Robolectric

JUnit testing on Android

Disclaimer

It will **probably** not compile...

Testing on Android

Three options:

— Don't bother

— Instrumentation tests (run on actual device)

— JUnit tests (run on your local JVM)

Why Robolectric?

— We want JVM testing because it is fast and predictable

— But it is so hard!! "java.lang.RuntimeException: Stub!"

Example

```
@RunWith(RobolectricTestRunner.class)
class MyActivityTest {

    @Before
    public void doSetup() {
        // setup before every test-case
    }

    @Test
    public void testSayHello() {
        // your actual test
    }

    @After
    public void cleanup() {
        // run after every test case
    }
}
```

```
@Test
public void testSayHello() {
    MainActivity activity = Robolectric.buildActivity(MainActivity.class)
        .attach()
        .create()
        .visible()
        .start()
        .resume()
        .get();

    TextView textView = (TextView) activity.findViewById(R.id.hello_textview);
    assertThat(textView).containsText("Hello world!");

    EditText editText = (EditText) activity.findViewById(R.id.input_field);
    editText.setText("Jeroen");

    Button sayHelloButton = (Button) activity.findViewById(R.id.say_button);
    Robolectric.clickOn(sayHelloButton);

    assertThat(textView).containsText("Hello Jeroen!");
}
```

Shadows

Some things are hard to get in Android API's (mainly because they are not public).

This is solved by using Shadow classes.

```
ShadowTextView shadowTextView = Robolectric.shadowOf(textView);
```

Current state (regarding Gradle)

Getting started

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.0.0'
        classpath 'com.github.jcandksolutions.gradle:android-unit-test:2.1.1'
    }
}

apply plugin: 'com.android.application'
android {
    // android stuff
}

apply plugin: 'android-unit-test'

dependencies {
    testCompile 'junit:junit:4.11'
    testCompile 'org.robolectric:robolectric:2.4'

    // rest of dependencies
}
```

Run tests with ./gradlew testDebug

Code Coverage

```
apply plugin: 'jacoco'

jacoco {
    toolVersion = "0.7.1.201405082137" // this version is old...
}

task jacocoTestReport(type:JacocoReport, dependsOn: "testDebug") {
    group = "Reporting"
    description = "Generate Jacoco coverage reports"

    classDirectories = fileTree(
            dir: '../app/build/intermediates/classes/debug',
            excludes: ['**/R.class',
                       '**/R$*.class',
                       '**/*$ViewInjector*.*',
                       '**/BuildConfig.*',
                       '**/Manifest*.*']
    )

    sourceDirectories = files('../app/src/main/java')
    executionData = files('../app/build/jacoco/testDebug.exec')

    reports {
        xml.enabled = true
        html.enabled = true
    }

}
```

Source: http://raptordigital.blogspot.com.au/2014/08/code-coverage-reports-using-robolectric.html

- Directories change based on variants and buildtypes
- You can add multiple source and class directories
- Groovy tip: FileCollection.plus() returns a **new** collection instance.

Android Studio integration

- Through a seperate plugin
  https://plugins.jetbrains.com/plugin/7488?pr=idea
- Some issues with compiling changes, be aware!

Tips

Bad application class!

```
class MyApplication extends Application {

    public void onCreate() {
        super.onCreate();
        doSomethingThatWontWork();
        doNetworkIO();
    }
}

@RunWith(RobolectricTestRunner.class)
class SomeTest {

    @Test
    public void testSomething() {}
}
```

```
class MyApplication extends Application {

    public void onCreate() {
        super.onCreate();
        doBadStuff();
    }

    protected void doBadStuff() {
        doSomethingThatWontWork();
        doNetworkIO();
    }

}

class TestMyApplication extends MyApplication {
    @Override
    protected void doBadStuff() {
        // don't do it!!
    }
}
```

Use dependency Injection

— It will allow you to easily swap implementations of certain components

— Take your pick of Dagger, RoboGuice or roll your own.

Mock your network calls

```
@Test
public void myTest() {
    Robolectric.addHttpReponseRule("https://my.superendpoint.com/", response);
    Robolectric.addPendingHttpResponse(response);
}
```

— Robolectric will by default *not* talk to the network (you can enable that though).

— Http interception only works with the DefaultHttpClient

Allow for single-threaded testing

- AsyncTasks are handled out of the box
- Avoid using the Thread class directly
- Inject appropriate ExecutorService instead

Don't talk to storage directly

— Create an interface for talking to SharedPreferences and SQLite databases, they persist between test cases!

Getting rid of statics

```
SharedPrefsUtil.saveAccount(context, account);

class IAccountStore {
    public void saveAccount(Account account);o
}

class SharedPrefsAccountStore implements IAccountStore {
    public void saveAccount(Account account) {
        SharedPrefsUtil.saveAccount(mContext, account);
    }
}
```

Questions?

Feel free to ask me any question via email:

jeroen@tietema.net