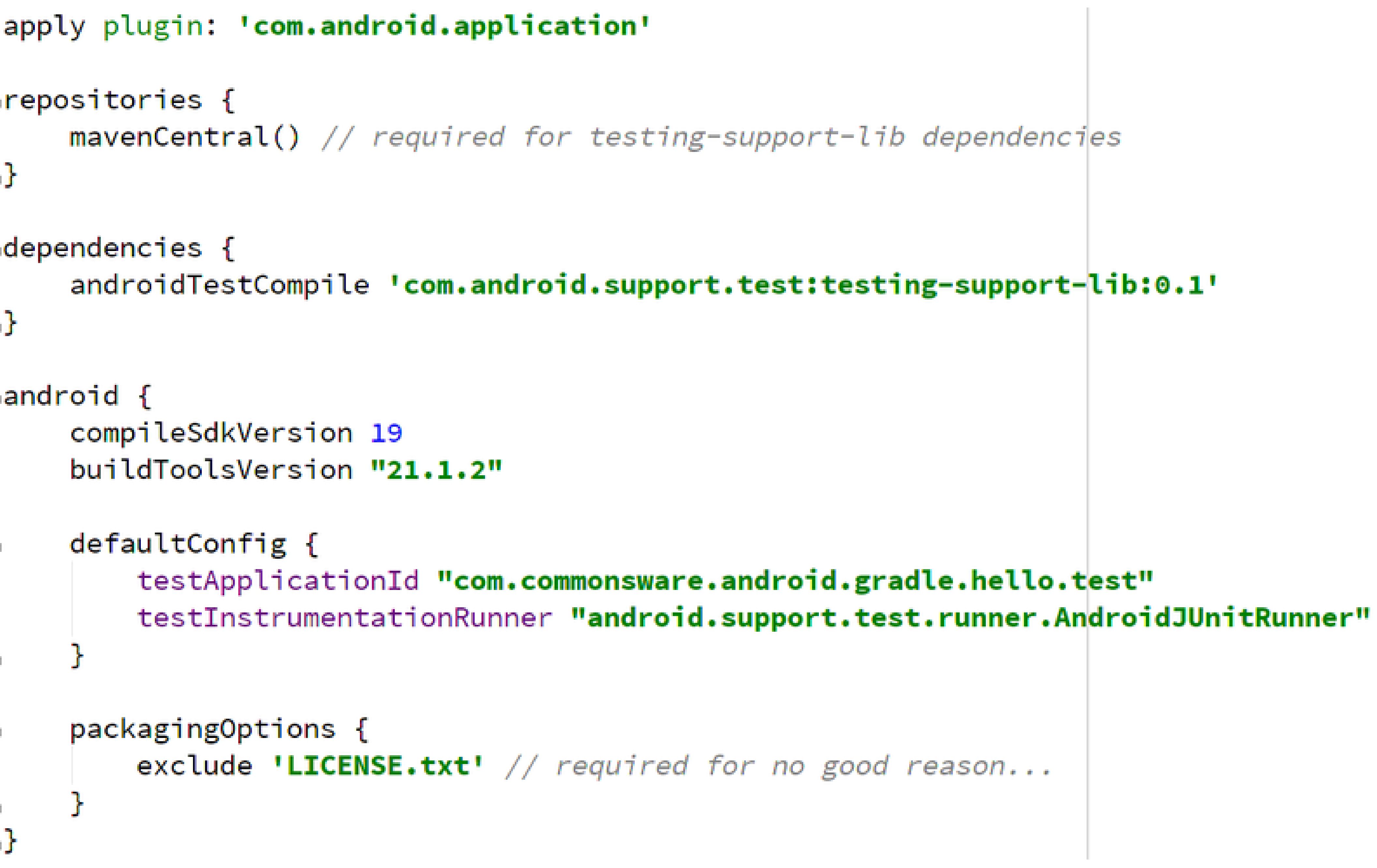
A Peek At JUnit4

JUnit3 and JUnit4

- JUnit3
  - Older than some Android developers
  - What comes stock with Android firmware
- JUnit4
  - Annotation-based tests
  - Now available via AndroidJUnitRunner
    - Android Studio only AFAICT
    - Requires test depedency

Step #1: Gradle Configuration

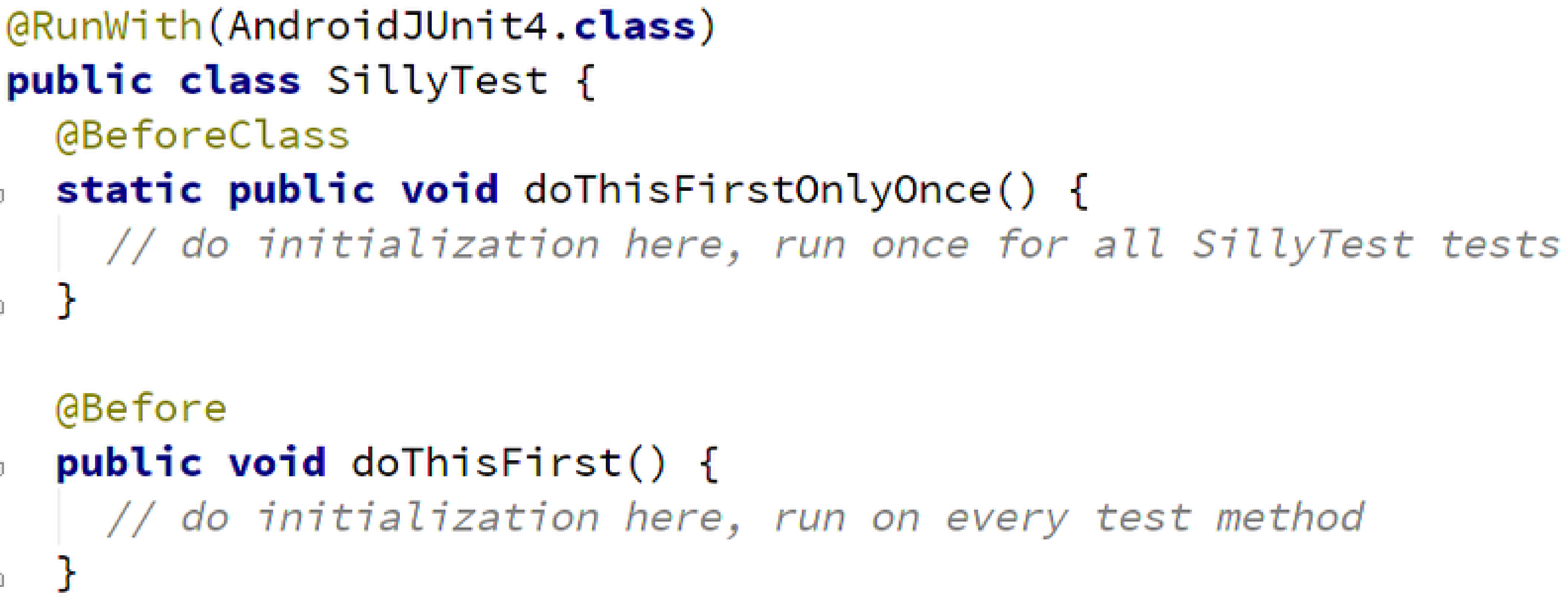
- `testApplicationId`
- `testInstrumentationRunner`
  - `android.test.InstrumentationTestRunner`
  - `android.support.test.runner.AndroidJUnitRunner`
- `androidTestCompile`
  - `com.android.support.test:testing-support-lib`
- Miscellaneous configuration bits for JUnit4



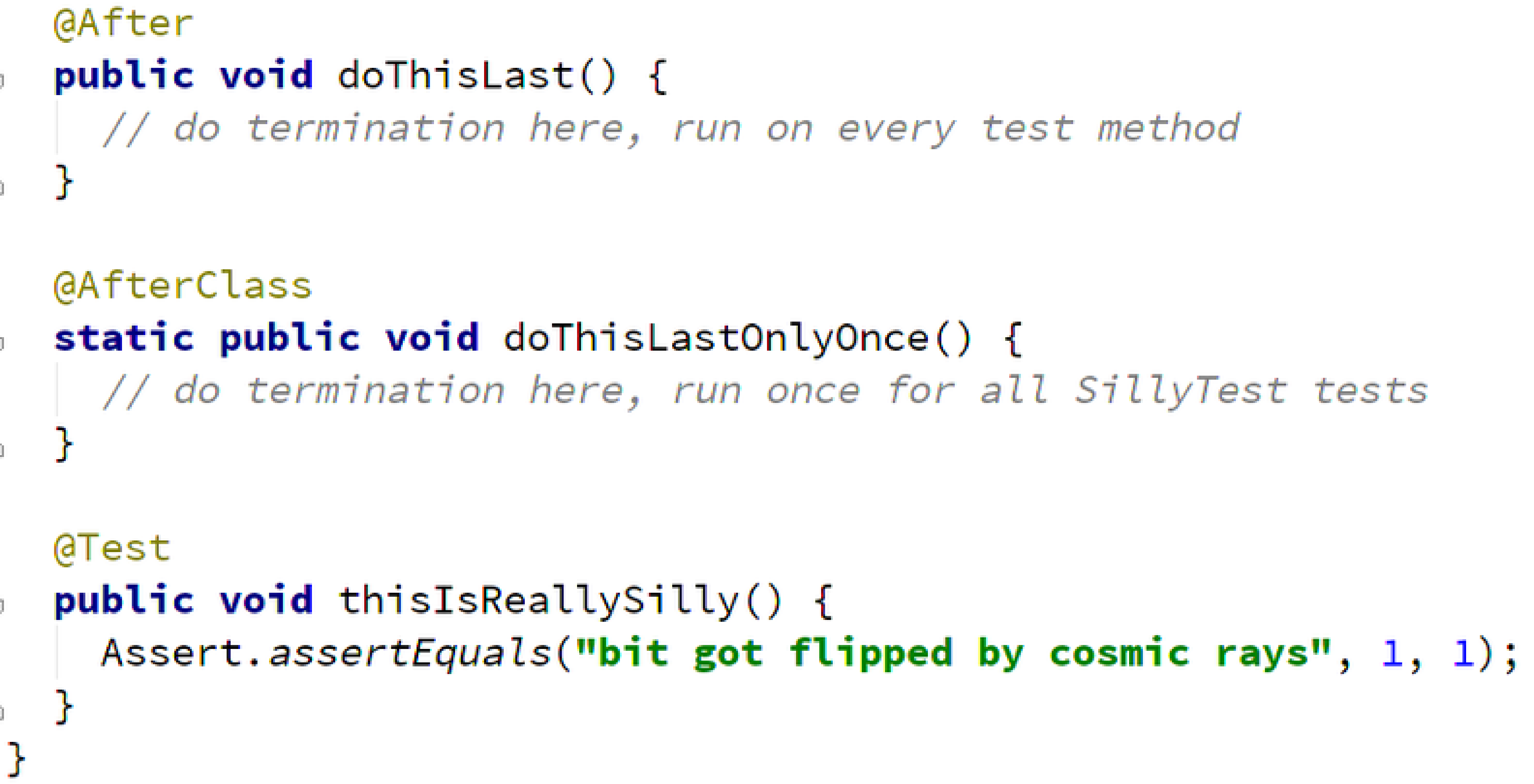
```
apply plugin: 'com.android.application'

repositories {
    mavenCentral() // required for testing-support-lib dependencies
}

dependencies {
    androidTestCompile 'com.android.support.test:testing-support-lib:0.1'
}

android {
    compileSdkVersion 19
    buildToolsVersion "21.1.2"

    defaultConfig {
        testApplicationId "com.commonsware.android.gradle.hello.test"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }

    packagingOptions {
        exclude 'LICENSE.txt' // required for no good reason...
    }
}
```

Step #2: Write Test Cases

- Code goes in `androidTest` sourceset
- Test Code Options
  - White-box: source goes in same package as what is being tested
  - Black-box: source goes in a separate package
- Test pure utility classes
  - JUnit3: ordinary `TestCase`
  - JUnit4: any class with `@RunWith(AndroidJUnit4.class)`



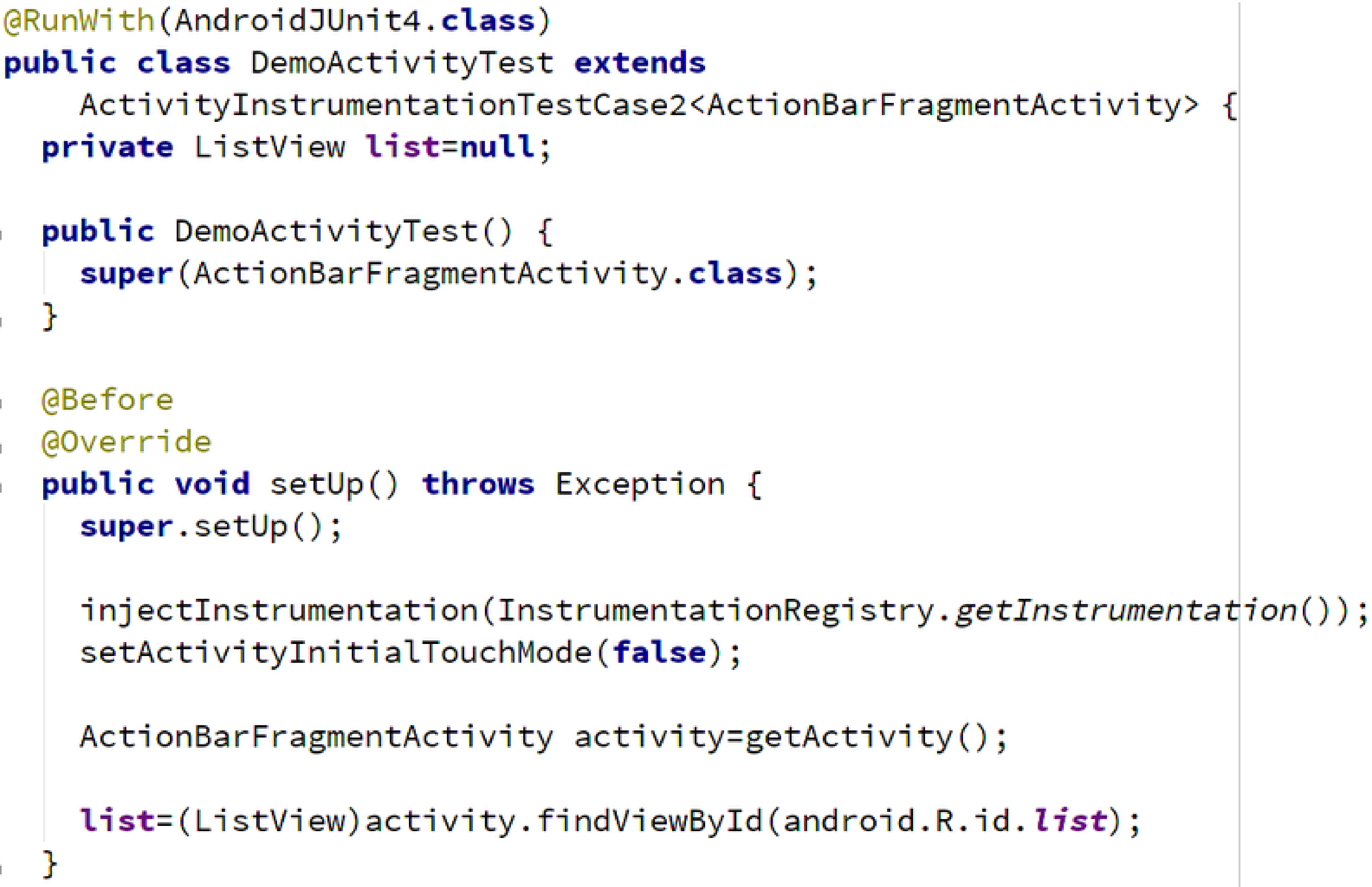
```java
@RunWith(AndroidJUnit4.class)
public class SillyTest {
  @BeforeClass
  static public void doThisFirstOnlyOnce() {
    // do initialization here, run once for all SillyTest tests
  }

  @Before
  public void doThisFirst() {
    // do initialization here, run on every test method
  }
```



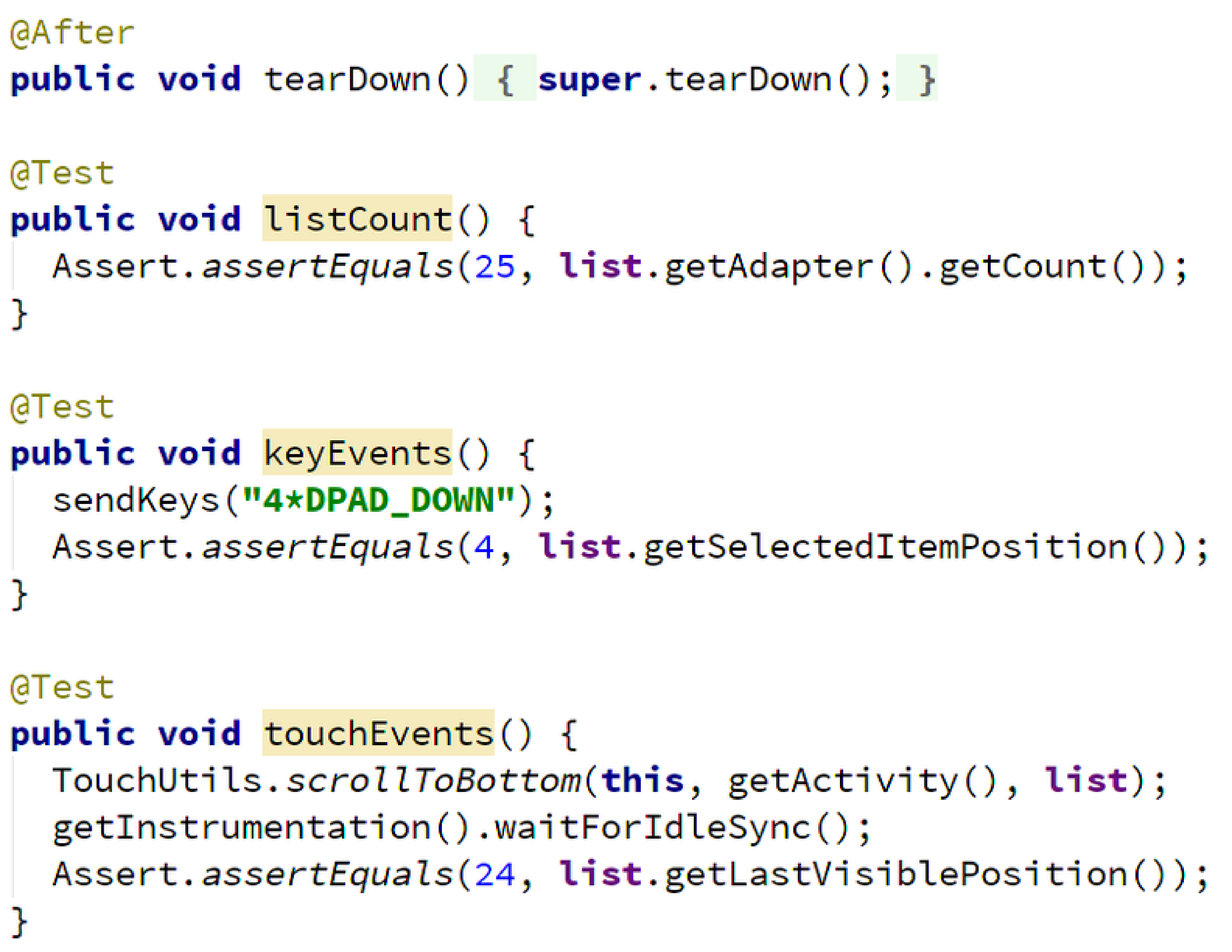
```java
  @After
  public void doThisLast() {
    // do termination here, run on every test method
  }

  @AfterClass
  static public void doThisLastOnlyOnce() {
    // do termination here, run once for all SillyTest tests
  }

  @Test
  public void thisIsReallySilly() {
    Assert.assertEquals("bit got flipped by cosmic rays", 1, 1);
  }
}
```

Step #2: Write Test Cases

- Activities via `ActivityInstrumentationTestCase2`
  - Automatically creates, destroys activity as test runs
  - APIs to inject key, touch events
  - Can access activity to examine child views, etc.
  - JUnit4 setup
    - `@Before`/`@After` on `setUp()`, `tearDown()`
    - `injectInstrumentation()` in `setUp()`
    - `@Test` on test methods



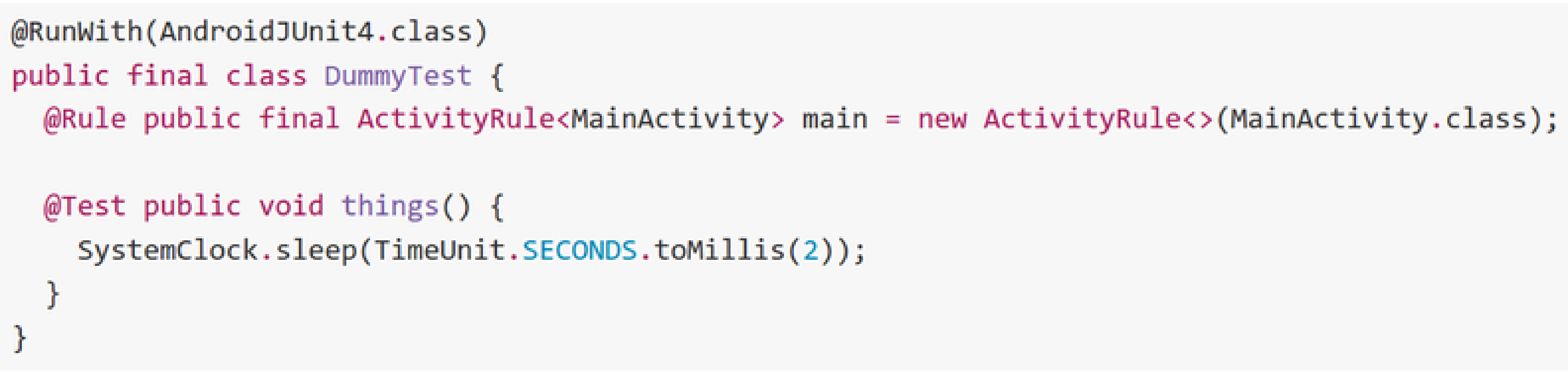
```java
@RunWith(AndroidJUnit4.class)
public class DemoActivityTest extends
    ActivityInstrumentationTestCase2<ActionBarFragmentActivity> {
  private ListView list=null;

  public DemoActivityTest() {
    super(ActionBarFragmentActivity.class);
  }

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

    injectInstrumentation(InstrumentationRegistry.getInstrumentation());
    setActivityInitialTouchMode(false);

    ActionBarFragmentActivity activity=getActivity();

    list=(ListView)activity.findViewById(android.R.id.list);
  }
```



```
@After
public void tearDown() { super.tearDown(); }

@Test
public void listCount() {
  Assert.assertEquals(25, list.getAdapter().getCount());
}

@Test
public void keyEvents() {
  sendKeys("4*DPAD_DOWN");
  Assert.assertEquals(4, list.getSelectedItemPosition());
}

@Test
public void touchEvents() {
  TouchUtils.scrollToBottom(this, getActivity(), list);
  getInstrumentation().waitForIdleSync();
  Assert.assertEquals(24, list.getLastVisiblePosition());
}
```

Step #2: Write Test Cases

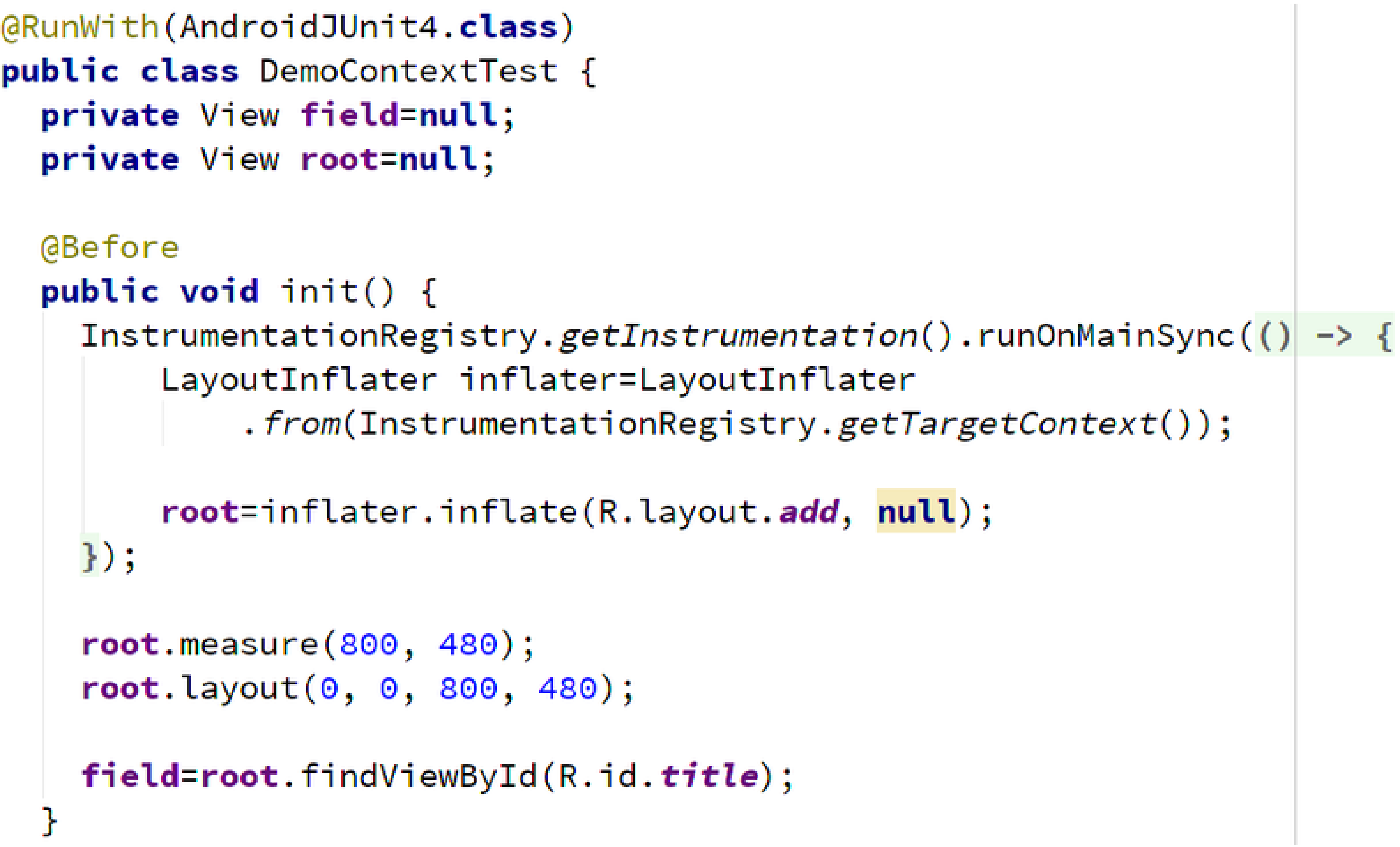
- Activities via ActivityRule
  - `https://gist.github.com/JakeWharton/1c2f2cadab2ddd97f9fb`
  - A theoretically simpler way to write activity tests
  - In practice... well...



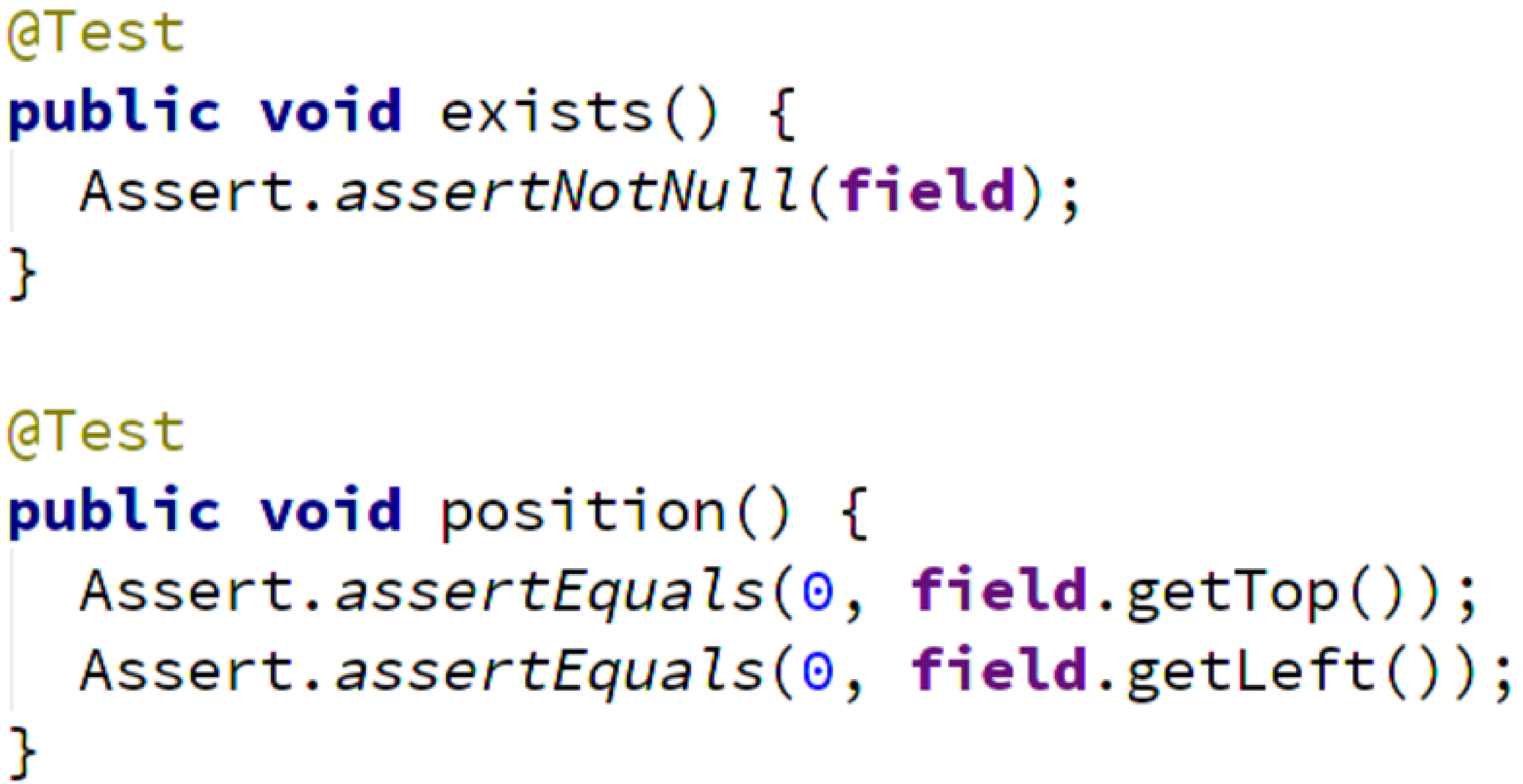
```java
@RunWith(AndroidJUnit4.class)
public final class DummyTest {
  @Rule public final ActivityRule<MainActivity> main = new ActivityRule<>(MainActivity.class);

  @Test public void things() {
    SystemClock.sleep(TimeUnit.SECONDS.toMillis(2));
  }
}
```

Step #2: Write Test Cases

- Testing General Stuff: `AndroidTestCase`
  - Simply provide a `Context` and some scaffolding
  - Good for testing layout inflation, database I/O
  - JUnit4 equivalent: use `InstrumentationRegistry.getTargetContext()`
- Test services via ServiceTestCase, etc.
  - No documented JUnit4 recipes just yet, though should resemble `ActivityInstrumentationTestCase2`



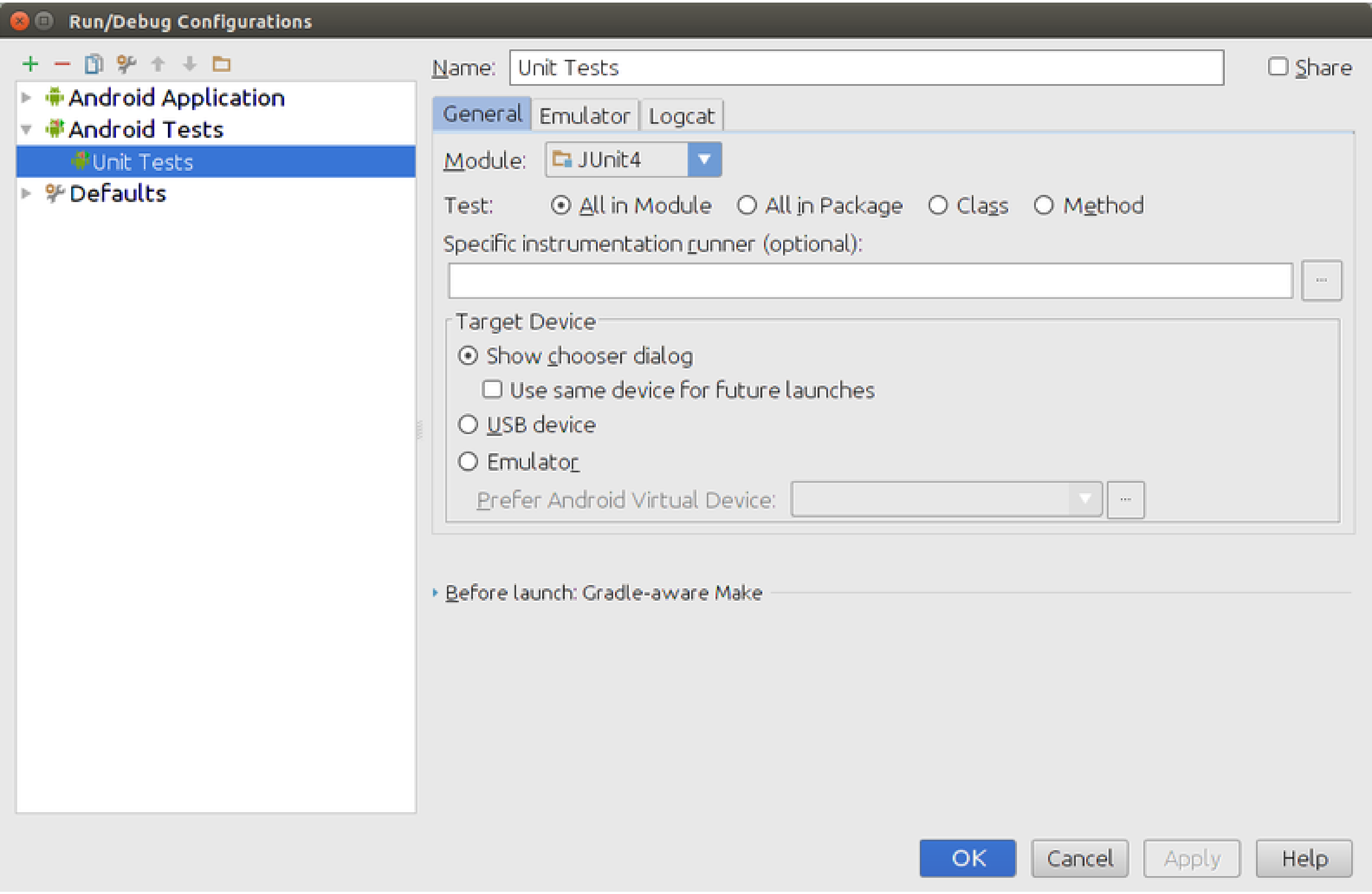
```
@RunWith(AndroidJUnit4.class)
public class DemoContextTest {
  private View field=null;
  private View root=null;

  @Before
  public void init() {
    InstrumentationRegistry.getInstrumentation().runOnMainSync(() -> {
        LayoutInflater inflater=LayoutInflater
            .from(InstrumentationRegistry.getTargetContext());

        root=inflater.inflate(R.layout.add, null);
    });

    root.measure(800, 480);
    root.layout(0, 0, 800, 480);

    field=root.findViewById(R.id.title);
  }
```



```java
@Test
public void exists() {
  Assert.assertNotNull(field);
}

@Test
public void position() {
  Assert.assertEquals(0, field.getTop());
  Assert.assertEquals(0, field.getLeft());
}
```

Step #3: Run the Test Cases

- Android Studio
  - Set Up Android Studio Run Configuration
    - "Android Tests", with scope, etc.
    - Override default test runner from Gradle
  - Run the Tests
    - Run the configuration you just defined
    - Results in dedicated GUI
- Gradle
  - `gradle connectedCheck` and kin



Run/Debug Configurations

- Android Application
- Android Tests
  - Unit Tests
- Defaults

Name: Unit Tests [ ] Share

General | Emulator | Logcat

Module: JUnit4

Test: [✓ All in Module] All in Package Class Method

Specific instrumentation runner (optional):

Target Device

[✓ Show chooser dialog]

[ ] Use same device for future launches

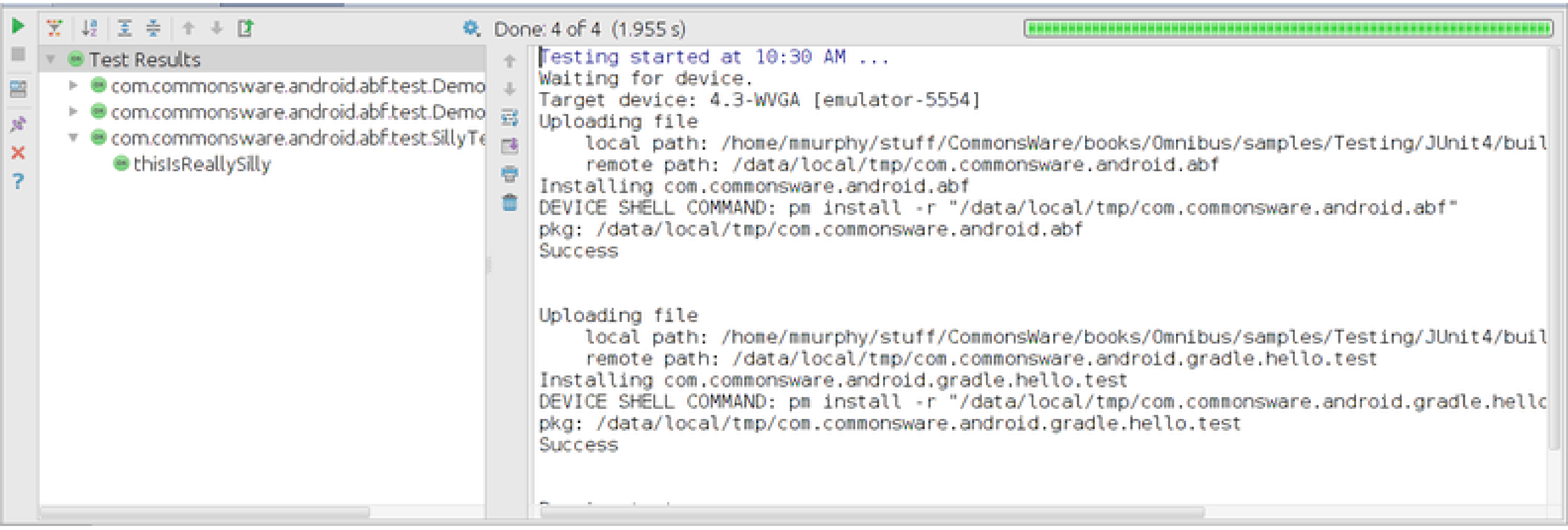
USB device

Emulator

Prefer Android Virtual Device:

Before launch: Gradle-aware Make

OK Cancel Apply Help


Done: 4 of 4 (1.955 s)
Test Results
com.commonsware.android.abf.test.Demo
com.commonsware.android.abf.test.Demo
com.commonsware.android.abf.test.SillyTe
thisIsReallySilly

```
Testing started at 10:30 AM ...
Waiting for device.
Target device: 4.3-WVGA [emulator-5554]
Uploading file
    local path: /home/mmurphy/stuff/CommonsWare/books/Omnibus/samples/Testing/JUnit4/buil
    remote path: /data/local/tmp/com.commonsware.android.abf
Installing com.commonsware.android.abf
DEVICE SHELL COMMAND: pm install -r "/data/local/tmp/com.commonsware.android.abf"
pkg: /data/local/tmp/com.commonsware.android.abf
Success


Uploading file
    local path: /home/mmurphy/stuff/CommonsWare/books/Omnibus/samples/Testing/JUnit4/buil
    remote path: /data/local/tmp/com.commonsware.android.gradle.hello.test
Installing com.commonsware.android.gradle.hello.test
DEVICE SHELL COMMAND: pm install -r "/data/local/tmp/com.commonsware.android.gradle.hello
pkg: /data/local/tmp/com.commonsware.android.gradle.hello.test
Success
```

Related JUnit Options

- uiautomator
  - With 2.0, lives as instrumentation tests alongside your more traditional test code
- Unit Testing
  - For testing code via mocks and stubs, running on your development machine, versus running in Android
  - Requires Android Studio 1.1+, Gradle for Android 1.1+