19CSI605 - Mobile Application Development

Testing

G. Pradeep, AP

Sri Krishna College of Engineering and Technology, Coimbatore-641008

Testing

About testing

- Testing an android application manually is a tedious work where every time you make a change and execute.
- Writing and running tests is a critical part of the software development process.
- **Test-driven development (TDD)** is a popular software development philosophy that places tests at the core of all software development for an app or service.
- The code you write to test your app doesn't end up in the production version of your app; it lives only on your development machine, alongside your app's code in Android Studio.

Testing

Types of tests

- Android supports several different kinds of tests and testing frameworks.
- Two basic forms of testing Android Studio supports are **local unit tests** and **instrumented tests**.

Local unit tests are tests that are compiled and run entirely on your local machine with the Java Virtual Machine (JVM). {Test the parts of the app}

Instrumented tests are tests that run on an Android-powered device or emulator. These tests have access to the Android framework and to Instrumentation information such as the app's Context. Ex: UI testing

Testing

Unit Testing

- ✓ Unit tests should be the fundamental tests in your app testing strategy.
- ✓ By creating and running unit tests against your code, you can verify that the logic of individual functional code areas or units is correct.
- ✓ Running unit tests after every build helps you catch and fix problems introduced by code changes to your app.
- ✓ A unit test generally exercises the functionality of the smallest possible unit of code (which could be a method, class, or component) in a repeatable way

Example : Mockito, JUnit

Testing

The Android Testing Support Library

- ✓ The Android Testing Support Library provides the infrastructure and APIs for testing Android apps, including support for JUnit 4.
- ✓ With the testing support library you can build and run test code for your apps.

To check for the Android Testing Support Repository, follow these steps:

1. In Android Studio choose **Tools > Android > SDK Manager**.
2. Click the **SDK Tools** tab, and look for the Support Repository.
3. If necessary, update or install the library.

Testing

<u>Setting up testing</u>

To prepare your project for testing in Android Studio, you need to:

- Organize your tests in a ***source set***.
- Configure your project's Gradle dependencies to include testing-related APIs.

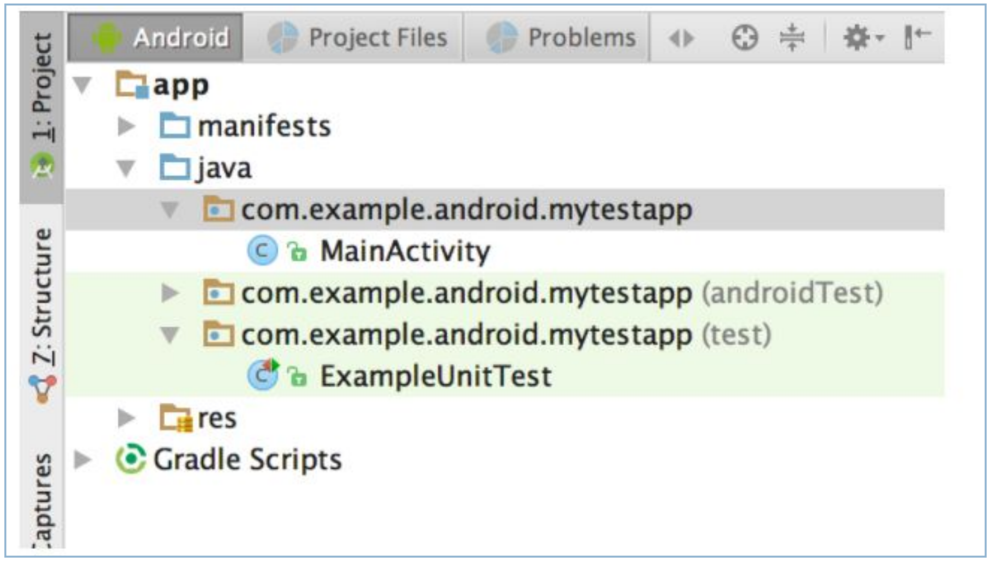
Testing

Android Studio source sets

- *Source sets* are collections of code in your project that are for different build targets or other "flavors" of your app.
- When Android Studio creates your project, it creates three source sets for you:
- The *main* source set, for your app's code and resources.
- The (test) source set, for your app's local unit tests. The source set shows (test) after the package name.
- The (androidTest) source set, for Android instrumented tests. The source set shows (androidTest) after the package name.

Android Studio source sets

- **Source sets** appear in the Android Studio **Project > Android** pane under the package name for your app.
- The main source set includes just the package name.
- The test and androidTest source sets have the package name followed by (test) or (androidTest), respectively.



Testing

Configuring Gradle for test dependencies

- To use the unit testing APIs, you may need to configure the dependencies for your project.
- The default Gradle build file, provided by Activity templates such as the Empty Activity template, includes some of these dependencies, but you may need to add more dependencies for additional testing features such as matching or mocking frameworks.
- In your app project's build.gradle (Module: app) file, the following dependency should already be included (if not, you should add it):

```
testImplementation 'junit:junit:4.12'
```

Testing

Configuring a test runner

- A test runner is a library or set of tools that enables testing to occur and the results to be printed to a log.
- Your Android project has access to a basic JUnit test runner as part of the JUnit4 APIs.
- The Android test support library includes a test runner for instrumented and Espresso tests, AndroidJUnitRunner, which also supports JUnit 3 and 4.
- The following test runner should already be included in the defaultConfig section (if not, you should add it):

```
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
```

Testing

<u>Creating and running unit tests</u>

- Create your unit tests as a generic Java file using the JUnit 4 APIs, and store those tests in the (test) source set.
- Each Android Studio project template includes this source set and a sample Java test file called ExampleUnitTest.

Creating a new test class

- To create a new test class file, add a Java file to the (test) source set for your project.
- Test class files for unit testing are typically named for the class in your app that you are testing, with "Test" appended.

Testing

Creating a new test class

- For example, if you have a class called Calculator in your app, the class for your unit tests would be CalculatorTest.

To add a new test class file, follow these steps:

1. Expand the **java** folder and the folder for your app's test source set. The existing unit test class files are shown.
2. **Right-click** (or **Control-click**) on the test source set folder and select **New > Java Class.**
3. Name the file and click **OK**.

Testing

Writing your tests

- Use JUnit 4 syntax and annotations to write your tests.

For example, the test class shown below includes the following annotations:

- ✓ The @RunWith annotation indicates the test runner that should be used for the tests in this class.
- ✓ The @SmallTest annotation indicates that this is a small (and fast) test.
- ✓ The @Before annotation marks a method as being the setup for the test.
- ✓ The @Test annotation marks a method as an actual test.

Testing

Writing your tests using Annotation

```
@RunWith(JUnit4.class)
@SmallTest
public class CalculatorTest {
    private Calculator mCalculator;
    // Set up the environment for testing
    @Before
    public void setUp() {
        mCalculator = new Calculator();
    }
```

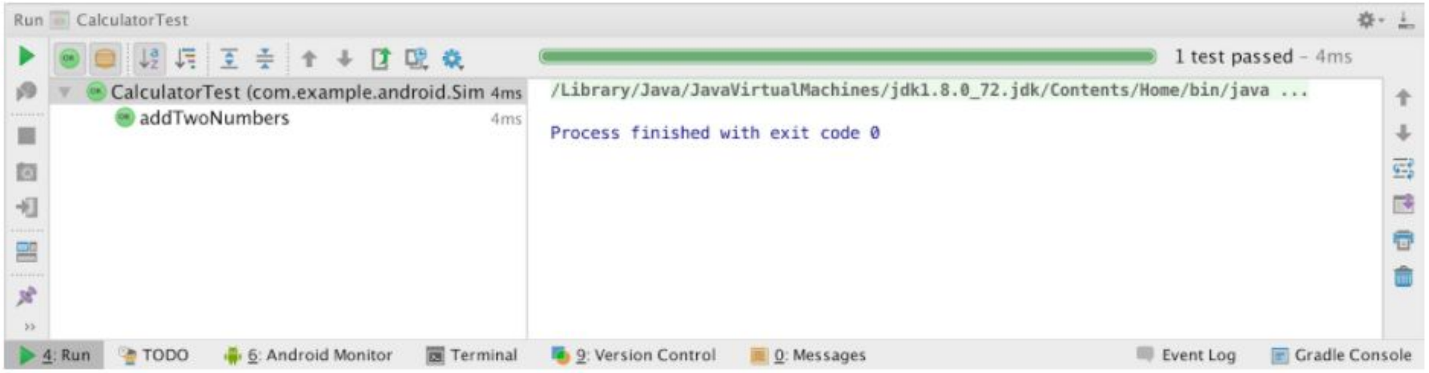
```
    // test for simple addition
    @Test
    public void addTwoNumbers() {
        double resultAdd = mCalculator.add(1d, 1d);
        assertThat(resultAdd, is(equalTo(2d)));
    }
}
```

Testing

Running your tests

To run your local unit tests, follow these steps:

- To run a single test, **right-click** (or **Control-click**) that test method and select **Run**.
- To test all the methods in a test class, **right-click** (or **Control-click**) the test file in the **Android > Project** pane, and select **Run**.
- To run all tests in a directory, **right-click** (or **Control-click**) on the directory and select **Run tests**



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_72.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

THANK YOU