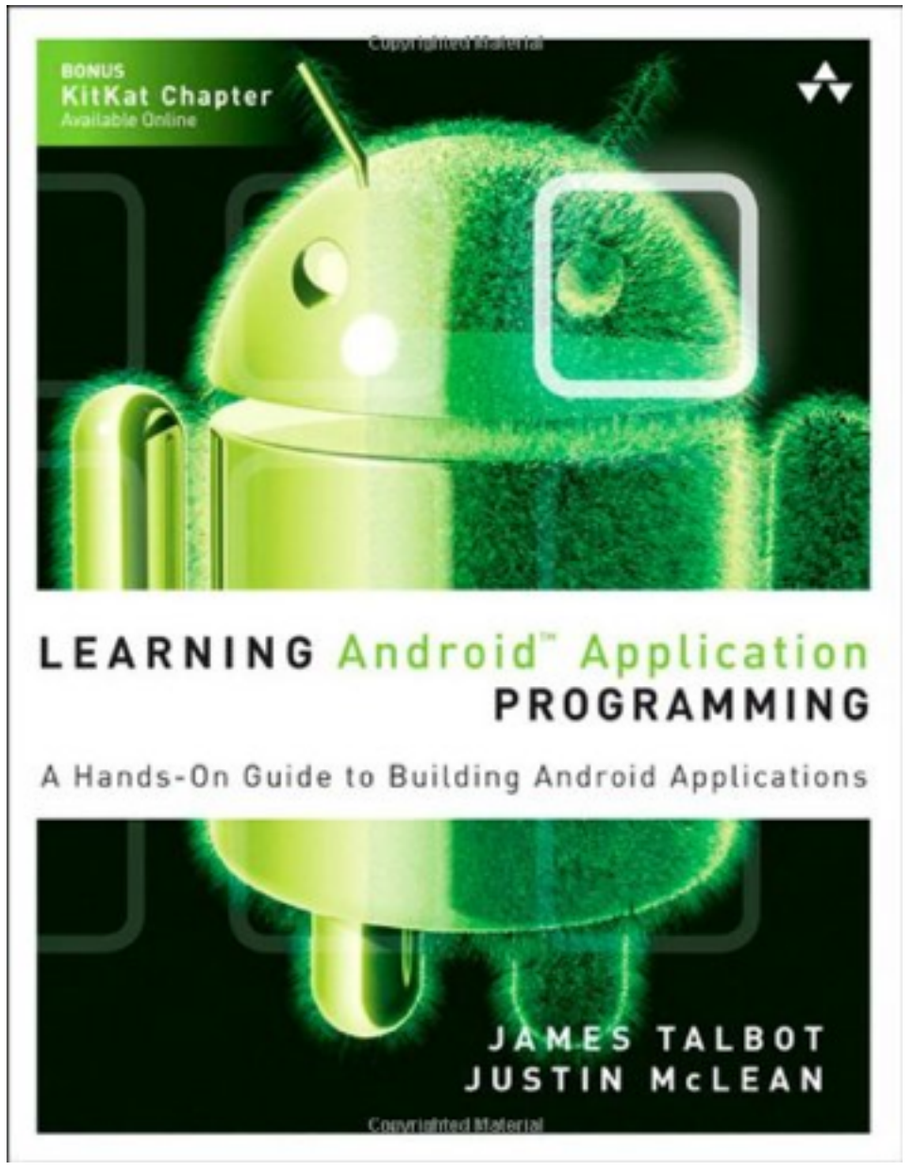
Testing Android Applications the Right Way - Part II

Justin Mclean
Email: justin@classsoftware.com
Twitter: @justinmclean

Who am I?

- Freelance developer for 20 years
- Written a book on Android Development
- Involved in the Apache Flex project
- Release manager for Apache Flex and Flex Unit
- Run the IoT Sydney meetup



Learning Android Application Programming

James Talbot and Justin Mclean

Book Discount

- 35% off from InformIT.com
- Use discount code MCLEAN2931
- Also available form Amazon and O'Reilly Safari Books online

Part I

- Looked at a few different types of testing
- Emulator, Device, The Monkey and JUnit 4 testing
- Focus on being pragmatic and using each method for what it's good at
- Sides for Part I and Part II are available
- https://github.com/justinmclean/AnDevCon2014/

JUnit Extensions

- Uses JUnit 3 not JUnit 4
- Test lifecycle of applications, services and activities
- Provide mock objects for Application, Context, PackageManager + others
- Enables UI testing
- Runs on emulator or device

JUnit 3 vs JUnit 4

- Doesn’t use annotations
- setUp method called before each test
- tearDown method called after each test
- All test methods names start with “test”
- Assets are static methods but Android adds more

Testing Activities

- Create an Android test project
- Test cases extend ActivityUnitTestCase
- Method to start activities (startActivity)
- Can test life cycle - common place for errors

setUp and tearDown

```
@Override
public void setUp() throws Exception {
    super.setUp();

    Intent intent = new Intent(
            getInstrumentation().getTargetContext(),
            TimerActivity.class);
    startActivity(intent, null, null);
    activity = getActivity();
}

@Override
public void tearDown() throws Exception {
    super.tearDown();

    activity = null;
}
```

Text UI Views Exist

```
public void testInitialUI() {
    TextView counter = (TextView)
        activity.findViewById(R.id.timer);
    Button start = (Button)
        activity.findViewById(R.id.start_button);
    Button stop = (Button)
        activity.findViewById(R.id.stop_button);
    Button takePhoto = (Button)
        activity.findViewById(R.id.photo_button);

    assertNotNull(counter);
    assertNotNull(start);
    assertNotNull(stop);
    assertNotNull(takePhoto);
}
```

Mock Application

- Application vs MockApplication
- Create interface from your existing application
- Create class than extends MockApplicaton and implements interface
- Change setup to set application to be mock class
- May need to add a few null checks in production

Stubbing / Mocking

- Can speed up tests
- Can exactly specify state of application
- Can easily test errors
- Can be expensive to maintain - double the amount of code changes

Using MockApplication

```
@Override
public void setUp() throws Exception {
    super.setUp();
    setApplication(new MockOnYourBike());

    Intent intent = new
Intent(getInstrumentation().getTargetContext(),
                TimerActivity.class);
    startActivity(intent, null, null);
    activity = getActivity();
}
```

Fill in Stubs

- Change mock methods to do minimum require for basic tests

- Create multiple mock classes to test for failure conditions or other test scenarios

- Stubs are usually very simple and return canned responses

Mock Stubs

```
@Override
public void stopTimer() {
    running = false;
}

@Override
public boolean isTimerRunning() {
    return running;
}

@Override
public String timerDisplay() {
    return "0:00:00";
}
```

Activity OnStart Test

```
public void testOnStart() {
    getInstrumentation()
        .callActivityOnStart(activity);

    assertFalse(activity.isCameraNull());
    assertTrue(activity.isHandlerNull());
    assertTrue(activity.isUpdateTimerNull());
}
```

Activity OnStop Test

```
public void testOnStop() {
        getInstrumentation()
                .callActivityOnStop(activity);

        assertTrue(activity.isCameraNull());
        assertTrue(activity.isHandlerNull());
        assertTrue(activity.isUpdateTimerNull());
    }
```

Testable Code

- Sometime you need to change production code to make it easily testable
- Adding helper methods to return internal state
- May be useful in existing code

Helper methods

```
// Methods used for testing
 public boolean isCameraNull() {
      return camera == null;
 }

public boolean isHandlerNull() {
     return handler == null;
}

public boolean isUpdateTimerNull() {
     return updateTimer == null;
}
```

Test all Activity States

```
public void testOnDestroy() {
    getInstrumentation()
        .callActivityOnDestroy(activity);
    // Nothing to test just check on RTE or the like
}

public void testOnRestart() {
    getInstrumentation()
        .callActivityOnRestart(activity);

    // Nothing to test just check on RTE or the like
}
```

UI Testing

- JUnit can’t do UI testing right? Yes it can!
- You can push buttons, interact with the UI and check that items exist on the screen
- Wont pick up pixel difference or the like or other UI visual issues

Testing Buttons

```
public void testStopStartButtons() {
    Button start = (Button) activity.findViewById(R.id.start_button);
    Button stop = (Button) activity.findViewById(R.id.stop_button);
    IOnYourBike application = ((IOnYourBike) activity.getApplication());

    getInstrumentation().callActivityOnStart(activity);
    assertFalse(application.isTimerRunning());

    TouchUtils.clickView(this, start);
    assertTrue(application.isTimerRunning());
    assertFalse("Start button is not enabled", start.isEnabled());
    assertTrue("Stop button is enabled", stop.isEnabled());

    TouchUtils.clickView(this, stop);
    assertFalse(application.isTimerRunning());
    assertTrue("Start button is enabled", start.isEnabled());
    assertFalse("Stop button is not enabled", stop.isEnabled());
}
```

Testing Services

- Test cases extend ServicesTestCase
- Method to start service (startService) and stop service (shutdownService)
- Make it easy to test service issue

Is a Service Running?

```
@Override
public void onCreate() {
    running = true;
    timer.reset();
}

@Override
public void onDestroy() {
    running = false;
}
```

Service Test

```
public class TimerServiceTests extends
ServiceTestCase<TimerService> {
        public TimerServiceTests() {
            super(TimerService.class);
        }
}
```

Service startUp and tearDown

```
@Override
protected void setUp() throws Exception {
    super.setUp();
    setApplication(new MockOnYourBike());
}

@Override
protected void tearDown() throws Exception {
    super.tearDown();
}
```

Testing Service Startup

```
public void testStart() {
    Intent intent = new Intent(getContext(),
            TimerService.class);

    assertNull(getService());

    startService(intent);
    assertTrue(getService().isRunning());

    startService(intent); // no effect
    assertTrue(getService().isRunning());

    assertNotNull(getService().getTimer());
}
```

Testing Service Shutdown

```
public void testShutdown() {
    Intent intent = new Intent(getContext(),
        TimerService.class);

    assertNull(getService());
    startService(intent);
    assertTrue(getService().isRunning());

    shutdownService();
    assertFalse(getService().isRunning());

}
```

Re-running Tests

- Set of test that run quickly
- Still need to manually run - must be a better way
- Run via command line:
  adb shell am instrument -w <test package/
  android.test.InstrumentationTestRunner>
- Needs device or emulator running (obviously)

Command Line

- Emulator can also be started via command line:
  emulator -avd <virtual device name> -no-snapshot-load

Ant scripts

- Generate build.xml file via:
  android update project -p .

- Test build via:
  ant clean debug

- Generate tests build.xml file via:
  android update test-project -m <project name> -p .

- Install and run via:
  ant debug install test

Continuous Integration

- Works off version control and runs all tests automatically on check in
- Little difficult to set up but well worth it
- Testing is automated and mostly painless
- Know exactly what code changes break something

Jenkins

- Jenkins is one of the more popular open source CI systems (http://jenkins-ci.org)
- Simple to install
- Simple easy to use web interface

Jenkins + Android

- Create new job that:
  - Checks code out of version control
  - Runs test every 15 minutes or when version control changes
  - Invoke ant debug install test

Cloud Testing

- Want to test on lot of real devices but not buy them? Try cloud testing
- Apkudo - it’s free
- Samsung Remote Test Lab - free for small blocks of time
- Lots of others: SkyForge, TestDroid, AppThwack, DroidCloud

Questions?

- Email me justin@classsoftware.com
- Twitter @justinmclean

Book Discount

- 35% off from InformIT.com
- Use discount code MCLEAN2931
- Also available form Amazon and O'Reilly Safari Books online