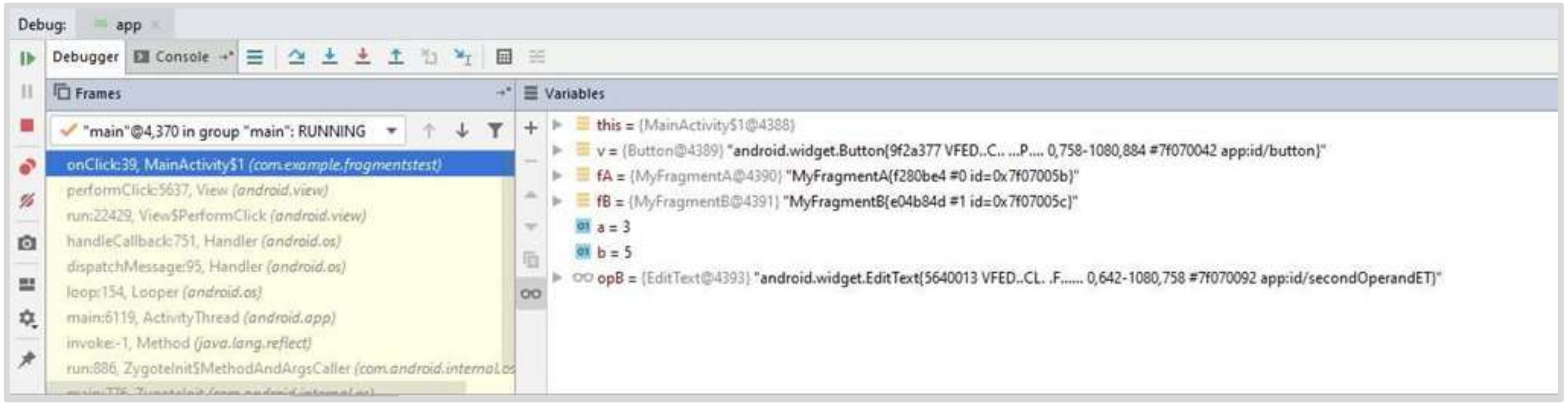
Click to start debugging

The code will run to this point

Make sure you choose a place ahead of any runtime errors you get on the console

Also choose points in your code that are informative to debug

It is used to assist your understanding of your code



```
"main"@4,370 in group "main": RUNNING
onClick:39, MainActivity$1 (com.example.fragmentstest)
performClick:5637, View (android.view)
run:22429, View$PerformClick (android.view)
handleCallback:751, Handler (android.os)
dispatchMessage:95, Handler (android.os)
loop:154, Looper (android.os)
main:6119, ActivityThread (android.app)
invoke:-1, Method (java.lang.reflect)
run:886, ZygoteInit$MethodAndArgsCaller (com.android.internal.os
```

```
this = {MainActivity$1@4388}
v = {Button@4389} "android.widget.Button{9f2a377 VFED..C.. ...P.... 0,758-1080,884 #7f070042 app:id/button}"
fA = {MyFragmentA@4390} "MyFragmentA{f280be4 #0 id=0x7f07005b}"
fB = {MyFragmentB@4391} "MyFragmentB{e04b84d #1 id=0x7f07005c}"
a = 3
b = 5
opB = {EditText@4393} "android.widget.EditText{5640013 VFED..CL. .F...... 0,642-1080,758 #7f070092 app:id/secondOperandET}"
```

https://developer.android.com/studio/debug

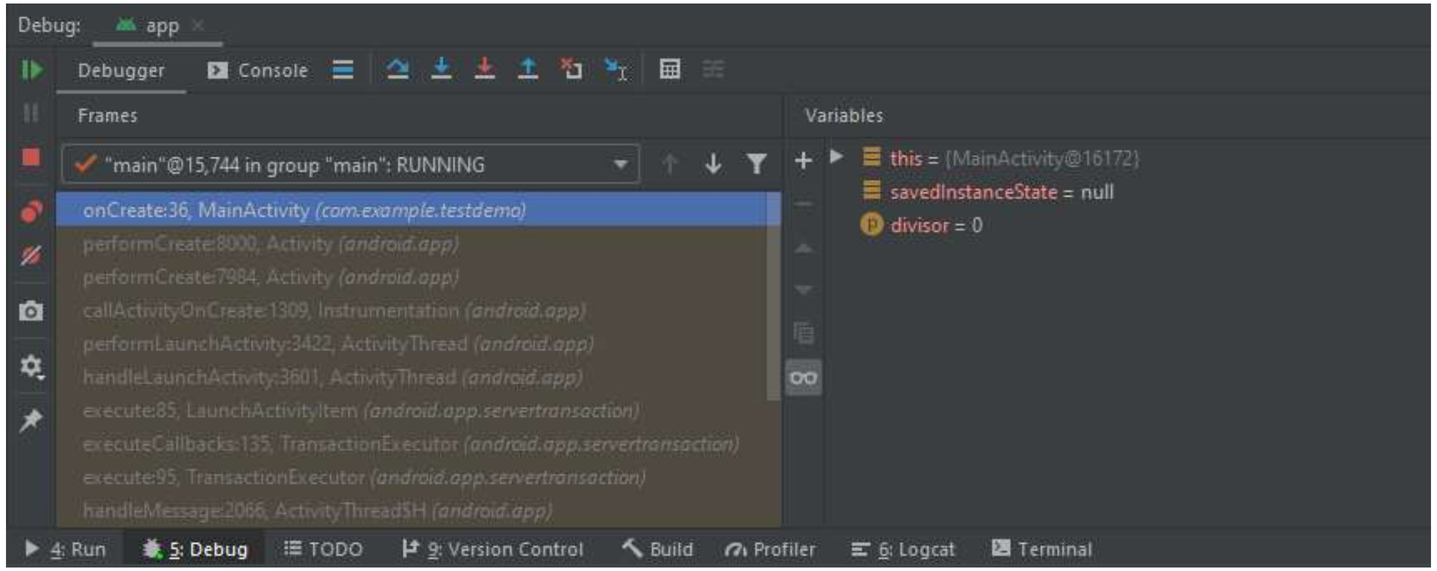
Stepping through code

Step Over, To advance to the next line in the code (without entering a method), click Step Over.

Step Into, To advance to the first line inside a method call

Step Out, To advance to the next line outside the current method

To **resume** normal execution of the app, select Run > Resume Program or click the Resume icon .



```
"main"@15,744 in group "main": RUNNING
onCreate:36, MainActivity (com.example.testdemo)
performCreate:8000, Activity (android.app)
performCreate:7984, Activity (android.app)
callActivityOnCreate:1309, Instrumentation (android.app)
performLaunchActivity:3422, ActivityThread (android.app)
handleLaunchActivity:3601, ActivityThread (android.app)
execute:85, LaunchActivityItem (android.app.servertransaction)
executeCallbacks:135, TransactionExecutor (android.app.servertransaction)
execute:95, TransactionExecutor (android.app.servertransaction)
handleMessage:2066, ActivityThread$H (android.app)
```

```
this = {MainActivity@16172}
savedInstanceState = null
divisor = 0
```

https://developer.android.com/studio/debug

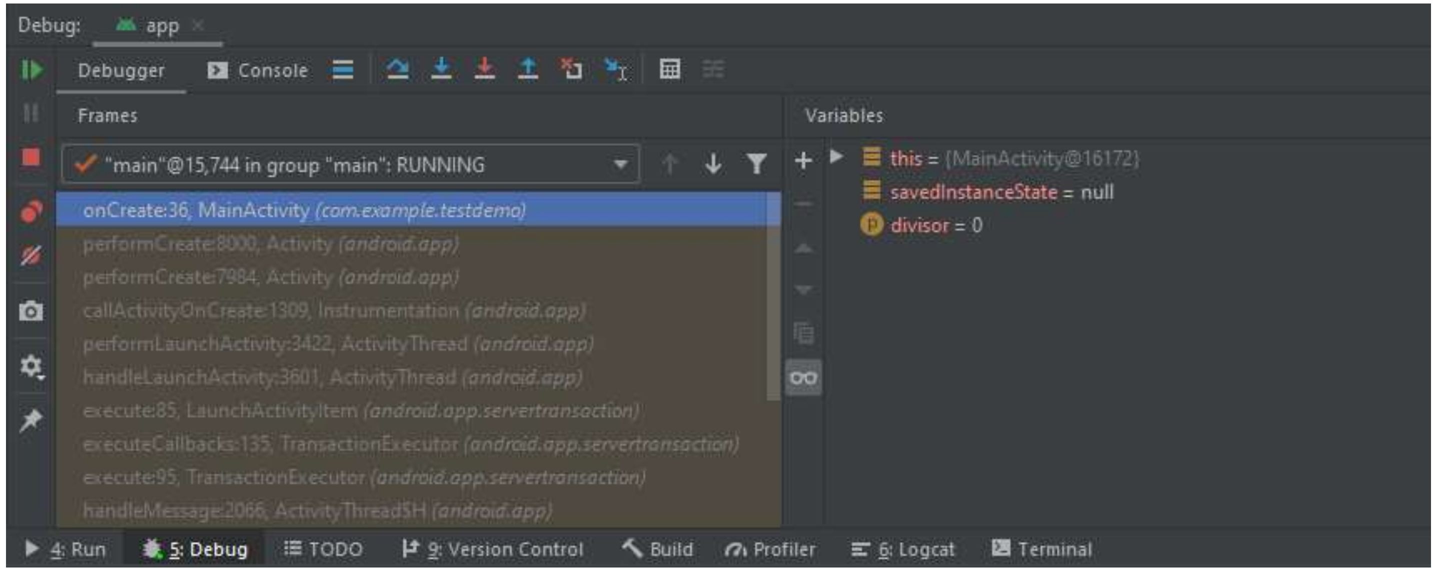
Stepping through code

Step Over, To advance to the next line in the code (without entering a method), click Step Over.

Step Into, To advance to the first line inside a method call

Step Out, To advance to the next line outside the current method

To **resume** normal execution of the app, select Run > Resume Program or click the Resume icon .



```
"main"@15,744 in group "main": RUNNING
onCreate:36, MainActivity (com.example.testdemo)
performCreate:8000, Activity (android.app)
performCreate:7984, Activity (android.app)
callActivityOnCreate:1309, Instrumentation (android.app)
performLaunchActivity:3422, ActivityThread (android.app)
handleLaunchActivity:3601, ActivityThread (android.app)
execute:85, LaunchActivityItem (android.app.servertransaction)
executeCallbacks:135, TransactionExecutor (android.app.servertransaction)
execute:95, TransactionExecutor (android.app.servertransaction)
handleMessage:2066, ActivityThread$H (android.app)
```

```
this = {MainActivity@16172}
savedInstanceState = null
divisor = 0
```

https://developer.android.com/studio/debug

Use the system log

```
import android.util.Log;
...
public class MyActivity extends Activity {
    private static final String TAG = MyActivity.class.getSimpleName();
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
       ...
       if (savedInstanceState != null) {
            Log.d(TAG, "onCreate() Restoring previous state");
            /* restore state */
        } else {
            Log.d(TAG, "onCreate() No saved state available");
            /* initialize app */
        }
    }
}
```

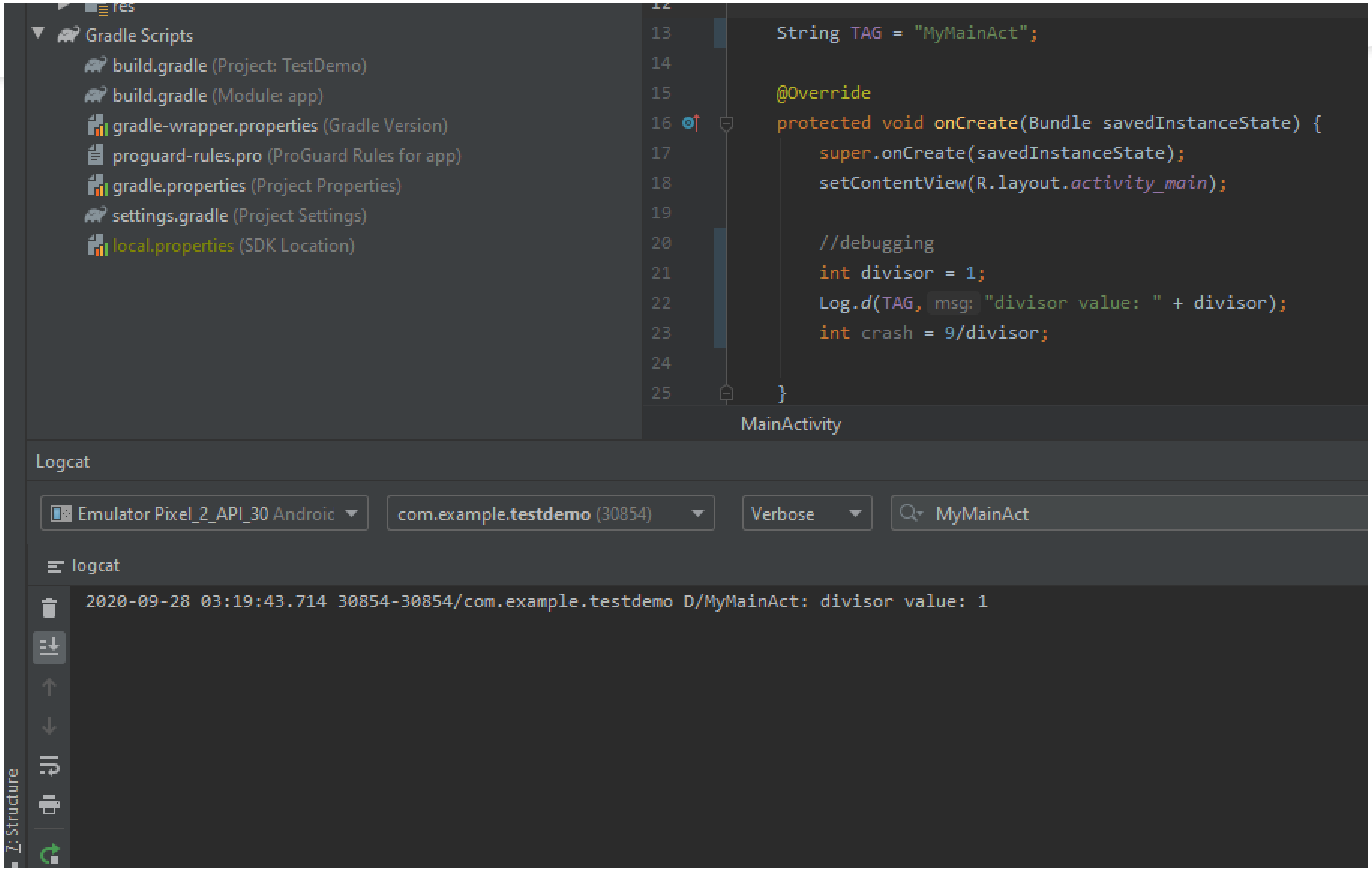
To write log messages in your code, use the Log class. Log messages help you understand the execution flow by collecting the system debug output while you interact with your app. Log messages can tell you what part of your application failed.

https://developer.android.com/studio/debug

View the system log

You can view and filter debug and other system messages in the Logcat window. For example, you can see messages when garbage collection occurs, or messages that you add to your app with the Log class.

To use logcat, start debugging and select the Logcat tab in the bottom toolbar



res
Gradle Scripts
build.gradle (Project: TestDemo)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
MainActivity
Logcat
logcat
7: Structure

```
    String TAG = "MyMainAct";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //debugging
        int divisor = 1;
        Log.d(TAG, msg: "divisor value: " + divisor);
        int crash = 9/divisor;

    }
```

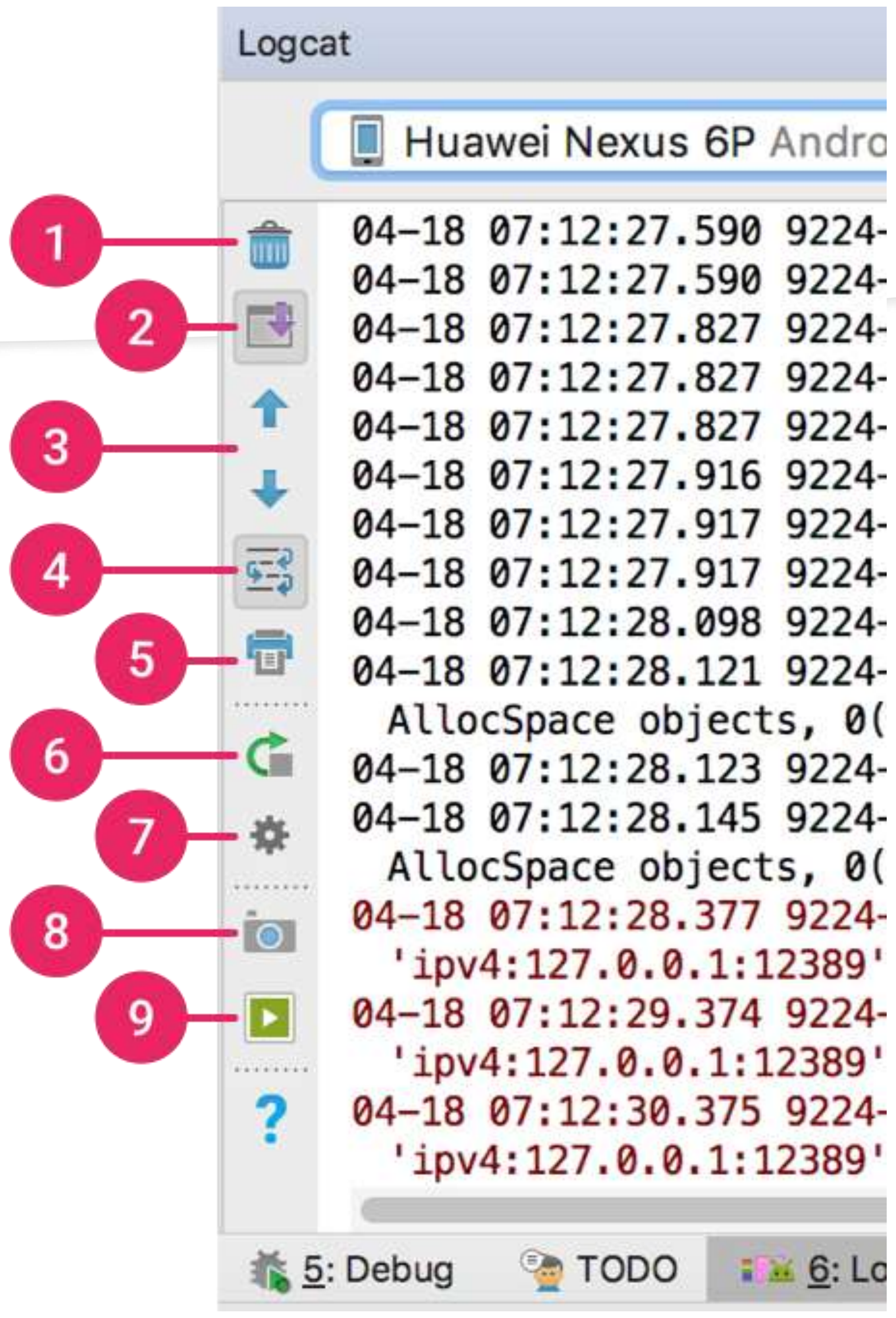
Emulator Pixel_2_API_30 Androic | com.example.testdemo (30854) | Verbose | MyMainAct

```
2020-09-28 03:19:43.714 30854-30854/com.example.testdemo D/MyMainAct: divisor value: 1
```

https://developer.android.com/studio/debug/am-logcat

More about logcat

1.Clear logcat : Click to clear the visible log.
2.Scroll to the end : Click to jump to the bottom of the log and see the latest log messages. If you then click a line in the log, the view pauses scrolling at that point.
3.Up the stack trace and **Down the stack trace** : Click to navigate up and down the stack traces in the log, selecting the subsequent filenames (and viewing the correspnding line numbers in the editor) that appear in the printed exceptions. This is the same behavior as when you click on a filename in the log.
4.Use soft wraps : Click to enable line wrapping and prevent horizontal scrolling (though any unbreakable strings will still require horizontal scrolling).
5.Print : Click to print the logcat messages. After selecting your print preferences in the dialog that appears, you can also choose to save to a PDF.
6.Restart : Click to clear the log and restart logcat. Unlike the **Clear logcat** button, this recovers and displays previous log messages, so is most useful if Logcat becomes unresponsive and you don't want to lose your log messages.
7.Logcat header : Click to open the **Configure Logcat Header** dialog, where you can customize the appearance of each logcat message, such as whether to show the date and time.
8.Screen capture : Click to capture a screenshot.
9.Screen record : Click to record a video of the device (for a maximum of 3 minutes).



```
04-18 07:12:27.590 9224-
04-18 07:12:27.590 9224-
04-18 07:12:27.827 9224-
04-18 07:12:27.827 9224-
04-18 07:12:27.827 9224-
04-18 07:12:27.916 9224-
04-18 07:12:27.917 9224-
04-18 07:12:27.917 9224-
04-18 07:12:28.098 9224-
04-18 07:12:28.121 9224-
  AllocSpace objects, 0(
04-18 07:12:28.123 9224-
04-18 07:12:28.145 9224-
  AllocSpace objects, 0(
04-18 07:12:28.377 9224-
  'ipv4:127.0.0.1:12389'
04-18 07:12:29.374 9224-
  'ipv4:127.0.0.1:12389'
04-18 07:12:30.375 9224-
  'ipv4:127.0.0.1:12389'
```

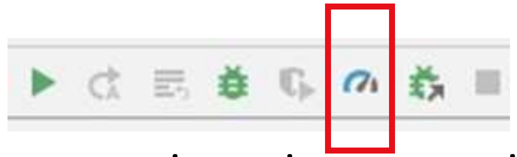
https://developer.android.com/studio/debug/am-logcat

https://youtu.be/rjlhSDhFwzM

Performance Profiling

Investigate the runtime performance of your application

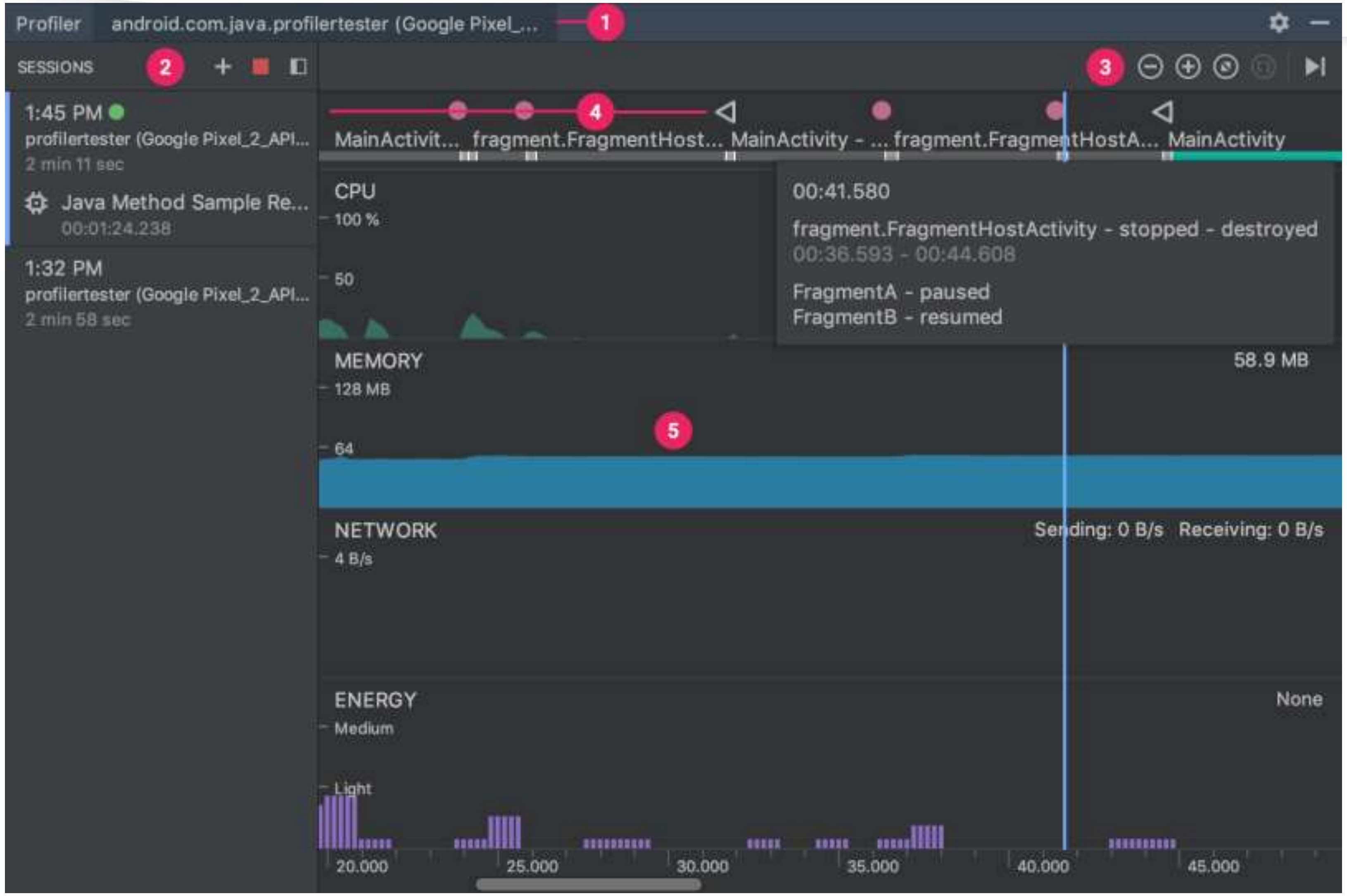
- CPU
- Memory
- Network
- Energy

Start the profiler by clicking the icon 

The app launches, now use all the features or investigate a portion of the app that is problematic.

https://developer.android.com/studio/profile/android-profiler

Performance Profiling


Profiler
android.com.java.profilertester (Google Pixel_...
1
SESSIONS
2
3
4
5
1:45 PM
profilertester (Google Pixel_2_API...
2 min 11 sec
Java Method Sample Re...
00:01:24.238
1:32 PM
profilertester (Google Pixel_2_API...
2 min 58 sec
MainActivit... fragment.FragmentHost... MainActivity - ... fragment.FragmentHostA... MainActivity
CPU
100 %
50
MEMORY
128 MB
64
58.9 MB
NETWORK
4 B/s
Sending: 0 B/s Receiving: 0 B/s
ENERGY
Medium
Light
None
20.000
25.000
30.000
35.000
40.000
45.000

00:41.580
fragment.FragmentHostActivity - stopped - destroyed
00:36.593 - 00:44.608
FragmentA - paused
FragmentB - resumed

1 Android Profiler shows the process and device currently being profiled.

2 In the Sessions pane, choose which session to view, or start a new profiling session.

3 Use the zoom buttons to control how much of the timeline to view, or use the Attach to live button to jump to the real-time updates.

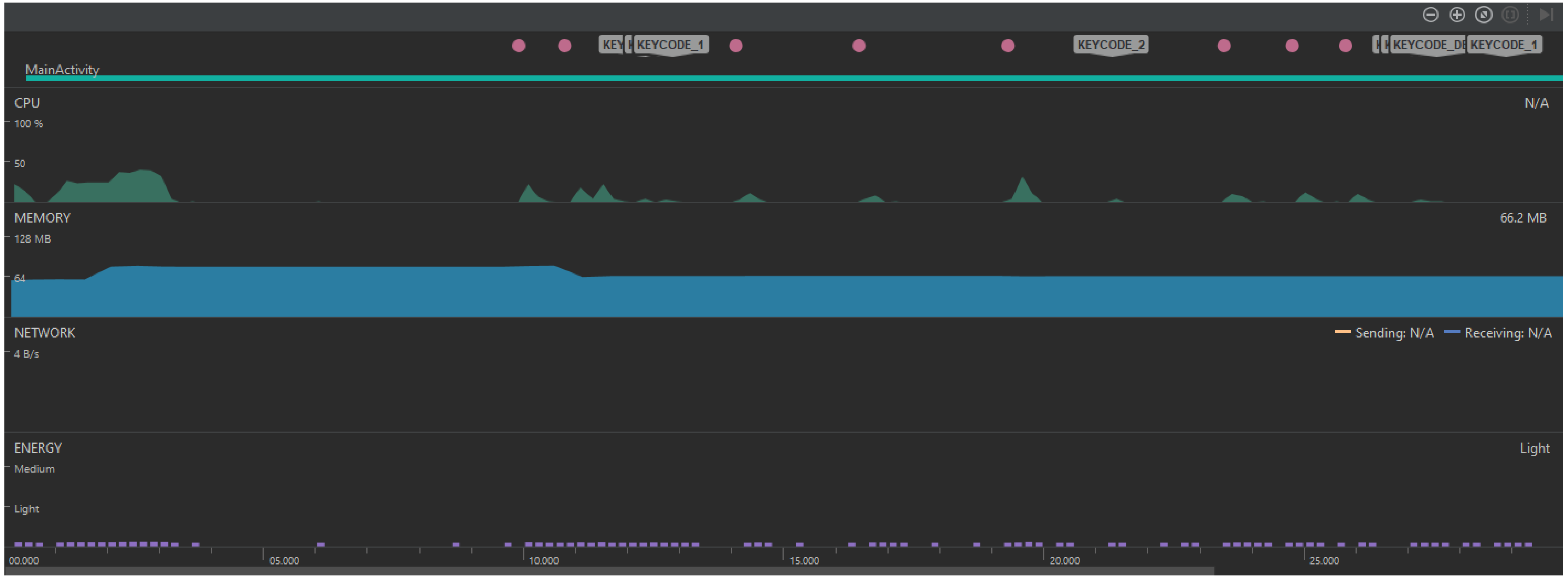
4 The event timeline shows events related to user input, including keyboard activity, volume control changes, and screen rotations.

5 The shared timeline view, which includes graphs for CPU, memory, network, and energy usage.

https://developer.android.com/studio/profile/android-profiler

Performance Profiling

Demo



https://developer.android.com/studio/profile/android-profiler

Privacy best practices

Permissions
Build trust with your users by being transparent and providing users control over how they experience your app.

Minimize your use of location
If your app requests permission to access location, help users make an informed decision.

Handle data safely
Be transparent and secure in how you handle sensitive data.

Use resettable identifiers
Respect your user's privacy and use resettable identifiers. (ie Don't access IMEI and device serial number, as these identifiers are persistent.)

https://developer.android.com/about/versions/11/privacy

https://developer.android.com/privacy/best-practices

Usability Testing

ISO/IEC 9126-4 Usability Metrics

Effectiveness: The accuracy and completeness with which users achieve specified goals

Efficiency: The resources expended in relation to the accuracy and completeness with which users achieve goals.

Satisfaction: The comfort and acceptability of use.

Quantify your usability testing

https://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability

Usability Testing

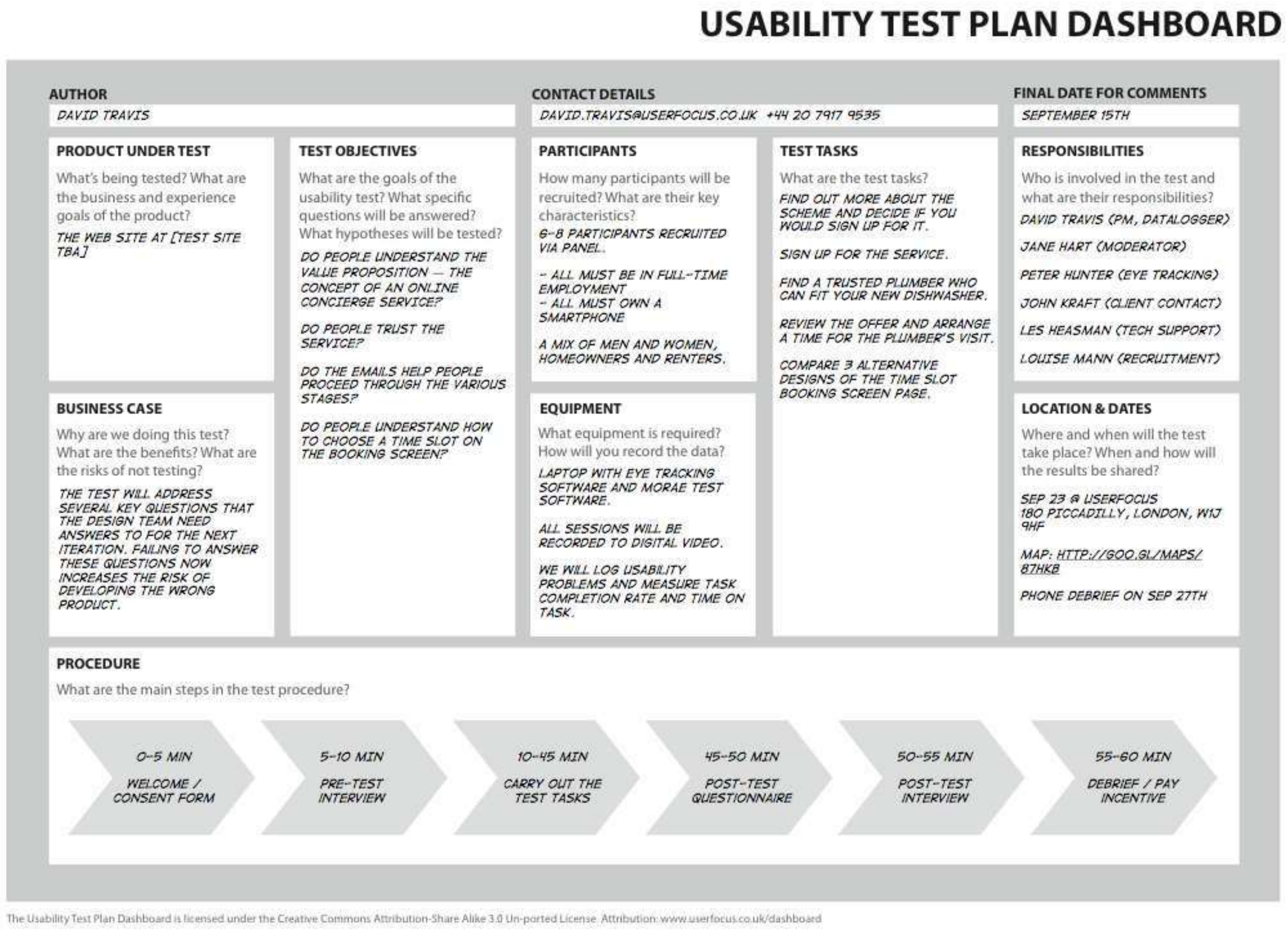
Fill this out for your assignment Phase 3

Design a number of scenarios (tasks) for your test subjects to act out

Post test questions…. Pair them.

x = process supporting a user story

What did you appreciate about x What did you not appreciate about



USABILITY TEST PLAN DASHBOARD

AUTHOR
DAVID TRAVIS

CONTACT DETAILS
DAVID.TRAVIS@USERFOCUS.CO.UK +44 20 7917 9535

FINAL DATE FOR COMMENTS
SEPTEMBER 15TH

PRODUCT UNDER TEST
What's being tested? What are the business and experience goals of the product?
THE WEB SITE AT [TEST SITE TBA]

BUSINESS CASE
Why are we doing this test? What are the benefits? What are the risks of not testing?
THE TEST WILL ADDRESS SEVERAL KEY QUESTIONS THAT THE DESIGN TEAM NEED ANSWERS TO FOR THE NEXT ITERATION. FAILING TO ANSWER THESE QUESTIONS NOW INCREASES THE RISK OF DEVELOPING THE WRONG PRODUCT.

TEST OBJECTIVES
What are the goals of the usability test? What specific questions will be answered? What hypotheses will be tested?
DO PEOPLE UNDERSTAND THE VALUE PROPOSITION — THE CONCEPT OF AN ONLINE CONCIERGE SERVICE?
DO PEOPLE TRUST THE SERVICE?
DO THE EMAILS HELP PEOPLE PROCEED THROUGH THE VARIOUS STAGES?
DO PEOPLE UNDERSTAND HOW TO CHOOSE A TIME SLOT ON THE BOOKING SCREEN?

PARTICIPANTS
How many participants will be recruited? What are their key characteristics?
6-8 PARTICIPANTS RECRUITED VIA PANEL.
- ALL MUST BE IN FULL-TIME EMPLOYMENT
- ALL MUST OWN A SMARTPHONE
A MIX OF MEN AND WOMEN, HOMEOWNERS AND RENTERS.

EQUIPMENT
What equipment is required? How will you record the data?
LAPTOP WITH EYE TRACKING SOFTWARE AND MORAE TEST SOFTWARE.
ALL SESSIONS WILL BE RECORDED TO DIGITAL VIDEO.
WE WILL LOG USABILITY PROBLEMS AND MEASURE TASK COMPLETION RATE AND TIME ON TASK.

TEST TASKS
What are the test tasks?
FIND OUT MORE ABOUT THE SCHEME AND DECIDE IF YOU WOULD SIGN UP FOR IT.
SIGN UP FOR THE SERVICE.
FIND A TRUSTED PLUMBER WHO CAN FIT YOUR NEW DISHWASHER.
REVIEW THE OFFER AND ARRANGE A TIME FOR THE PLUMBER'S VISIT.
COMPARE 3 ALTERNATIVE DESIGNS OF THE TIME SLOT BOOKING SCREEN PAGE.

RESPONSIBILITIES
Who is involved in the test and what are their responsibilities?
DAVID TRAVIS (PM, DATALOGGER)
JANE HART (MODERATOR)
PETER HUNTER (EYE TRACKING)
JOHN KRAFT (CLIENT CONTACT)
LES HEASMAN (TECH SUPPORT)
LOUISE MANN (RECRUITMENT)

LOCATION & DATES
Where and when will the test take place? When and how will the results be shared?
SEP 23 @ USERFOCUS
180 PICCADILLY, LONDON, W1J 9HF
MAP: HTTP://GOO.GL/MAPS/87HKB
PHONE DEBRIEF ON SEP 27TH

PROCEDURE
What are the main steps in the test procedure?
0-5 MIN WELCOME / CONSENT FORM → 5-10 MIN PRE-TEST INTERVIEW → 10-45 MIN CARRY OUT THE TEST TASKS → 45-50 MIN POST-TEST QUESTIONNAIRE → 50-55 MIN POST-TEST INTERVIEW → 55-60 MIN DEBRIEF / PAY INCENTIVE

https://www.userfocus.co.uk/pdf/usabilitydashboard.pdf
https://www.userfocus.co.uk/articles/usability_test_plan_dashboard.html
https://usabilitygeek.com/usability-testing-mobile-applications/
https://www.playbookux.com/usability-testing-questions/

CSE2MAD

Live Lecture

Lecture 10

Fragment Communication

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture10_FragmentExample

Firebase UI

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture10_FirebaseAuthUIDemo

https://github.com/firebase/FirebaseUI-Android