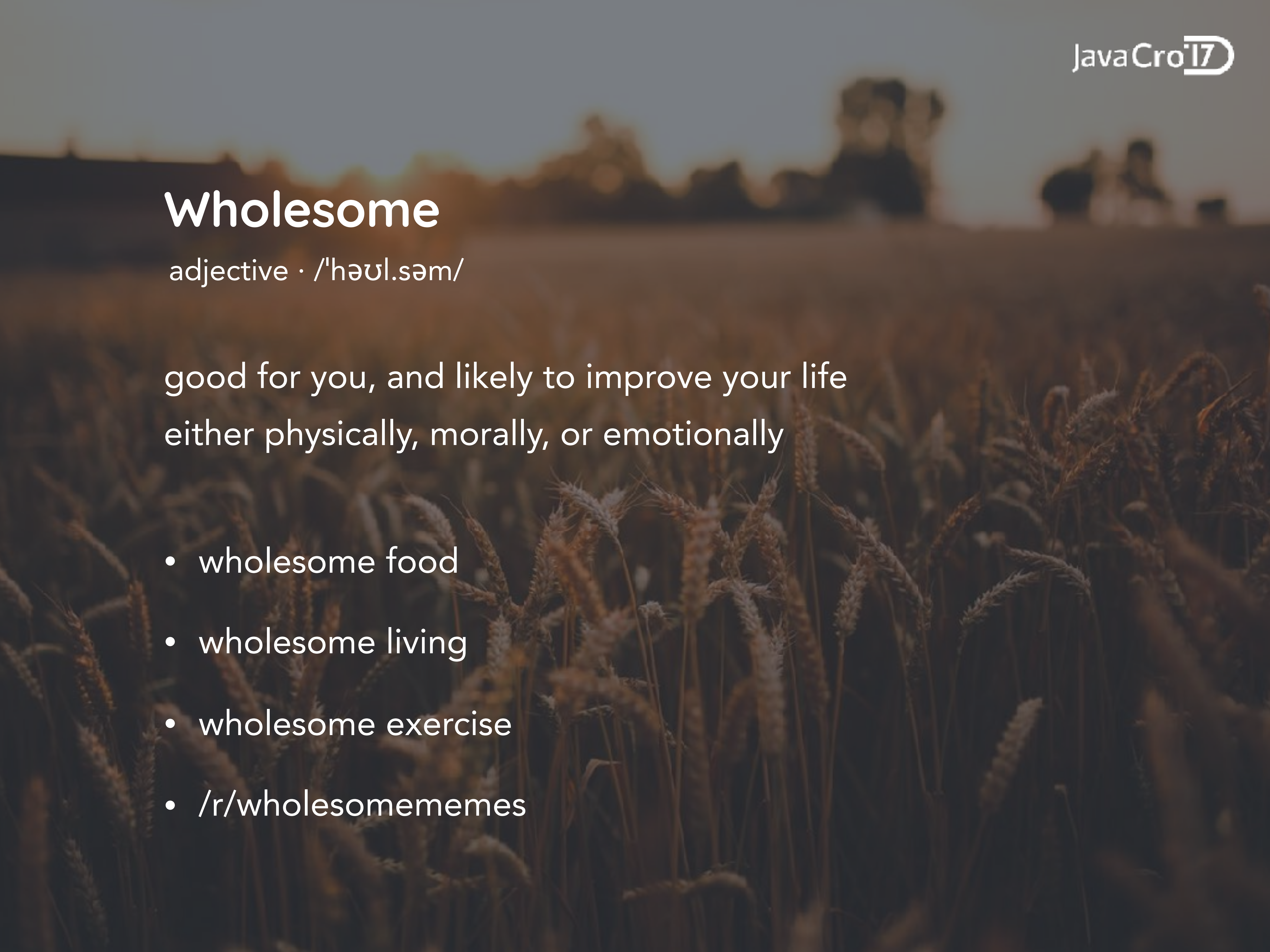
JavaCro'17

Wholesome tests on Android

Dejan Tošić

Undabot



Wholesome

adjective · /ˈhəʊl.səm/

good for you, and likely to improve your life either physically, morally, or emotionally

- wholesome food
- wholesome living
- wholesome exercise
- /r/wholesomememes

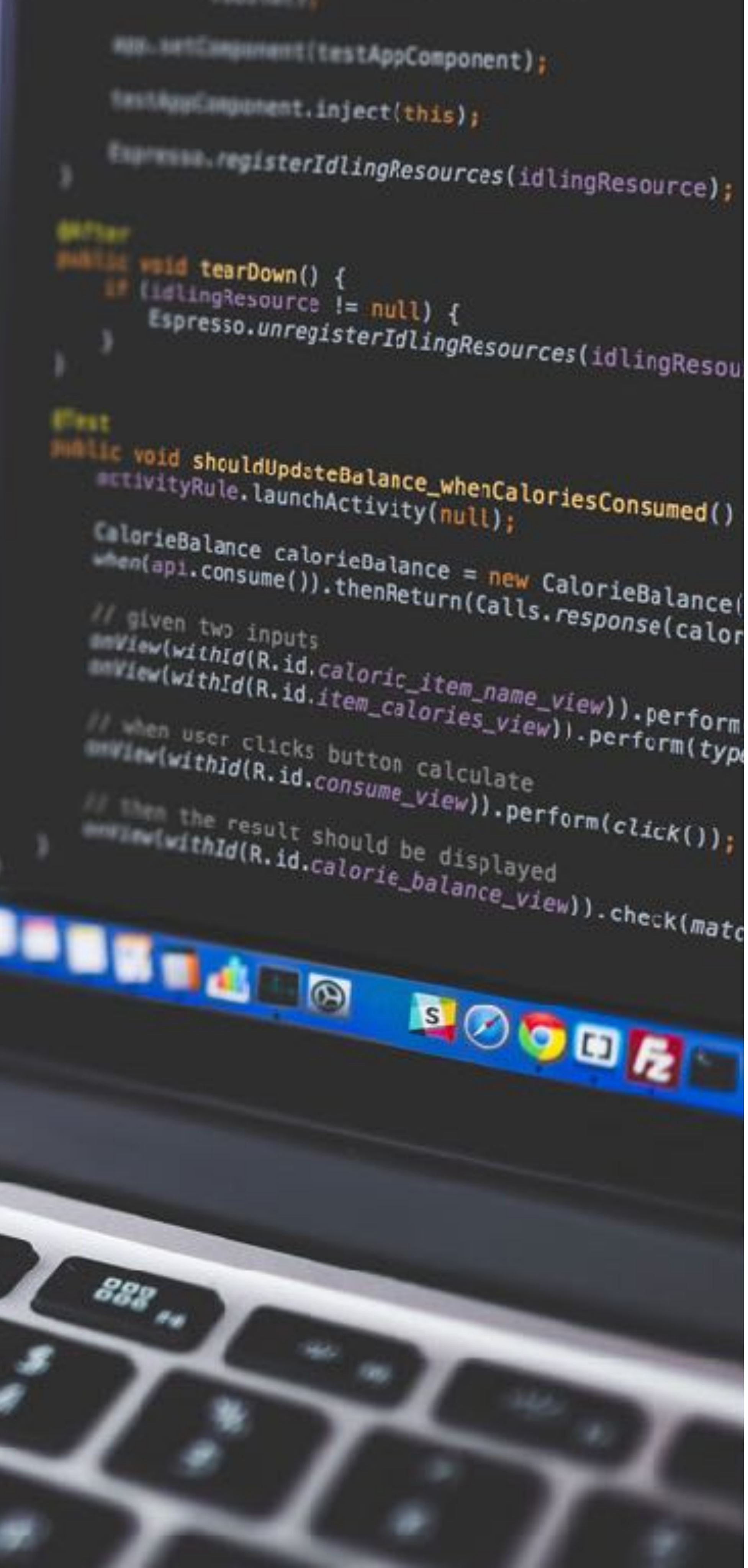
Wholesome tests

PROJECT PERSPECTIVE

- verification
- confidence
- prevent regression
- speed when refactoring
- documentation

DEVELOPER PERSPECTIVE

- stress reducing
- maintainable
- fast
- aligned with our development process
- keep our sanity in check

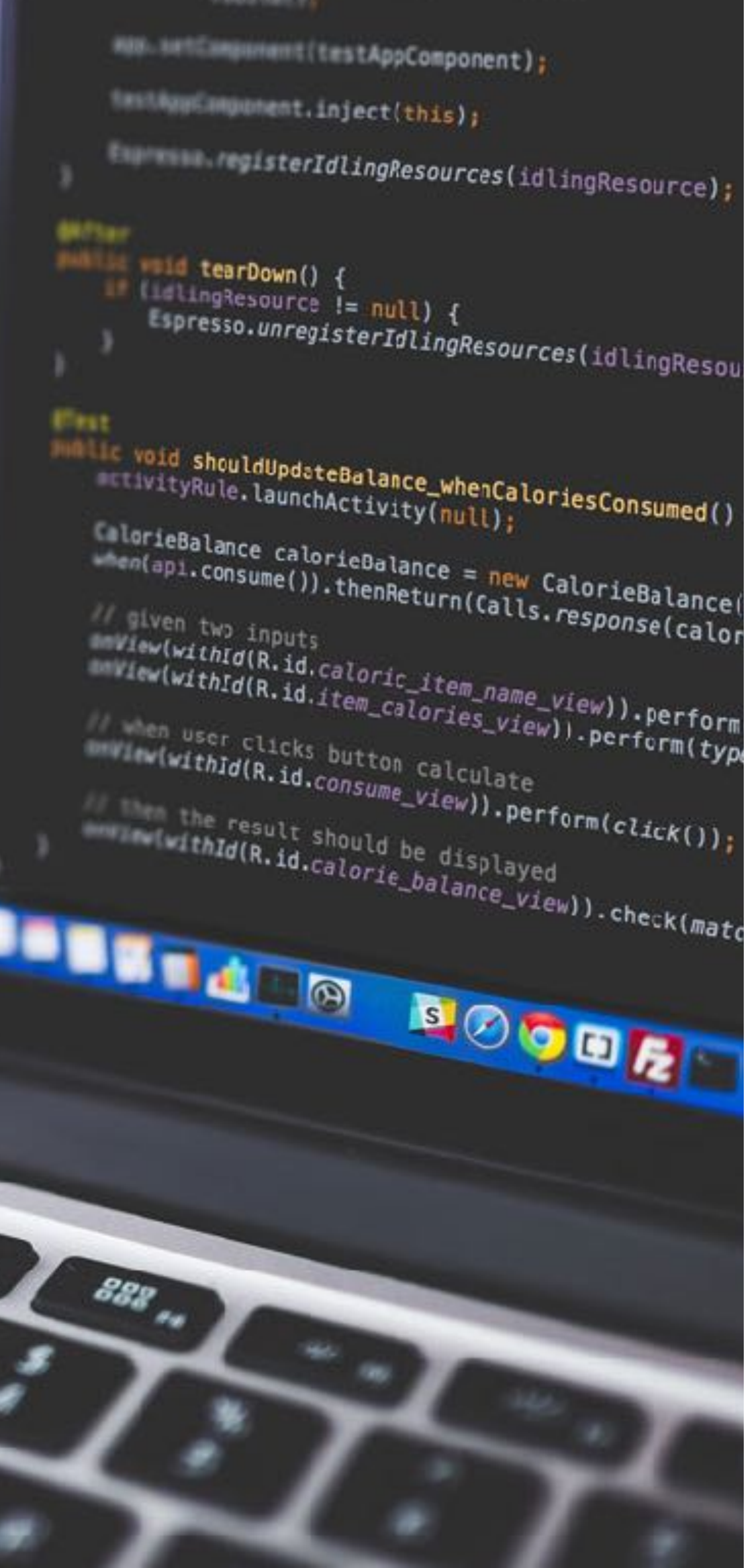


```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ

    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

JavaCro'17

Testing in Android

TESTS SHOULD BE

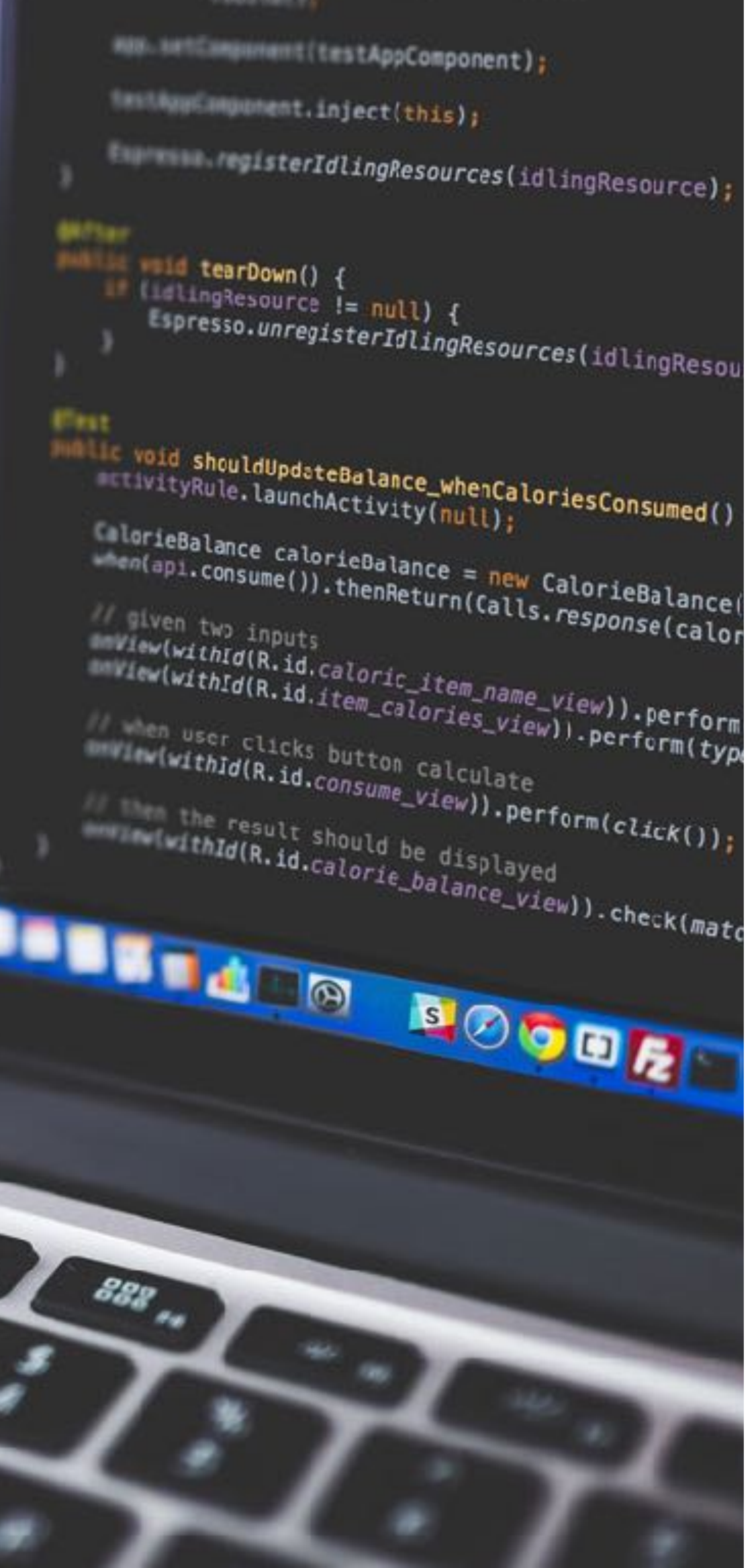


```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Testing in Android

TESTS SHOULD BE

- readable

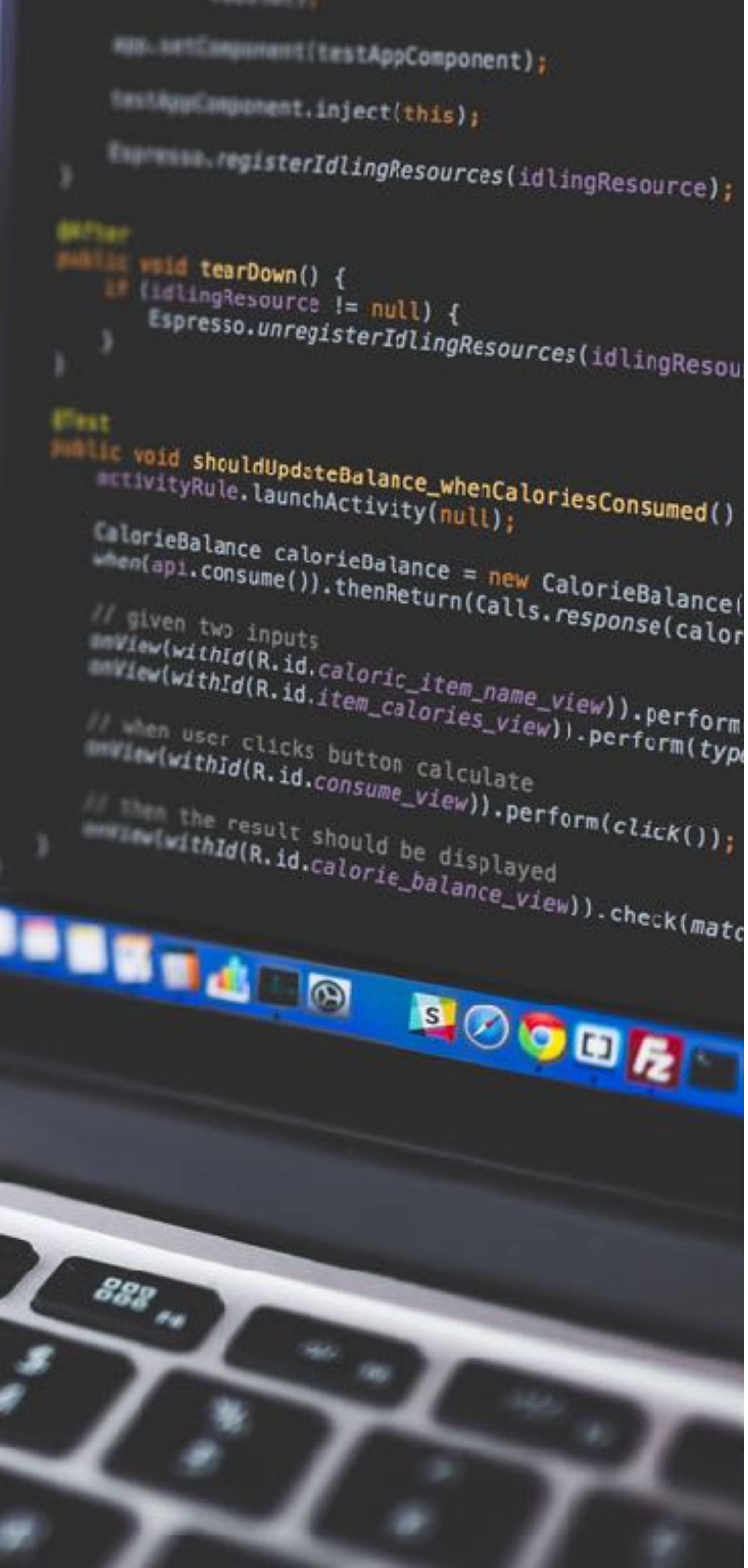


```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Testing in Android

TESTS SHOULD BE

- readable ✔



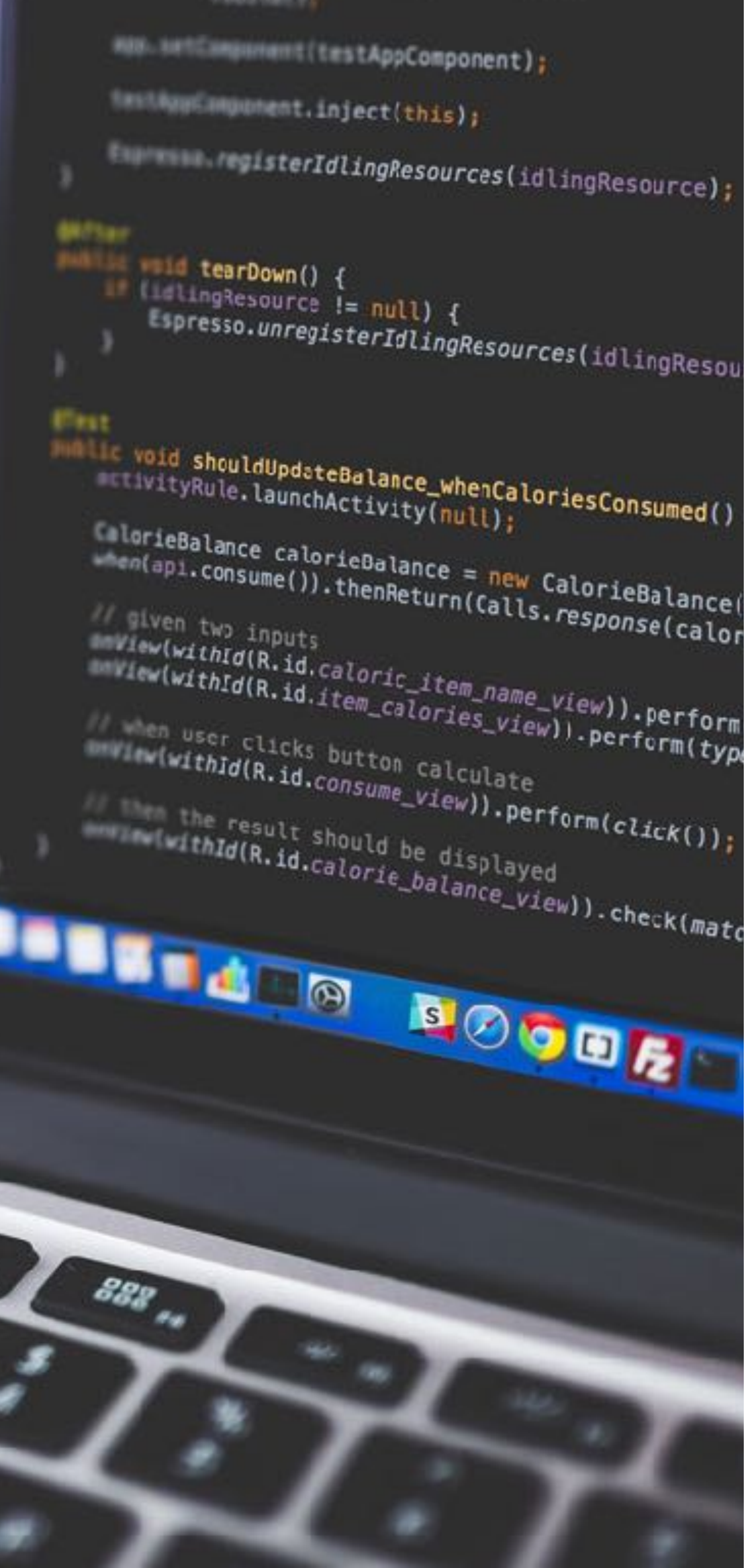
```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ

    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy

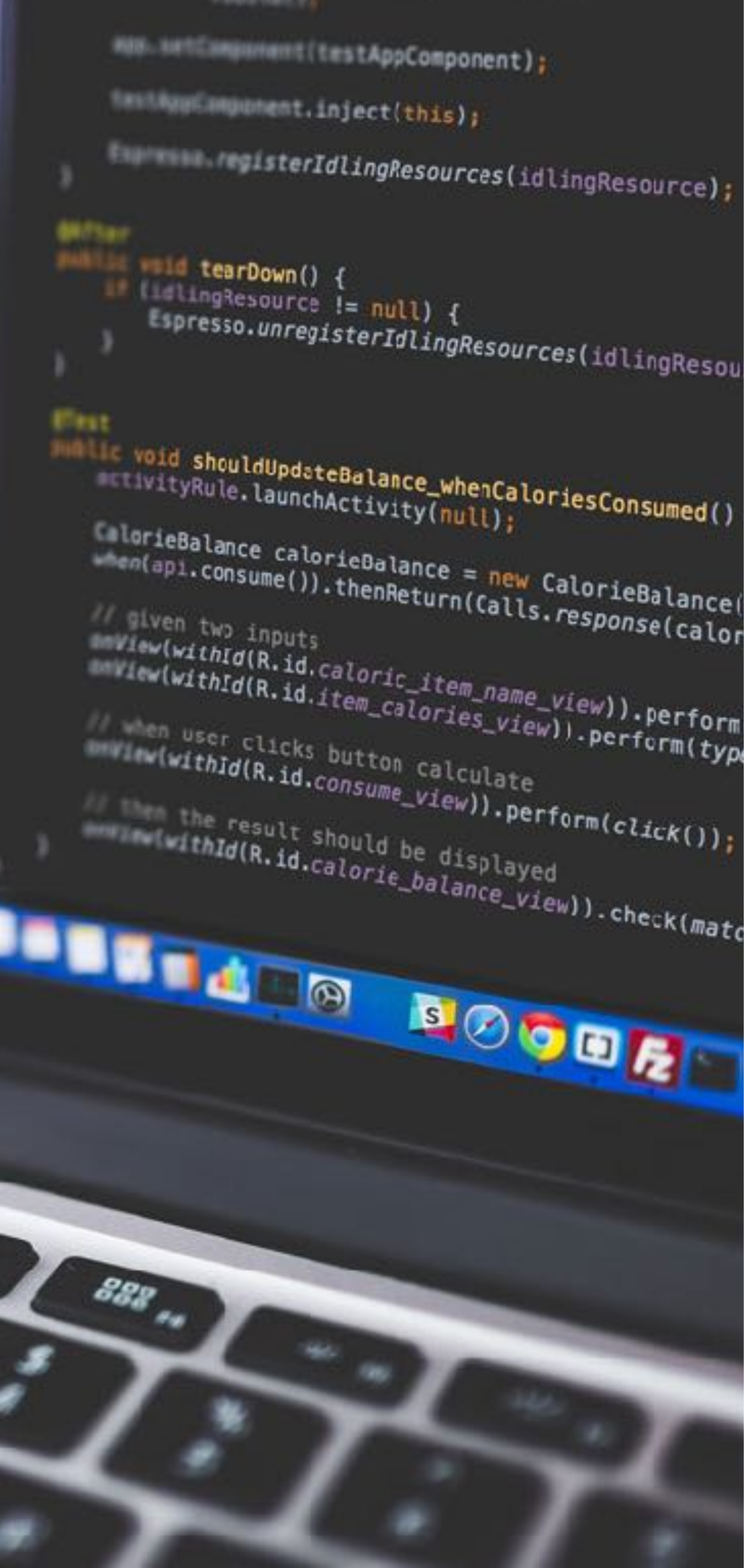
Testing in Android



```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);

        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor

        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(type
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

TESTS SHOULD BE

- readable ✔
- trustworthy ✔



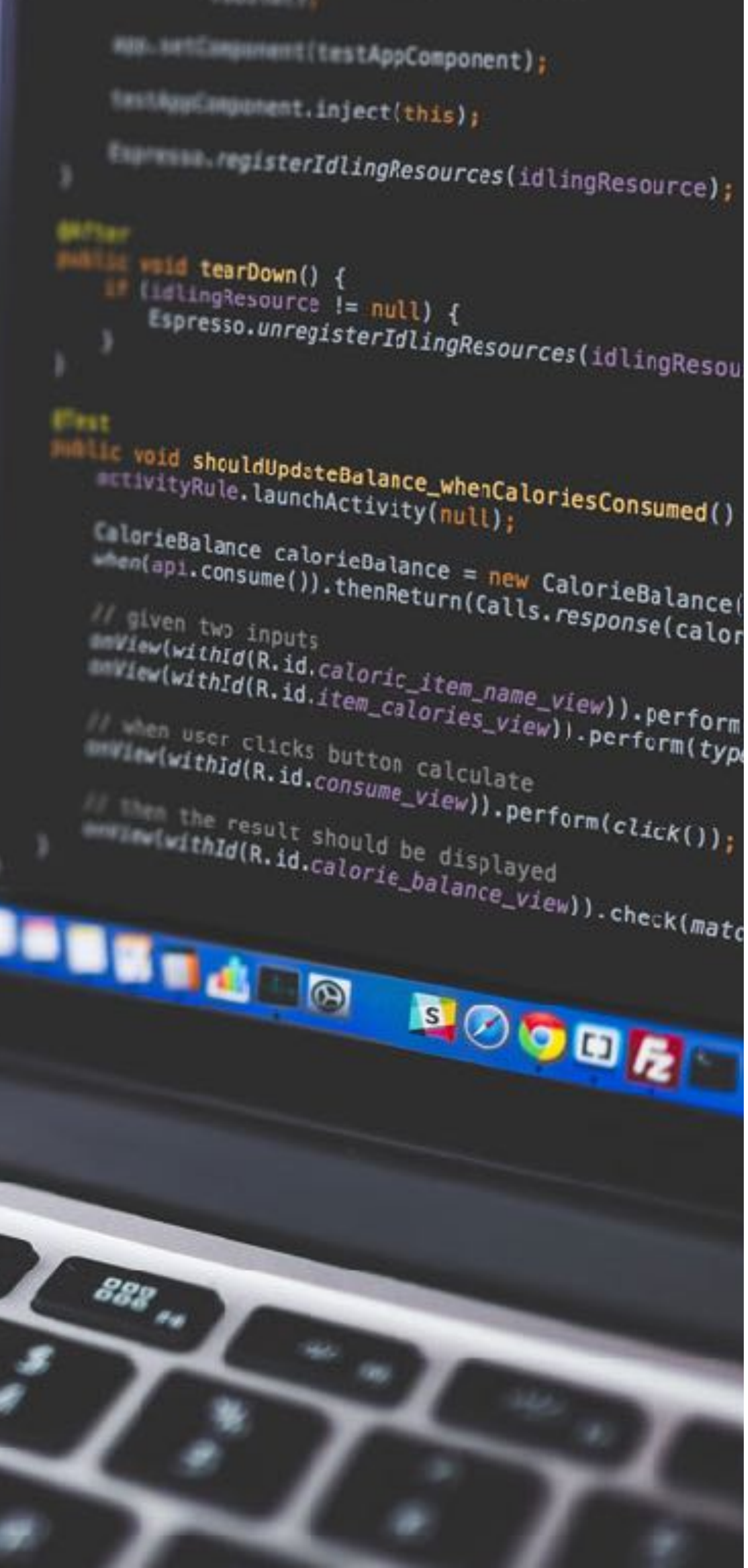
```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}
@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}
@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

JavaCro'17

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive



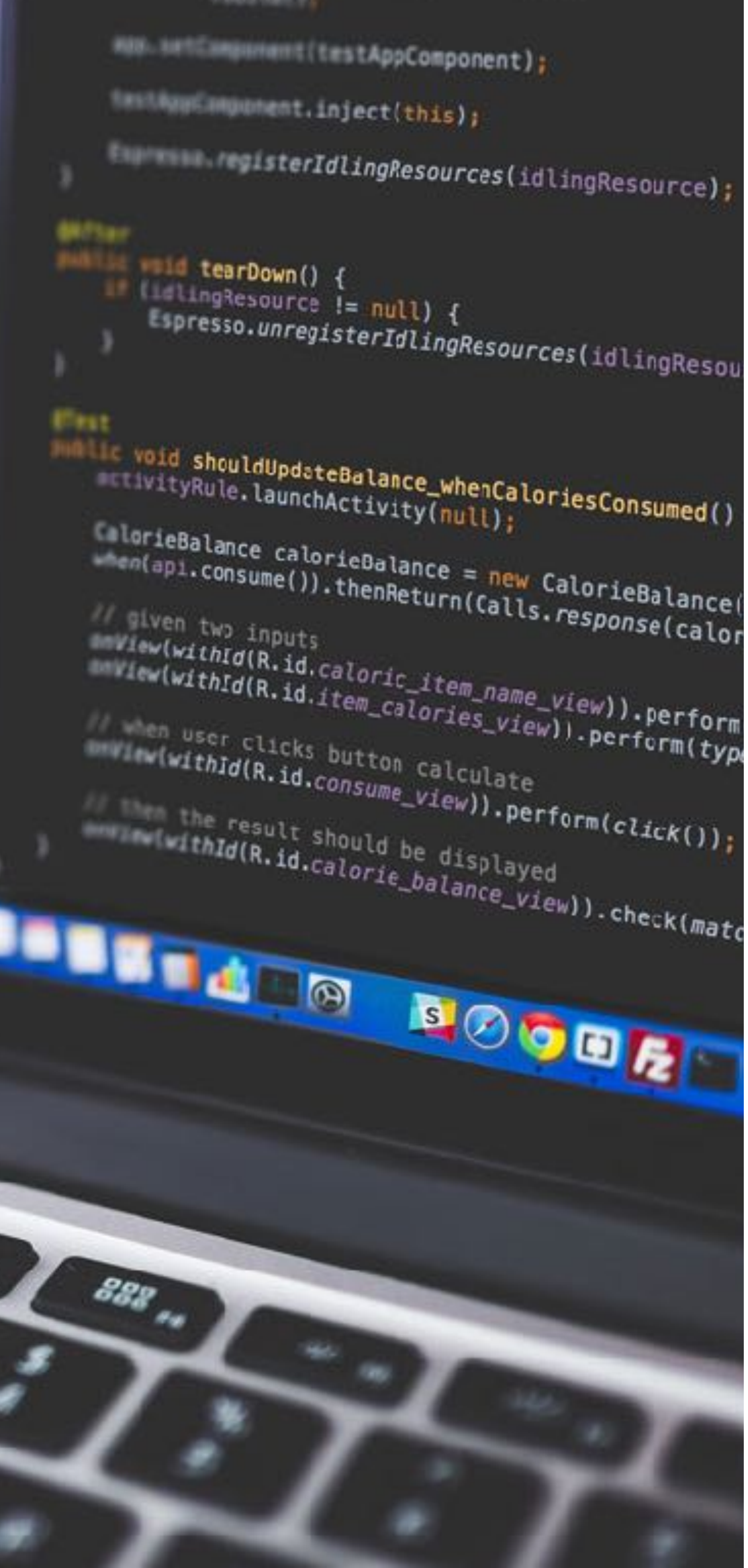
```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

JavaCro'17

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔

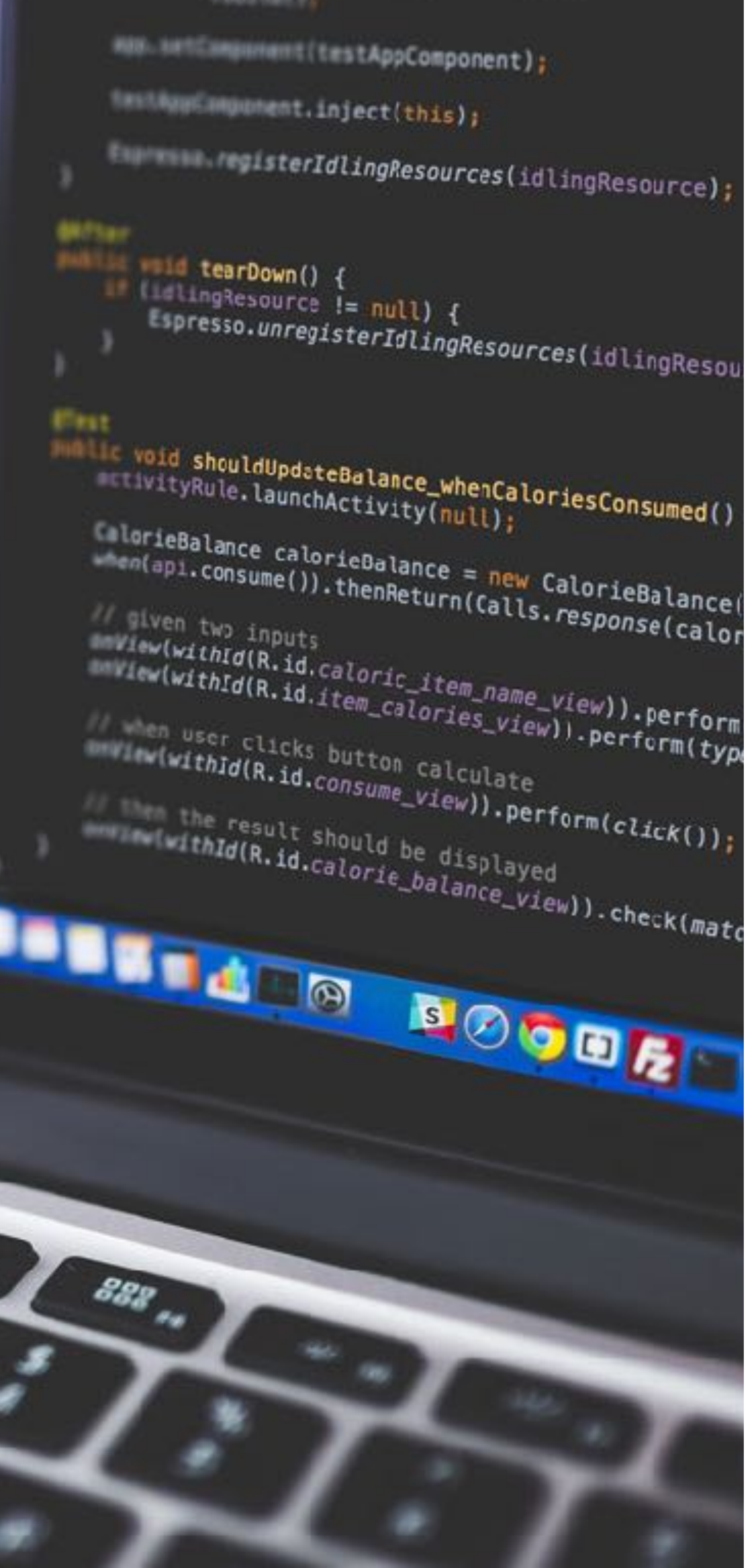


```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}
@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}
@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast



```
app.setComponent(testAppComponent);
testAppComponent.inject(this);
Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

JavaCro'17

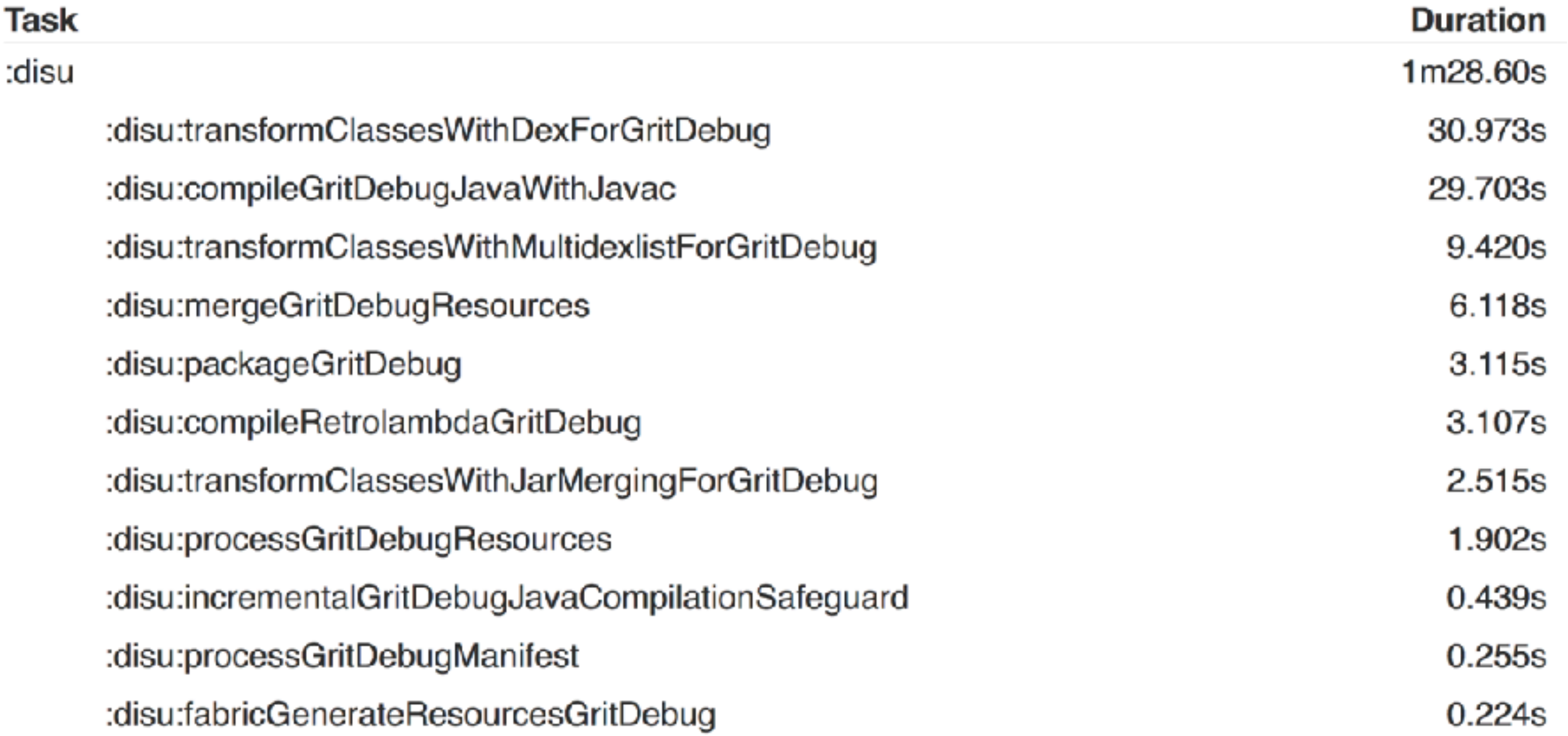
Testing in Android

TESTS SHOULD BE

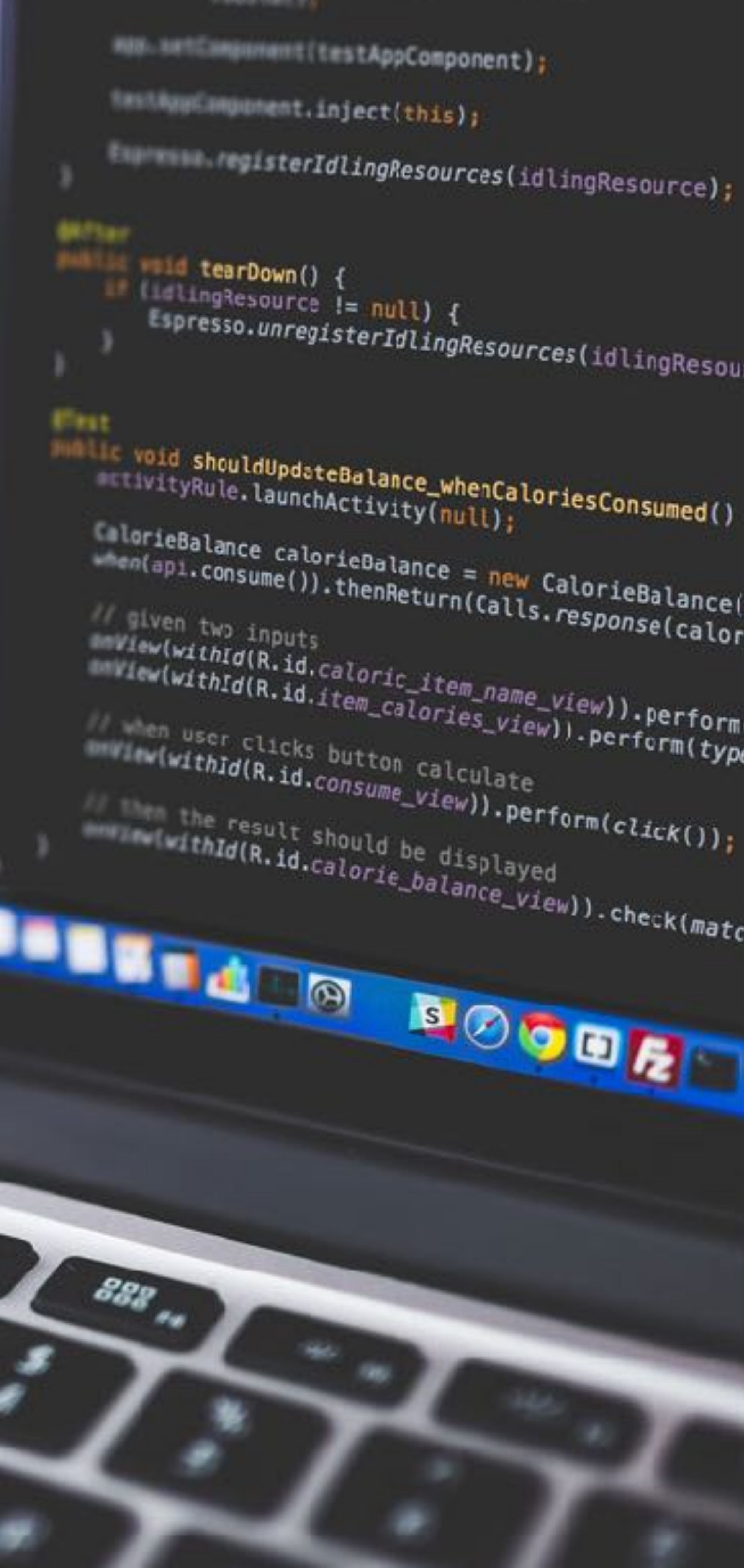
- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast

    └ TDD

Gradle



Task	Duration
:disu	1m28.60s
:disu:transformClassesWithDexForGritDebug	30.973s
:disu:compileGritDebugJavaWithJavac	29.703s
:disu:transformClassesWithMultidexlistForGritDebug	9.420s
:disu:mergeGritDebugResources	6.118s
:disu:packageGritDebug	3.115s
:disu:compileRetrolambdaGritDebug	3.107s
:disu:transformClassesWithJarMergingForGritDebug	2.515s
:disu:processGritDebugResources	1.902s
:disu:incrementalGritDebugJavaCompilationSafeguard	0.439s
:disu:processGritDebugManifest	0.255s
:disu:fabricGenerateResourcesGritDebug	0.224s



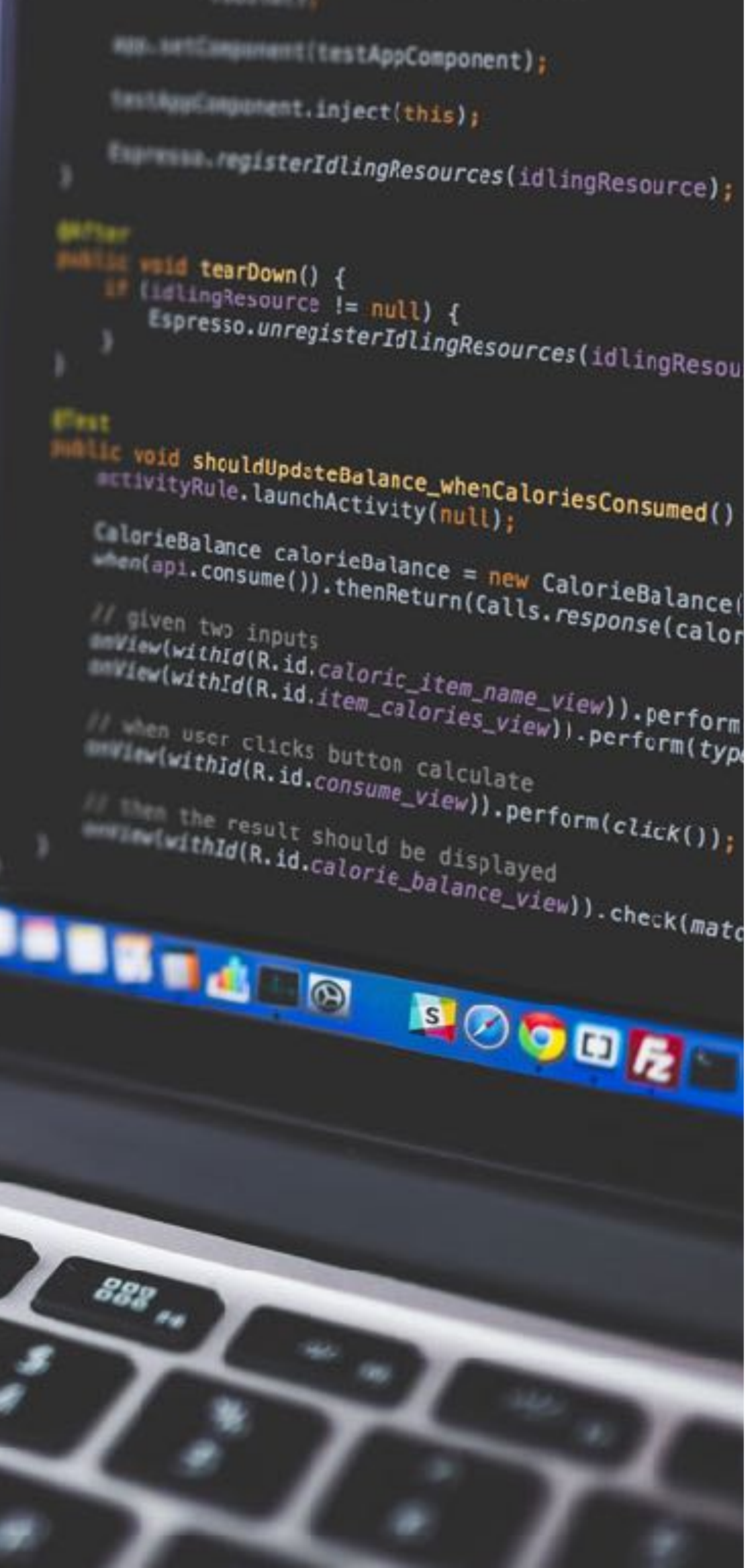
```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

JavaCro'17

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast
  - ┗ TDD

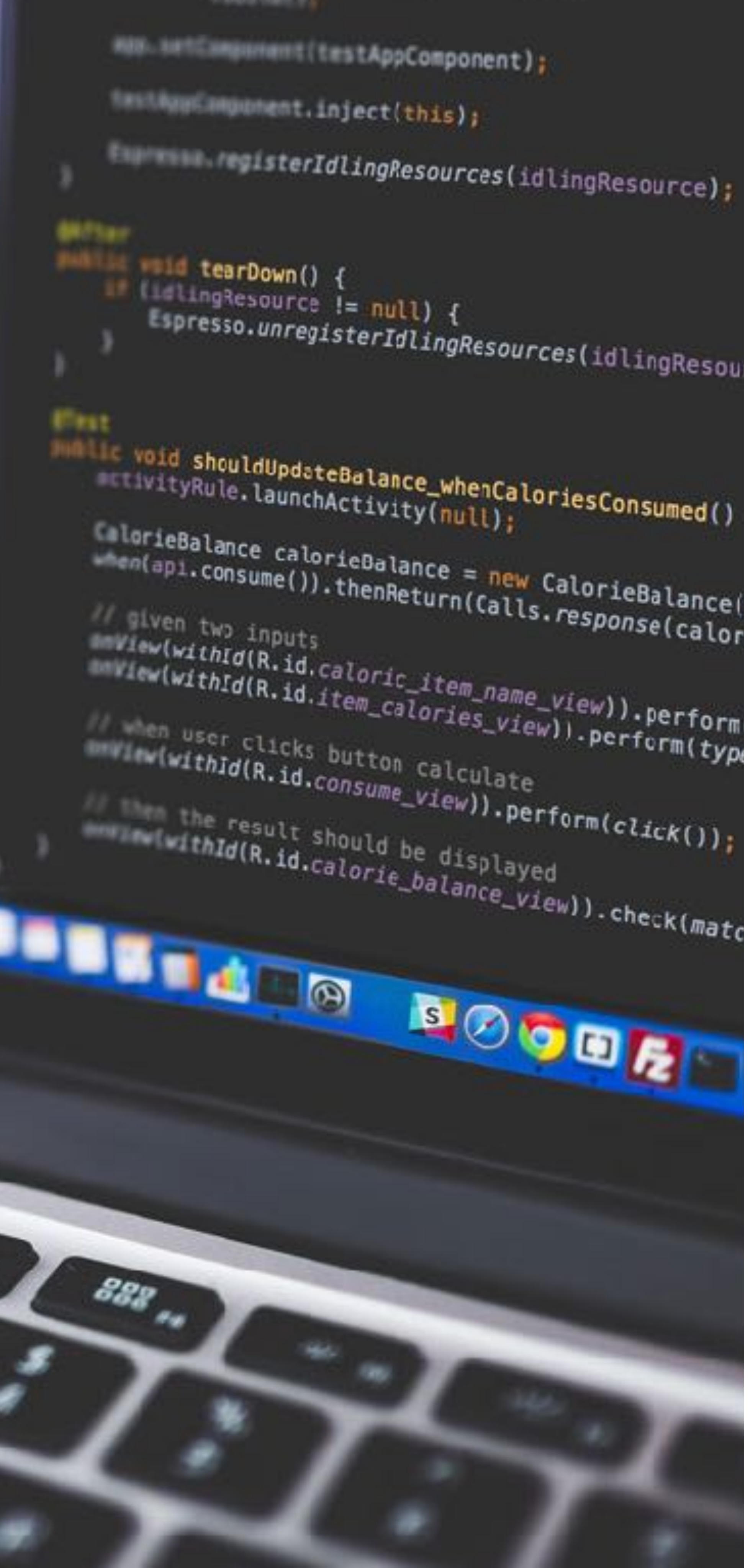


```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ

    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast ?
  - ┗ TDD



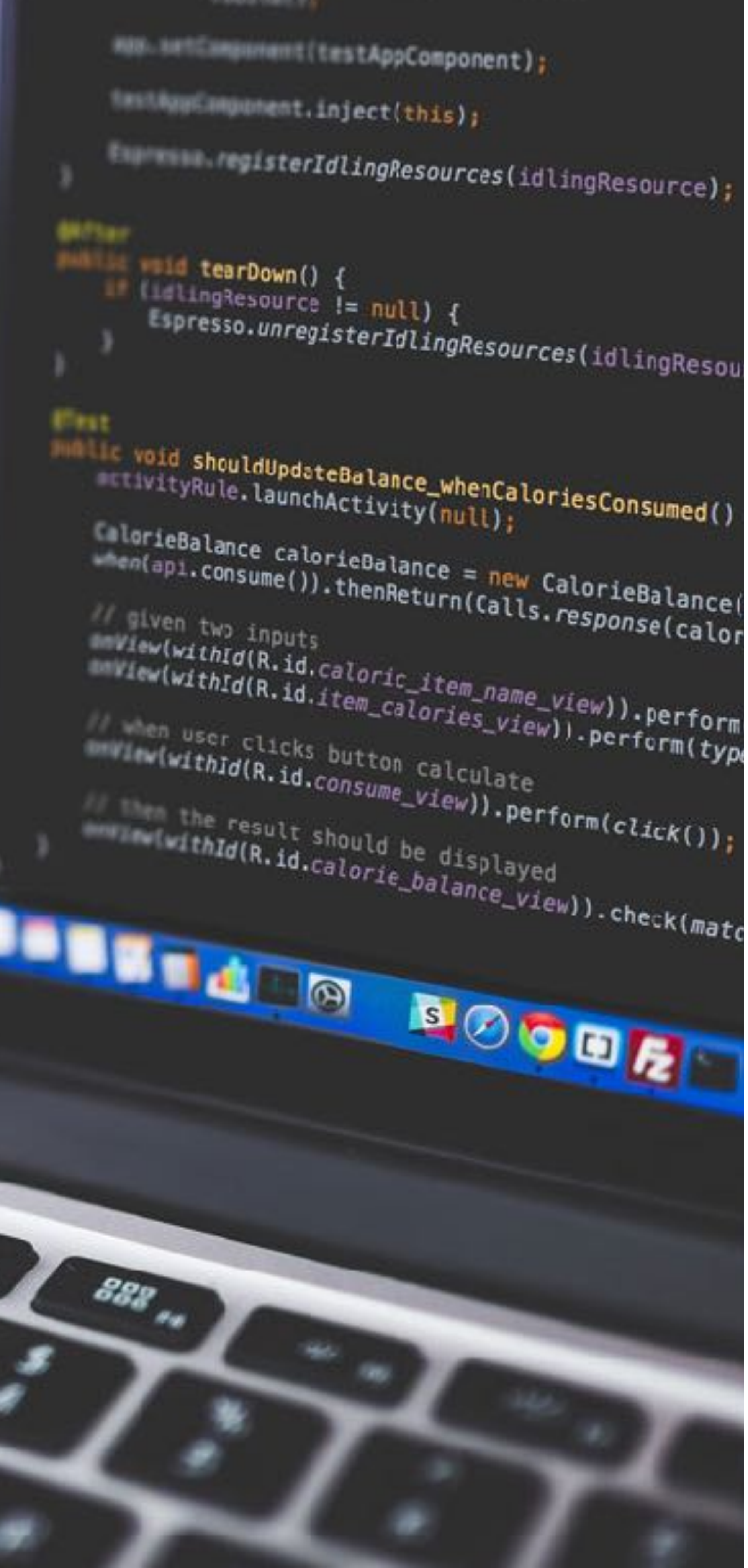
```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);
        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor
        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(typ
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast ?

  └ TDD ?



```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);
        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor
        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(type
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

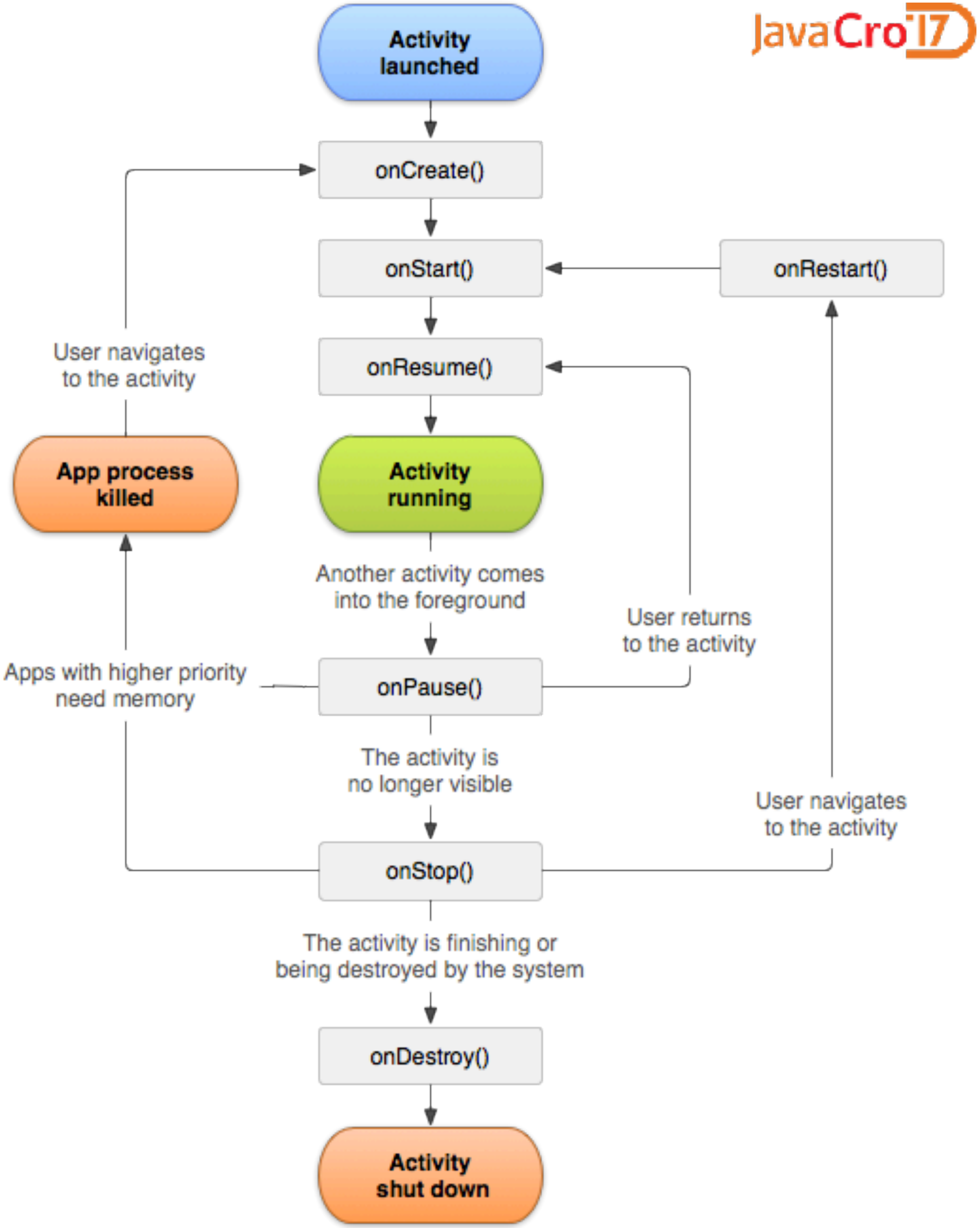
JavaCro'17

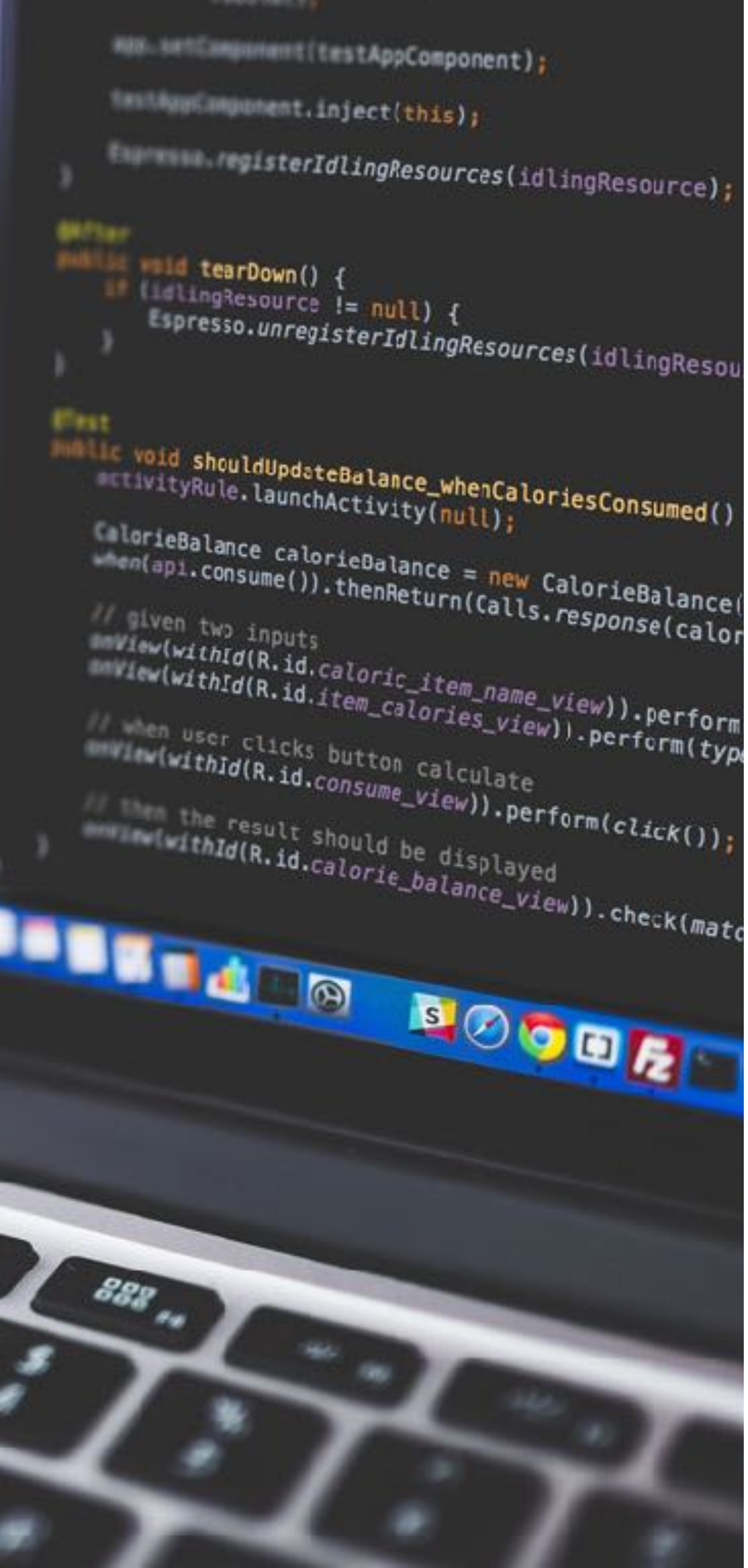
Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast ?
  - ┗ TDD ?
- isolated / decoupled

Android lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down



```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);
        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor
        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(type
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

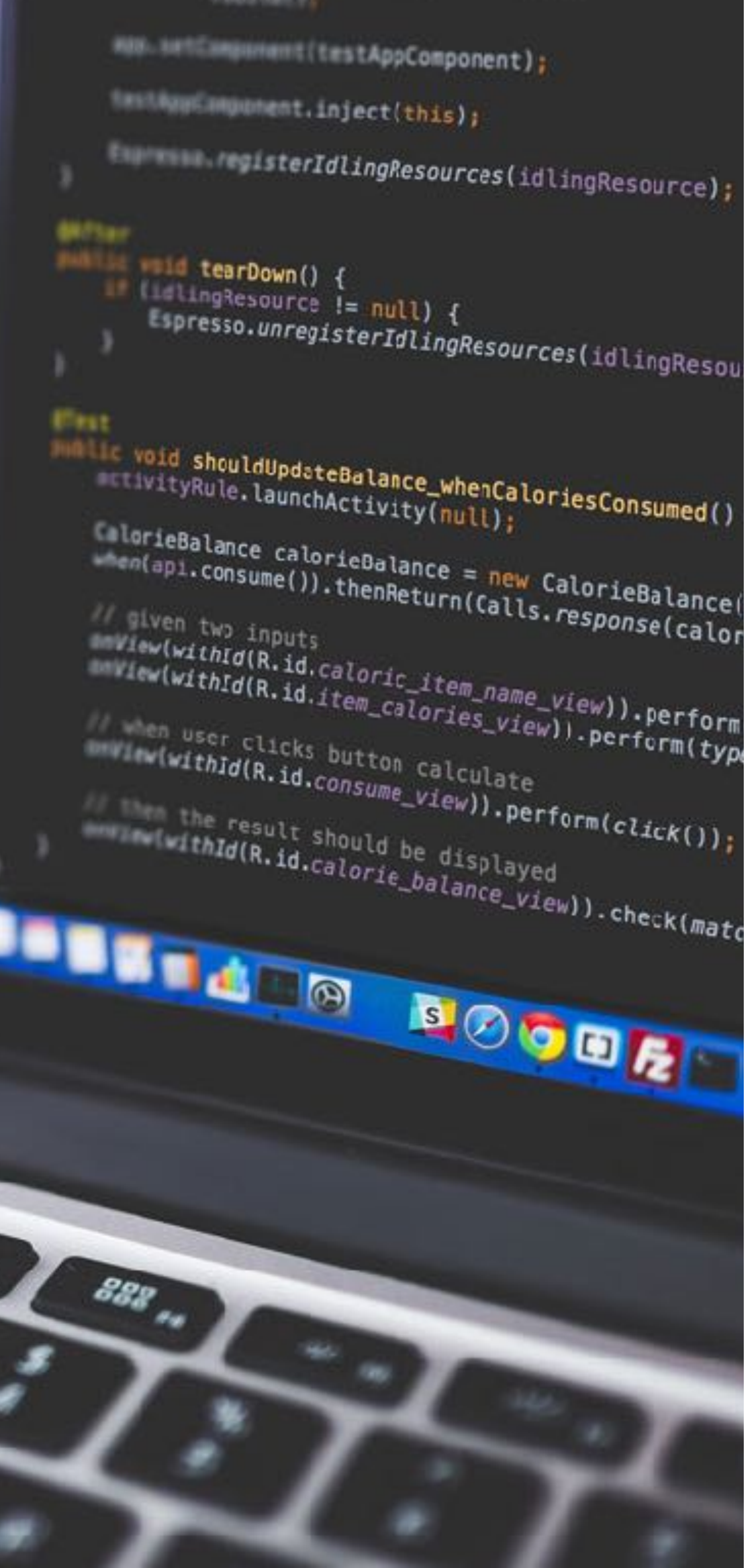
JavaCro'17

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast ?

  └ TDD ?
- isolated / decoupled ? ✔

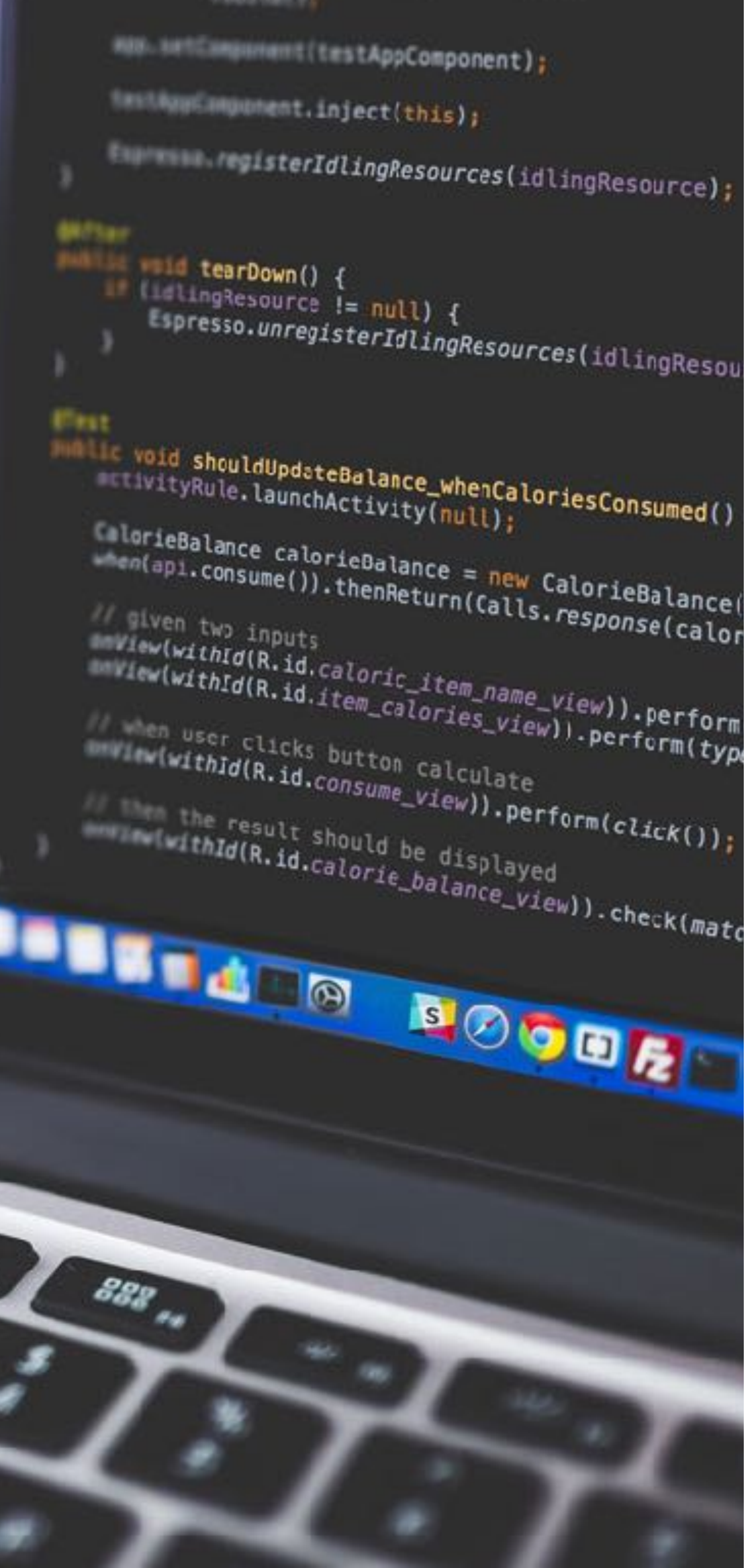


```
app.setComponent(testAppComponent);
testAppComponent.inject(this);
Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ

    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast ?
  - └ TDD ?
- isolated / decoupled ? ✔
- **maintainable**

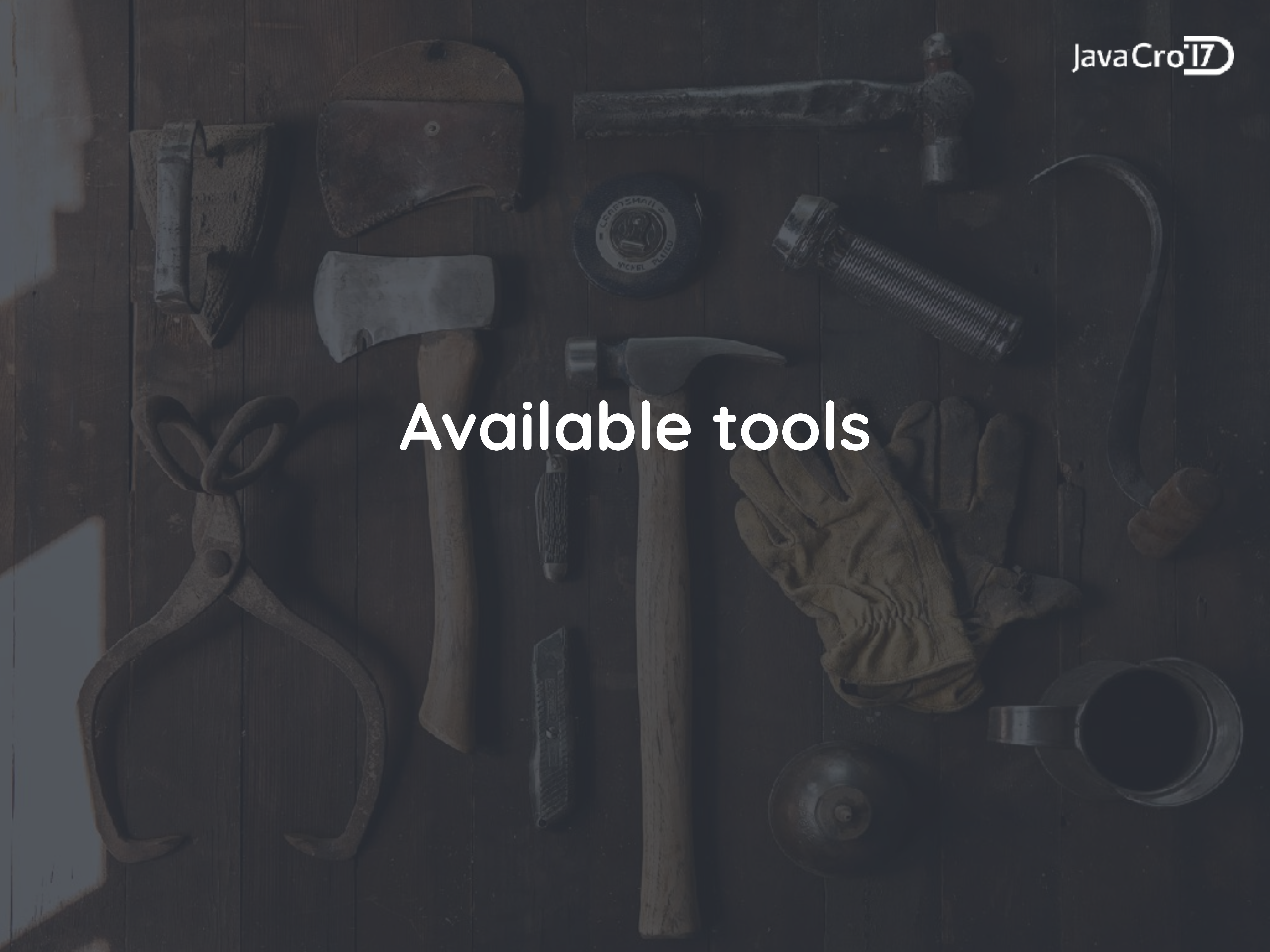


```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);
        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor
        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(type
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

Testing in Android

TESTS SHOULD BE

- readable ✔
- trustworthy ✔
- comprehensive ✔
- fast ?
  - └ TDD ?
- isolated / decoupled ? ✔
- **maintainable** ?



Available tools

Available tools

Available tools

JUnit

ROBOLECTRIC

UI TESTING FOR ANDROID

espresso

mockito

UI Automator

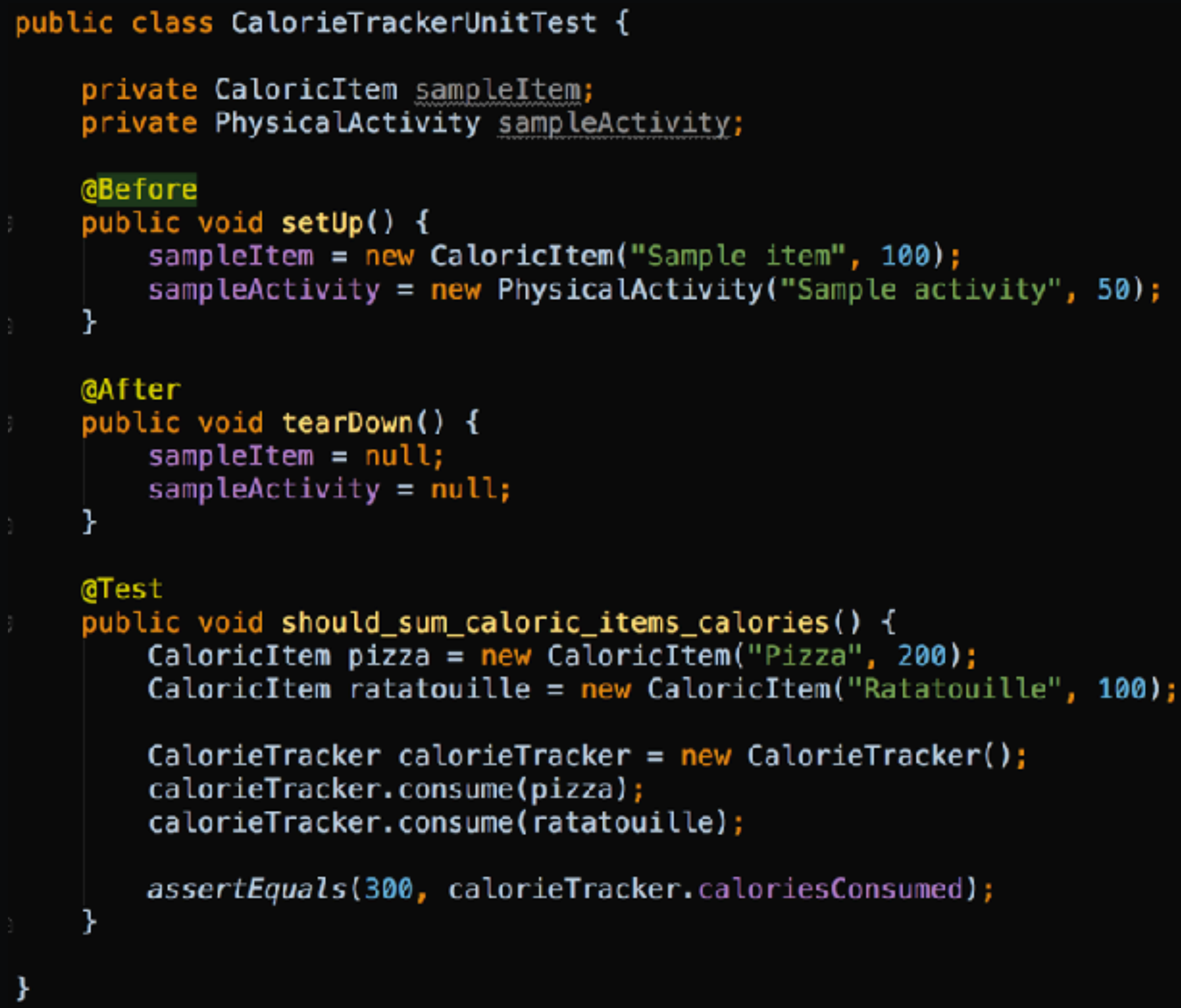
Spoon

JUnit

https://github.com/junit-team/junit4

- fast
- run on JVM
- no Android framework

JUnit test

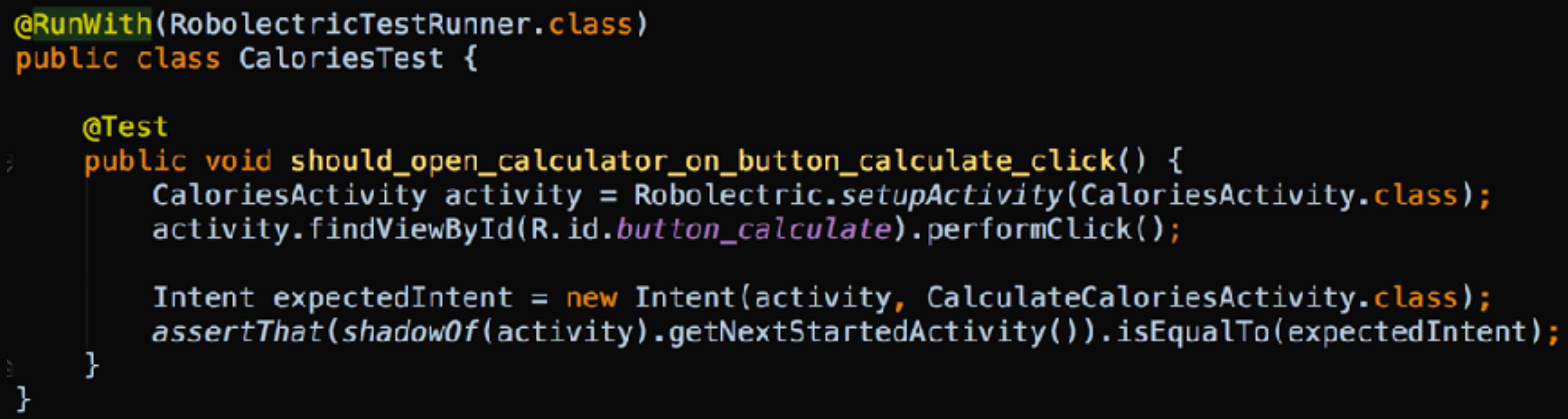


```java
public class CalorieTrackerUnitTest {

    private CaloricItem sampleItem;
    private PhysicalActivity sampleActivity;

    @Before
    public void setUp() {
        sampleItem = new CaloricItem("Sample item", 100);
        sampleActivity = new PhysicalActivity("Sample activity", 50);
    }

    @After
    public void tearDown() {
        sampleItem = null;
        sampleActivity = null;
    }

    @Test
    public void should_sum_caloric_items_calories() {
        CaloricItem pizza = new CaloricItem("Pizza", 200);
        CaloricItem ratatouille = new CaloricItem("Ratatouille", 100);

        CalorieTracker calorieTracker = new CalorieTracker();
        calorieTracker.consume(pizza);
        calorieTracker.consume(ratatouille);

        assertEquals(300, calorieTracker.caloriesConsumed);
    }

}
```

http://robolectric.org/

- unit test framework that simulates Android framework
- runs on JVM
- faster than Espresso
- not all framework features are supported
- tests not reliable/trustworthy

Robolectric test



```
@RunWith(RobolectricTestRunner.class)
public class CaloriesTest {

    @Test
    public void should_open_calculator_on_button_calculate_click() {
        CaloriesActivity activity = Robolectric.setupActivity(CaloriesActivity.class);
        activity.findViewById(R.id.button_calculate).performClick();

        Intent expectedIntent = new Intent(activity, CalculateCaloriesActivity.class);
        assertThat(shadowOf(activity).getNextStartedActivity()).isEqualTo(expectedIntent);
    }
}
```

Android Testing Support Library

https://google.github.io/android-testing-support-library/

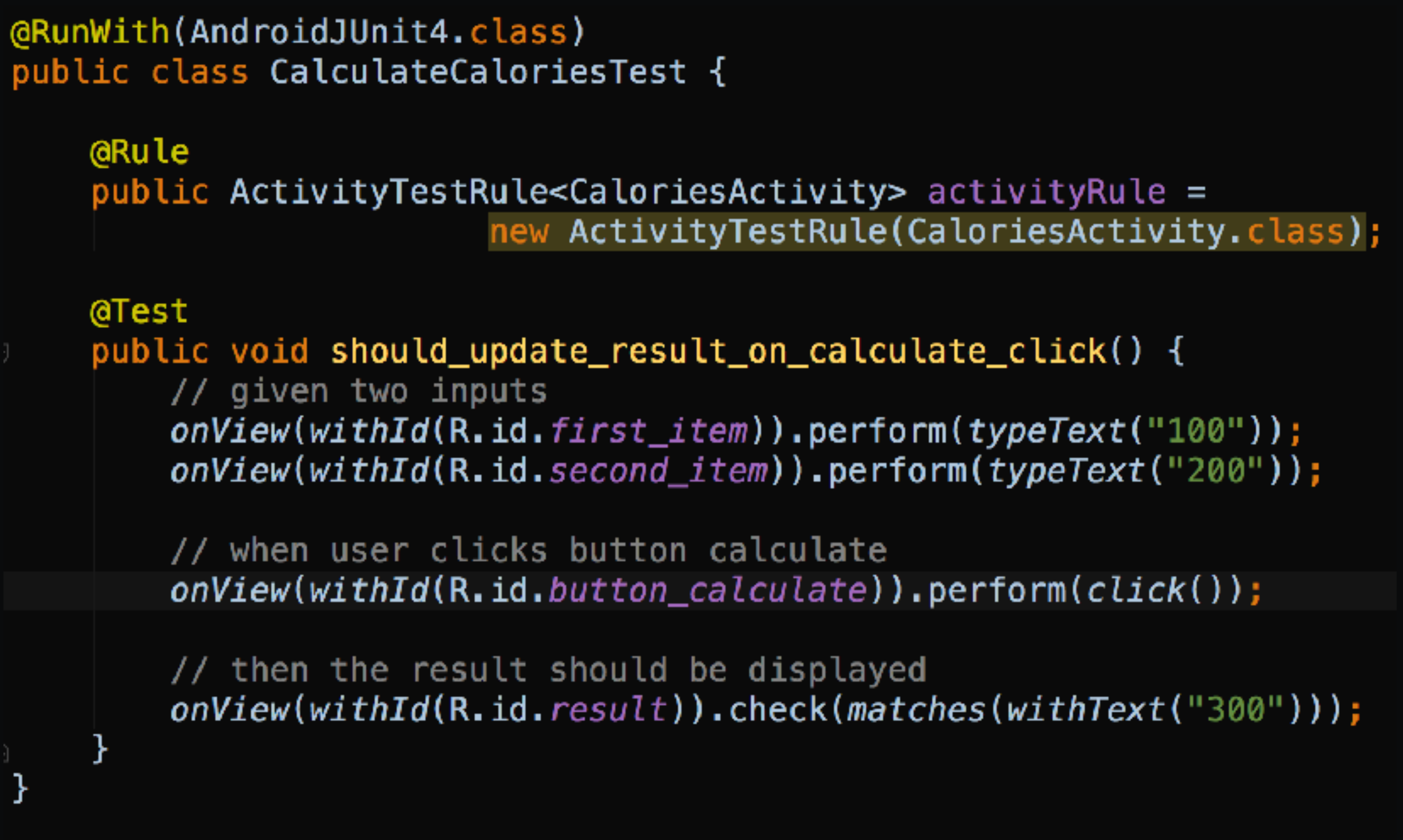
Android Testing Support Library - a set of APIs for testing

- Espresso
- AndroidJUnitRunner
- JUnit4 Rules
- UI Automator

https://google.github.io/android-testing-support-library/docs/espresso/

Espresso is a framework used to write UI Android UI tests and provides an expressive API to traverse and validate UI hierarchy.

Espresso test



```java
@RunWith(AndroidJUnit4.class)
public class CalculateCaloriesTest {

    @Rule
    public ActivityTestRule<CaloriesActivity> activityRule =
                                new ActivityTestRule(CaloriesActivity.class);

    @Test
    public void should_update_result_on_calculate_click() {
        // given two inputs
        onView(withId(R.id.first_item)).perform(typeText("100"));
        onView(withId(R.id.second_item)).perform(typeText("200"));

        // when user clicks button calculate
        onView(withId(R.id.button_calculate)).perform(click());

        // then the result should be displayed
        onView(withId(R.id.result)).check(matches(withText("300")));
    }
}
```

AndroidJUnitRunner

JUnit test runner that enables us to run JUnit tests on Android device - for example tests written with Espresso or UI Automator.

JUnit4 Rules

Set of JUnit rules to reduce boilerplate code:

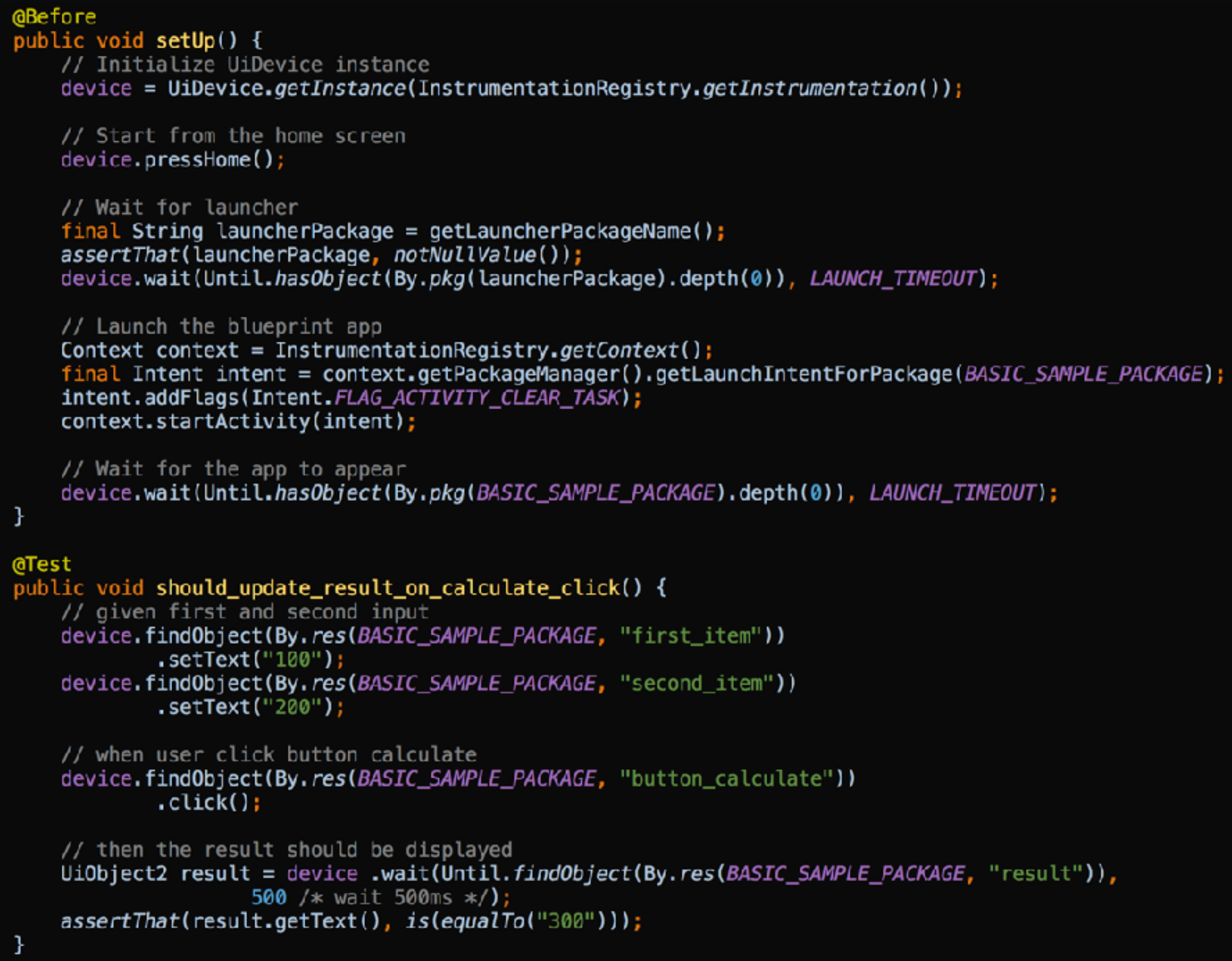
- ActivityTestRule
- ServiceTestRule
- IntentsTestRule

UI Automator

UI testing framework suitable for cross-app functional UI testing across system and installed apps

UI Automator test

JavaCro'17



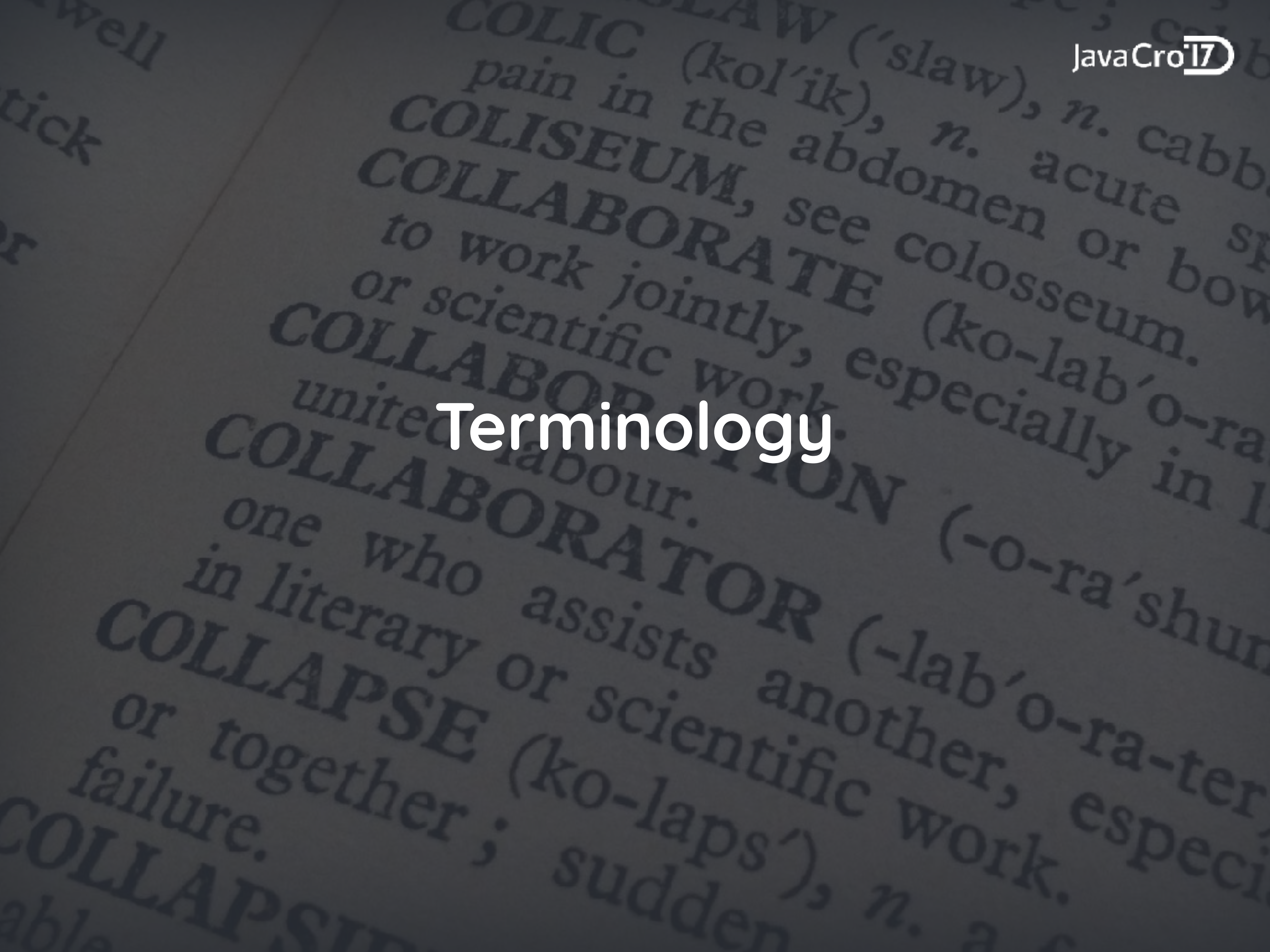
```java
@Before
public void setUp() {
    // Initialize UiDevice instance
    device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());

    // Start from the home screen
    device.pressHome();

    // Wait for launcher
    final String launcherPackage = getLauncherPackageName();
    assertThat(launcherPackage, notNullValue());
    device.wait(Until.hasObject(By.pkg(launcherPackage).depth(0)), LAUNCH_TIMEOUT);

    // Launch the blueprint app
    Context context = InstrumentationRegistry.getContext();
    final Intent intent = context.getPackageManager().getLaunchIntentForPackage(BASIC_SAMPLE_PACKAGE);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
    context.startActivity(intent);

    // Wait for the app to appear
    device.wait(Until.hasObject(By.pkg(BASIC_SAMPLE_PACKAGE).depth(0)), LAUNCH_TIMEOUT);
}

@Test
public void should_update_result_on_calculate_click() {
    // given first and second input
    device.findObject(By.res(BASIC_SAMPLE_PACKAGE, "first_item"))
            .setText("100");
    device.findObject(By.res(BASIC_SAMPLE_PACKAGE, "second_item"))
            .setText("200");

    // when user click button calculate
    device.findObject(By.res(BASIC_SAMPLE_PACKAGE, "button_calculate"))
            .click();

    // then the result should be displayed
    UiObject2 result = device .wait(Until.findObject(By.res(BASIC_SAMPLE_PACKAGE, "result")),
                    500 /* wait 500ms */);
    assertThat(result.getText(), is(equalTo("300")));
}
```

Tools overview

- **JUnit** - fast, no framework
- **Espresso** - slow, with framework, white-box
- **UI Automator** - slow, with framework, black-box

Which tests to write?

Terminology



well
tick
r

SLAW ('slaw), n. cabb
COLIC (kol'ik), n. acute sp
pain in the abdomen or bow
COLISEUM, see colosseum.
COLLABORATE (ko-lab'o-ra
to work jointly, especially in l
or scientific work.
COLLABORATION (-o-ra'shun
united labour.
COLLABORATOR (-lab'o-ra-ter
one who assists another, especi
in literary or scientific work.
COLLAPSE (ko-laps'), n. a f
or together; sudden
failure.
COLLAPSI
able

Types of tests

unit tests; integration tests; component tests; instrumented tests; UI tests; end-to-end tests; acceptance tests; functional tests; performance tests; endurance tests; manual tests; parameterised tests; smoke tests; usability tests;

Types of tests

unit tests; integration tests; component tests; instrumented tests; UI tests; end-to-end tests; acceptance tests; functional tests; performance tests; endurance tests; manual tests; parameterised tests; smoke tests; usability tests;

Unit test

Unit - smallest testable part of software.

Unit test - automated piece of code that invokes a unit of work in the system and determines whether it behaves exactly as expected.

Integration test

automated code that tests individual units combined in a group. The purpose of this level of testing is to expose faults in the interaction between integrated units

Functional test

tests the features specified in functional requirements specification vertically (as a whole)

End-to-end test

test that validates the system as a whole is working as expected (including the app, back-end, third-party code)

Instrumented tests

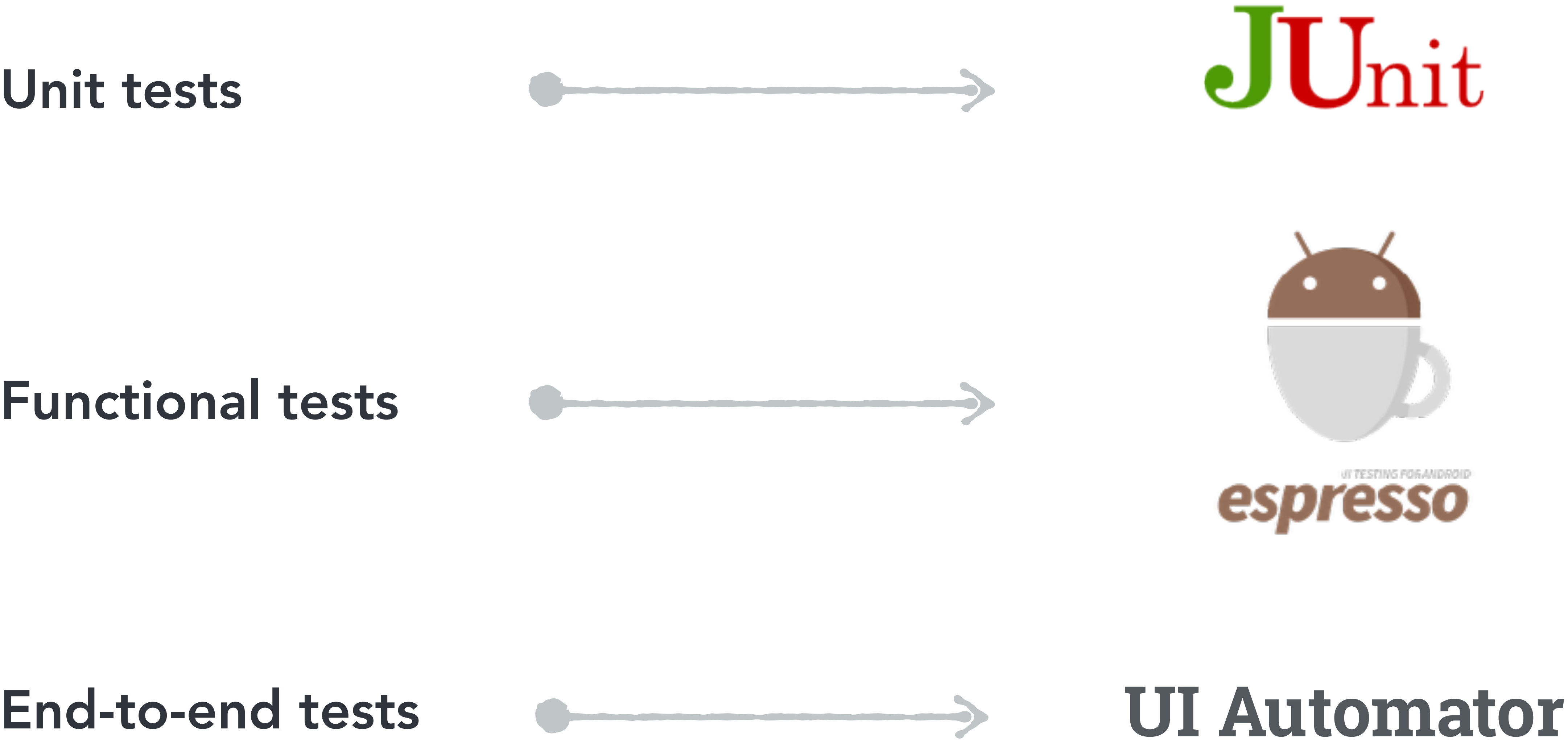
test that require a device or an emulator

Manual tests

test performed manually by a human

The Approach

Test - Tool mapping



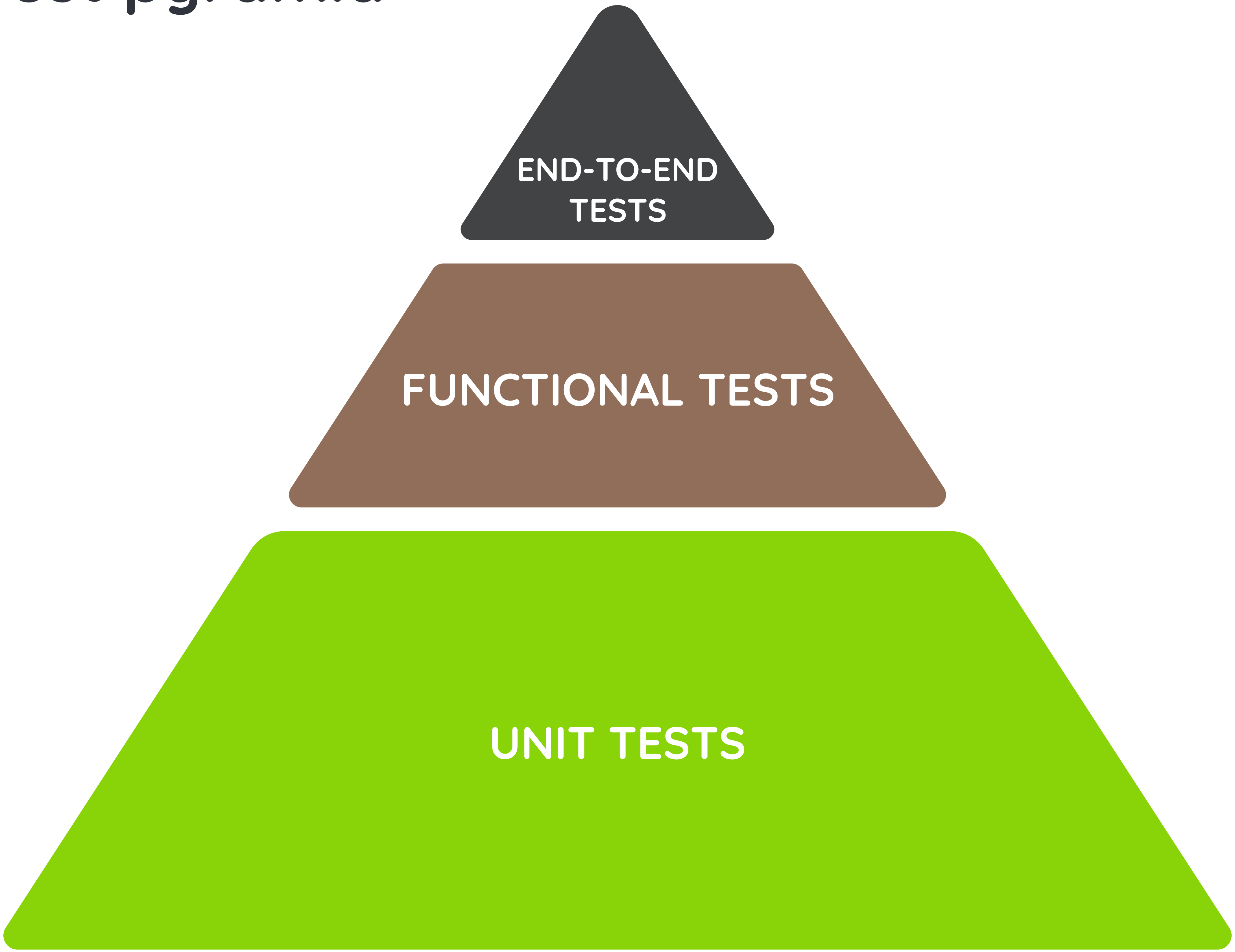
Unit tests → JUnit

Functional tests → espresso

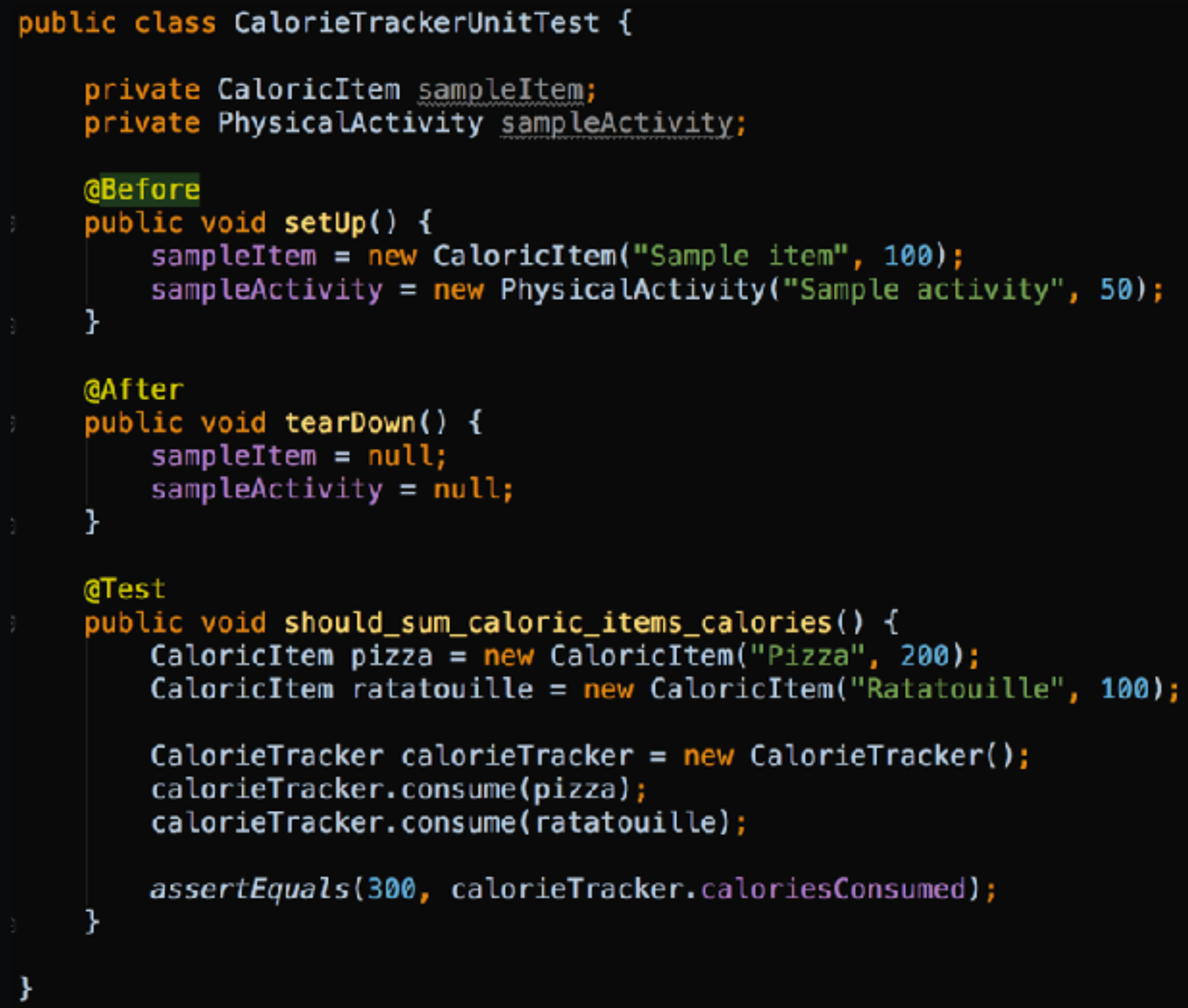
End-to-end tests → UI Automator

JavaCro'17

Test pyramid

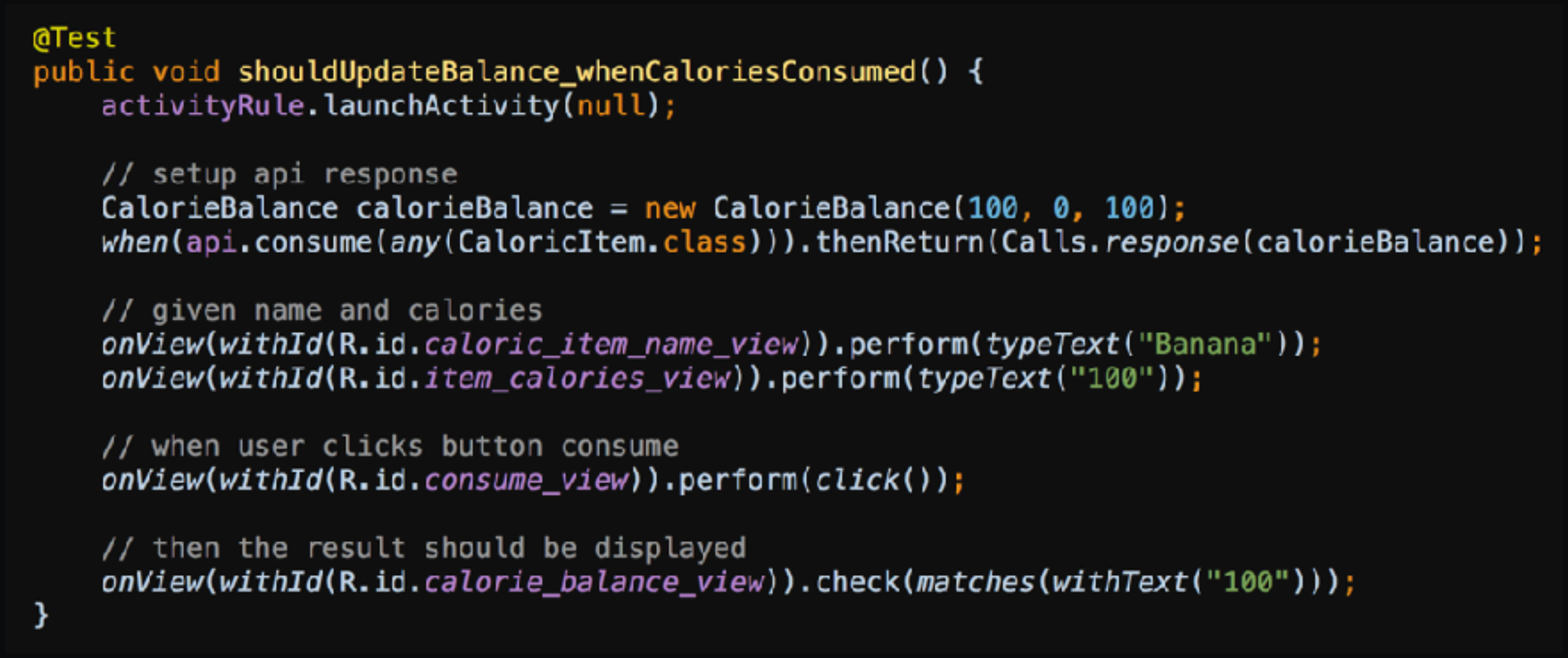


Unit test in practice



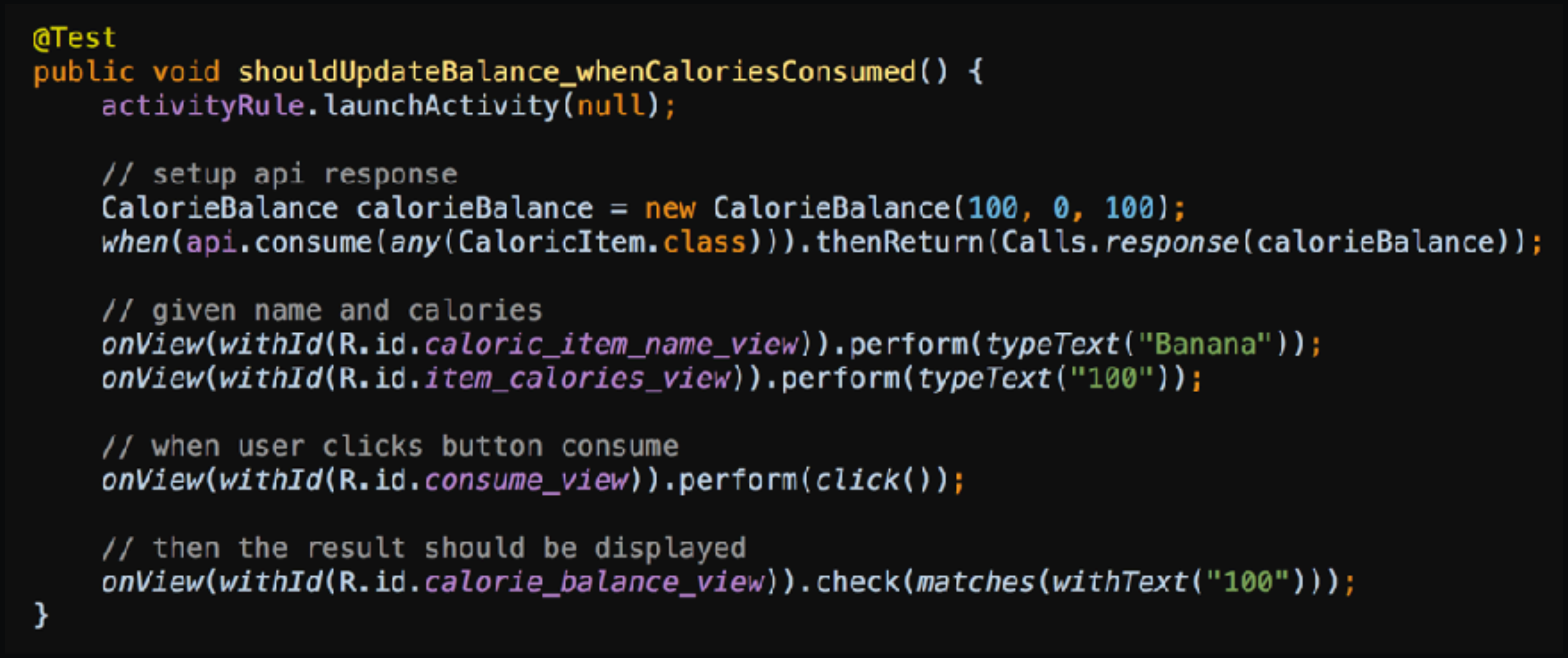
```
public class CalorieTrackerUnitTest {

    private CaloricItem sampleItem;
    private PhysicalActivity sampleActivity;

    @Before
    public void setUp() {
        sampleItem = new CaloricItem("Sample item", 100);
        sampleActivity = new PhysicalActivity("Sample activity", 50);
    }

    @After
    public void tearDown() {
        sampleItem = null;
        sampleActivity = null;
    }

    @Test
    public void should_sum_caloric_items_calories() {
        CaloricItem pizza = new CaloricItem("Pizza", 200);
        CaloricItem ratatouille = new CaloricItem("Ratatouille", 100);

        CalorieTracker calorieTracker = new CalorieTracker();
        calorieTracker.consume(pizza);
        calorieTracker.consume(ratatouille);

        assertEquals(300, calorieTracker.caloriesConsumed);
    }

}
```

Functional test in practice



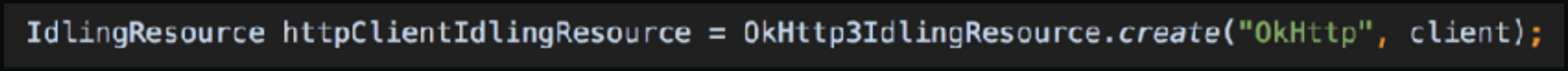
```
@Test
public void shouldUpdateBalance_whenCaloriesConsumed() {
    activityRule.launchActivity(null);

    // setup api response
    CalorieBalance calorieBalance = new CalorieBalance(100, 0, 100);
    when(api.consume(any(CaloricItem.class))).thenReturn(Calls.response(calorieBalance));

    // given name and calories
    onView(withId(R.id.caloric_item_name_view)).perform(typeText("Banana"));
    onView(withId(R.id.item_calories_view)).perform(typeText("100"));

    // when user clicks button consume
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matches(withText("100")));
}
```

Functional test in practice

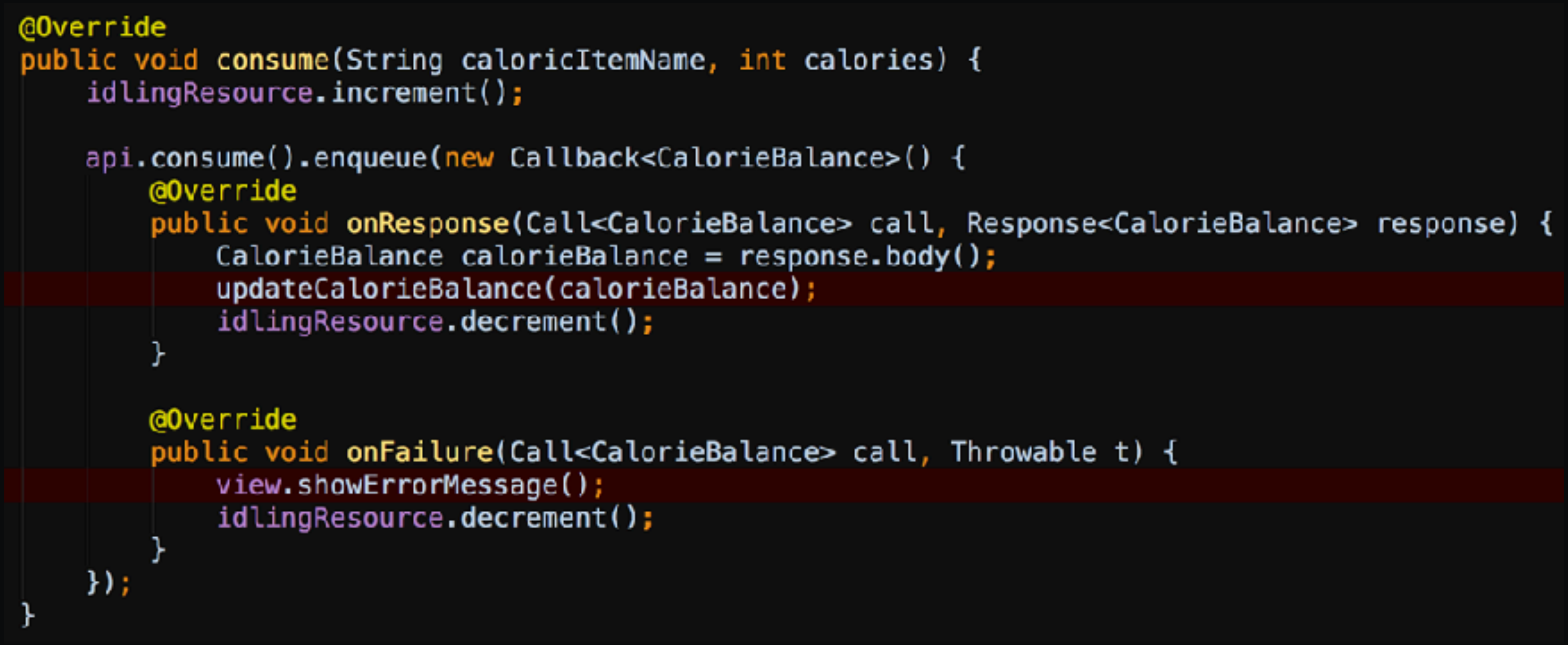


```java
@Test
public void shouldUpdateBalance_whenCaloriesConsumed() {
    activityRule.launchActivity(null);

    // setup api response
    CalorieBalance calorieBalance = new CalorieBalance(100, 0, 100);
    when(api.consume(any(CaloricItem.class))).thenReturn(Calls.response(calorieBalance));

    // given name and calories
    onView(withId(R.id.caloric_item_name_view)).perform(typeText("Banana"));
    onView(withId(R.id.item_calories_view)).perform(typeText("100"));

    // when user clicks button consume
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matches(withText("100")));
}
```

Espresso Idling Resource

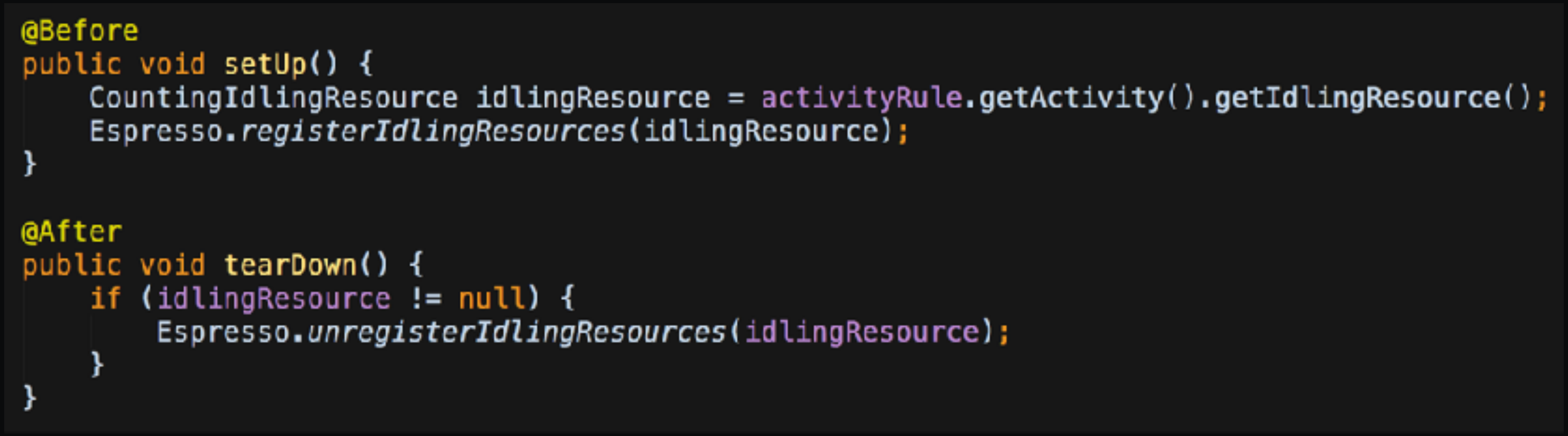


```java
IdlingResource httpClientIdlingResource = OkHttp3IdlingResource.create("OkHttp", client);
```



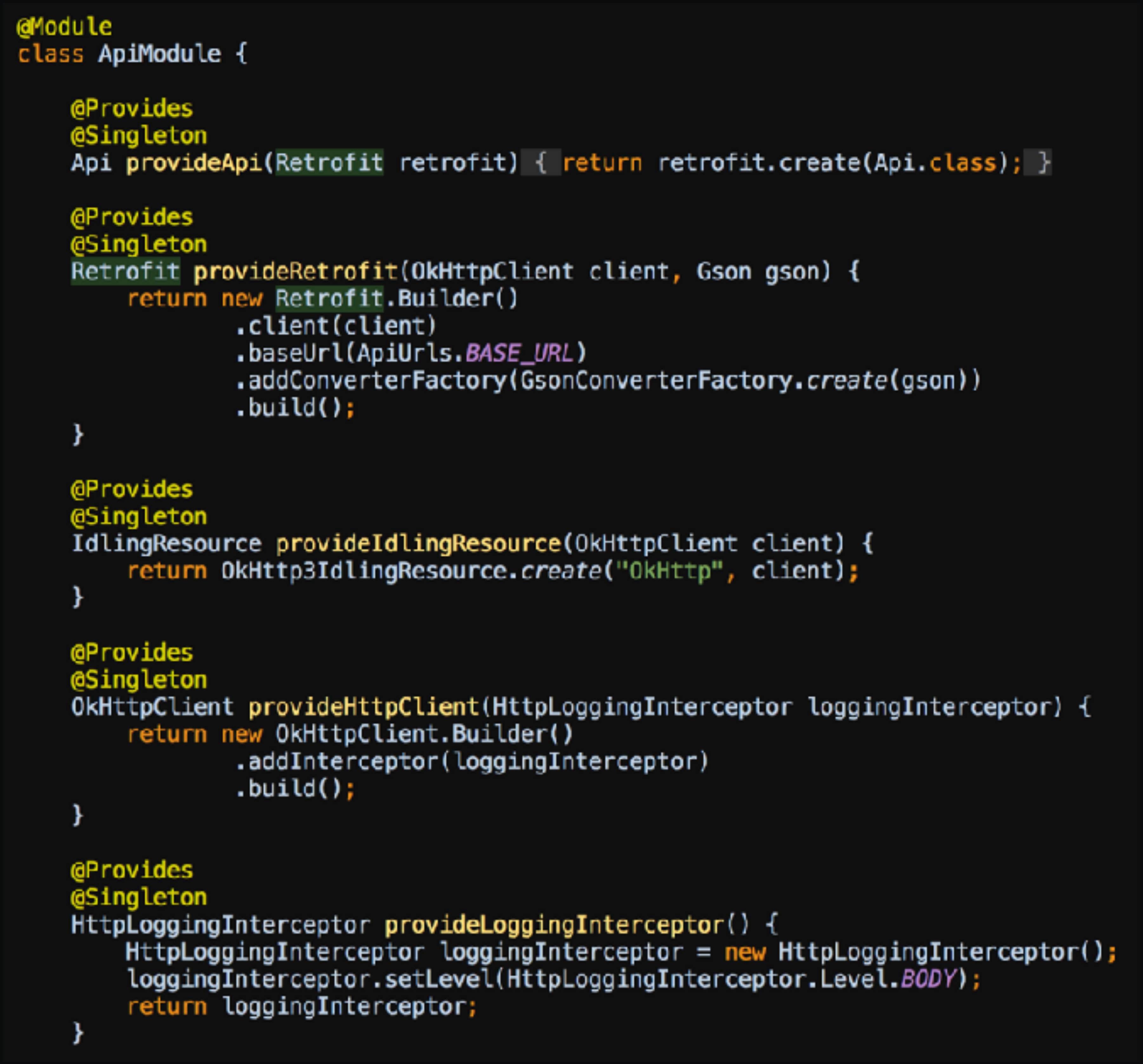
```java
@Override
public void consume(String caloricItemName, int calories) {
    idlingResource.increment();

    api.consume().enqueue(new Callback<CalorieBalance>() {
        @Override
        public void onResponse(Call<CalorieBalance> call, Response<CalorieBalance> response) {
            CalorieBalance calorieBalance = response.body();
            updateCalorieBalance(calorieBalance);
            idlingResource.decrement();
        }

        @Override
        public void onFailure(Call<CalorieBalance> call, Throwable t) {
            view.showErrorMessage();
            idlingResource.decrement();
        }
    });
}
```

Espresso Idling Resource



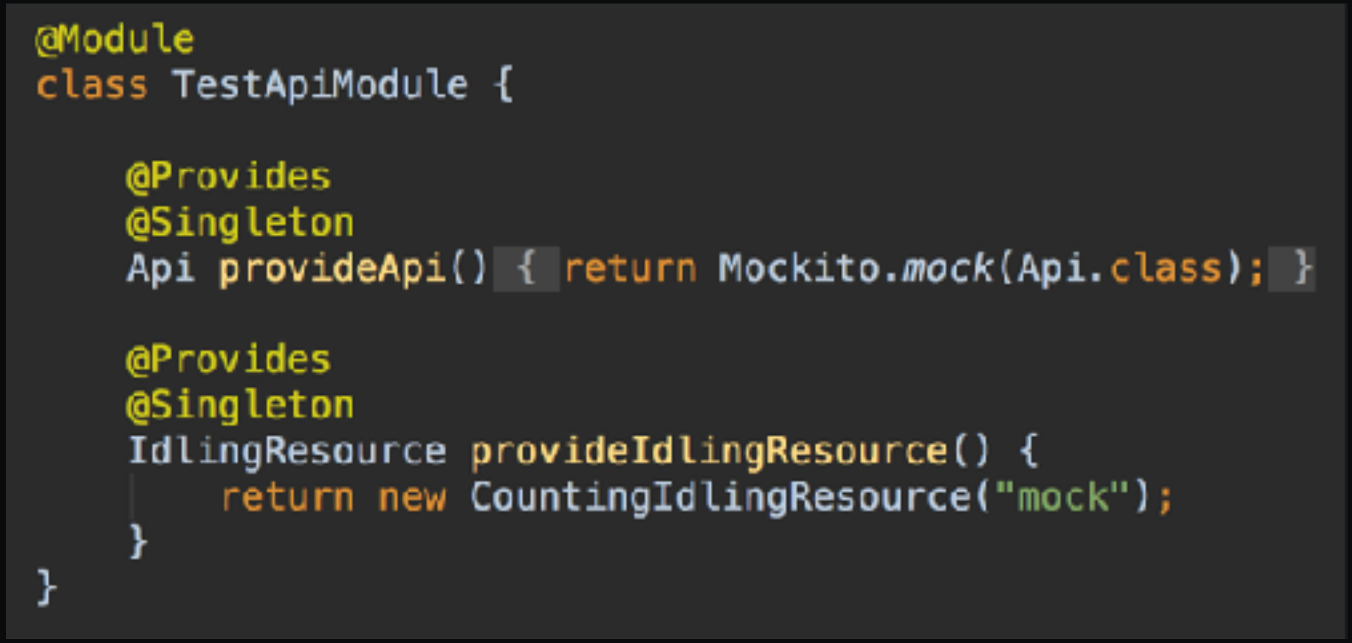
```java
@Before
public void setUp() {
    CountingIdlingResource idlingResource = activityRule.getActivity().getIdlingResource();
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResource);
    }
}
```

DI - Dagger 2 Setup

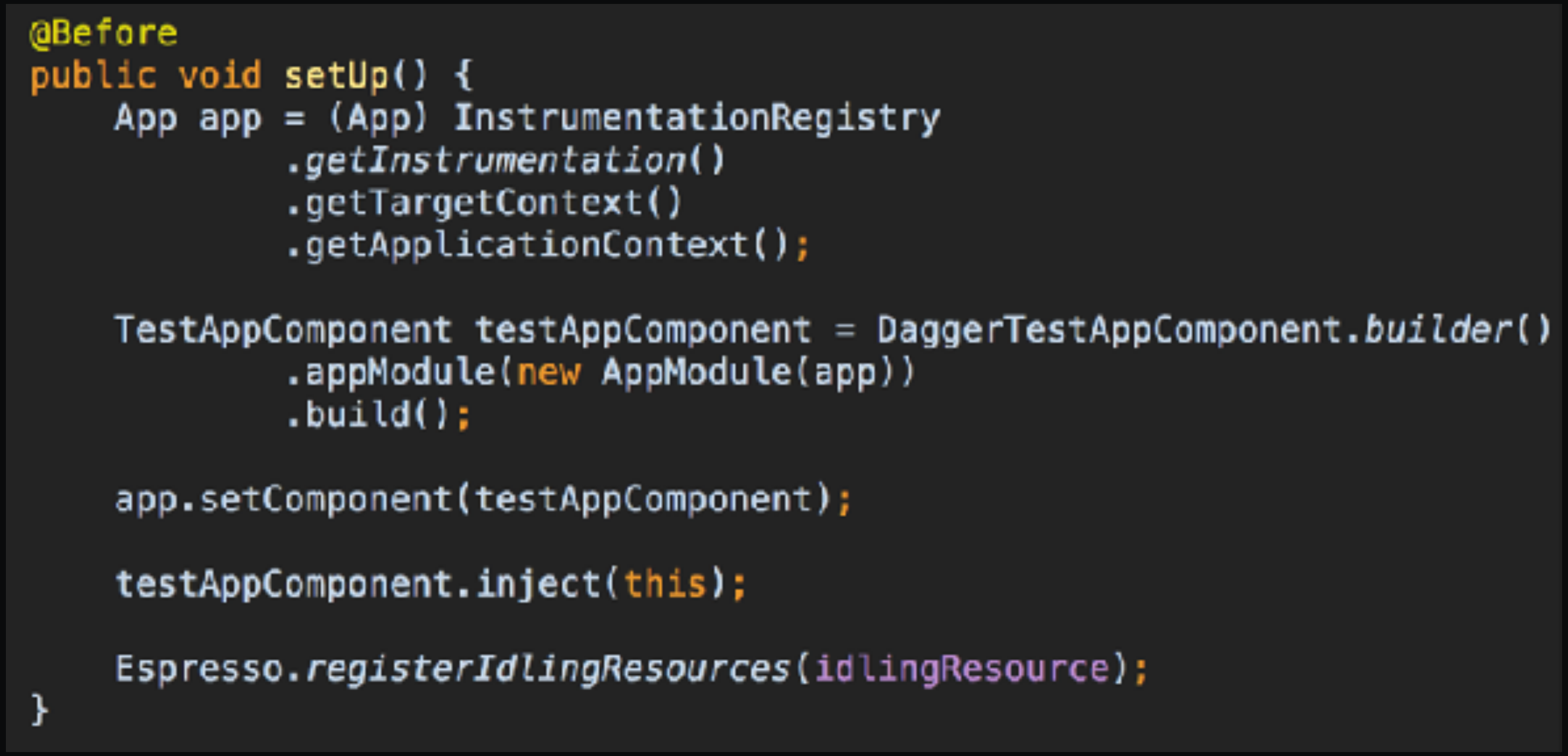


```
@Module
class ApiModule {

    @Provides
    @Singleton
    Api provideApi(Retrofit retrofit) { return retrofit.create(Api.class); }

    @Provides
    @Singleton
    Retrofit provideRetrofit(OkHttpClient client, Gson gson) {
        return new Retrofit.Builder()
                .client(client)
                .baseUrl(ApiUrls.BASE_URL)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();
    }

    @Provides
    @Singleton
    IdlingResource provideIdlingResource(OkHttpClient client) {
        return OkHttp3IdlingResource.create("OkHttp", client);
    }

    @Provides
    @Singleton
    OkHttpClient provideHttpClient(HttpLoggingInterceptor loggingInterceptor) {
        return new OkHttpClient.Builder()
                .addInterceptor(loggingInterceptor)
                .build();
    }

    @Provides
    @Singleton
    HttpLoggingInterceptor provideLoggingInterceptor() {
        HttpLoggingInterceptor loggingInterceptor = new HttpLoggingInterceptor();
        loggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        return loggingInterceptor;
    }
```

DI - Dagger 2 Setup

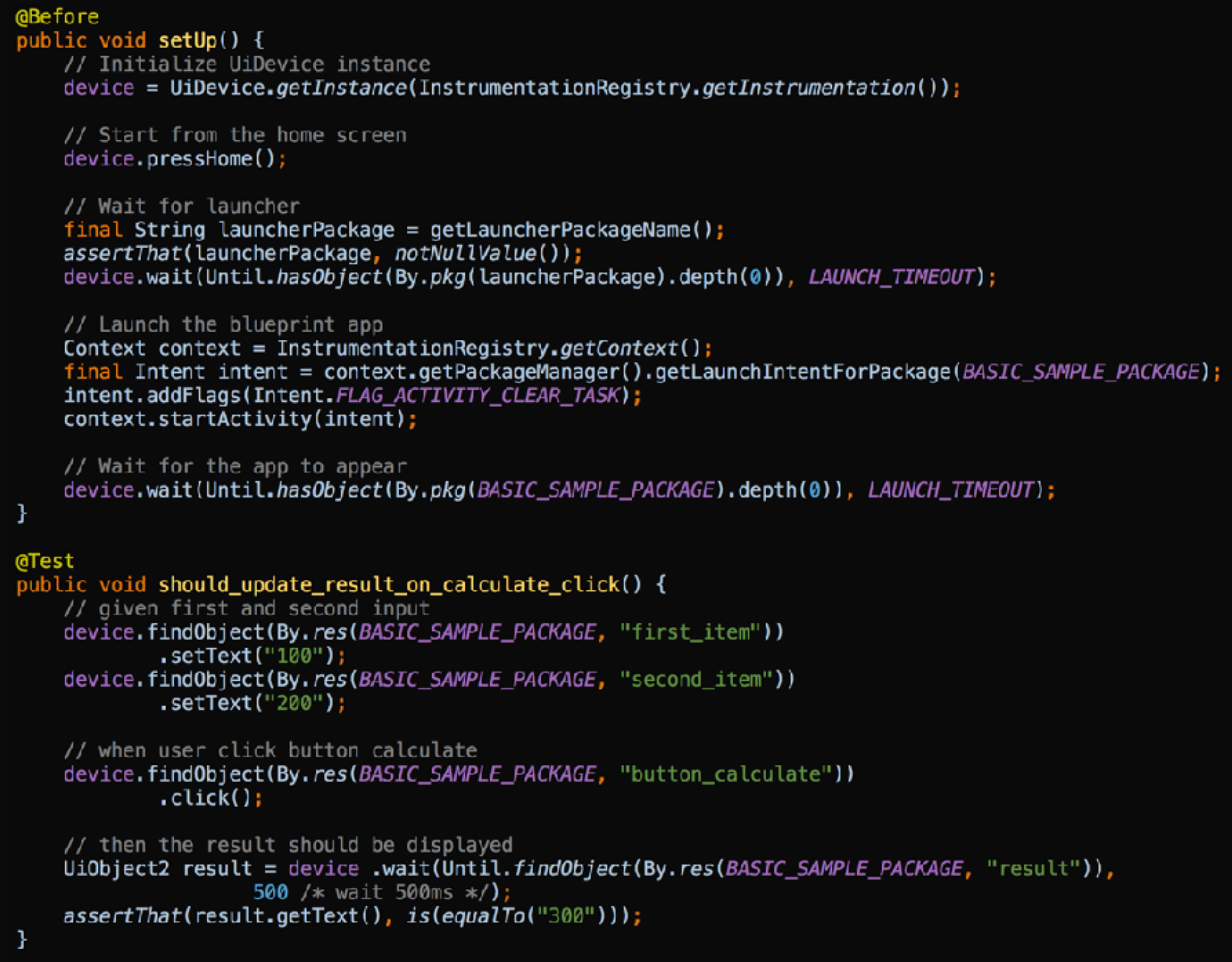


```
@Module
class TestApiModule {

    @Provides
    @Singleton
    Api provideApi() { return Mockito.mock(Api.class); }

    @Provides
    @Singleton
    IdlingResource provideIdlingResource() {
        return new CountingIdlingResource("mock");
    }
}
```

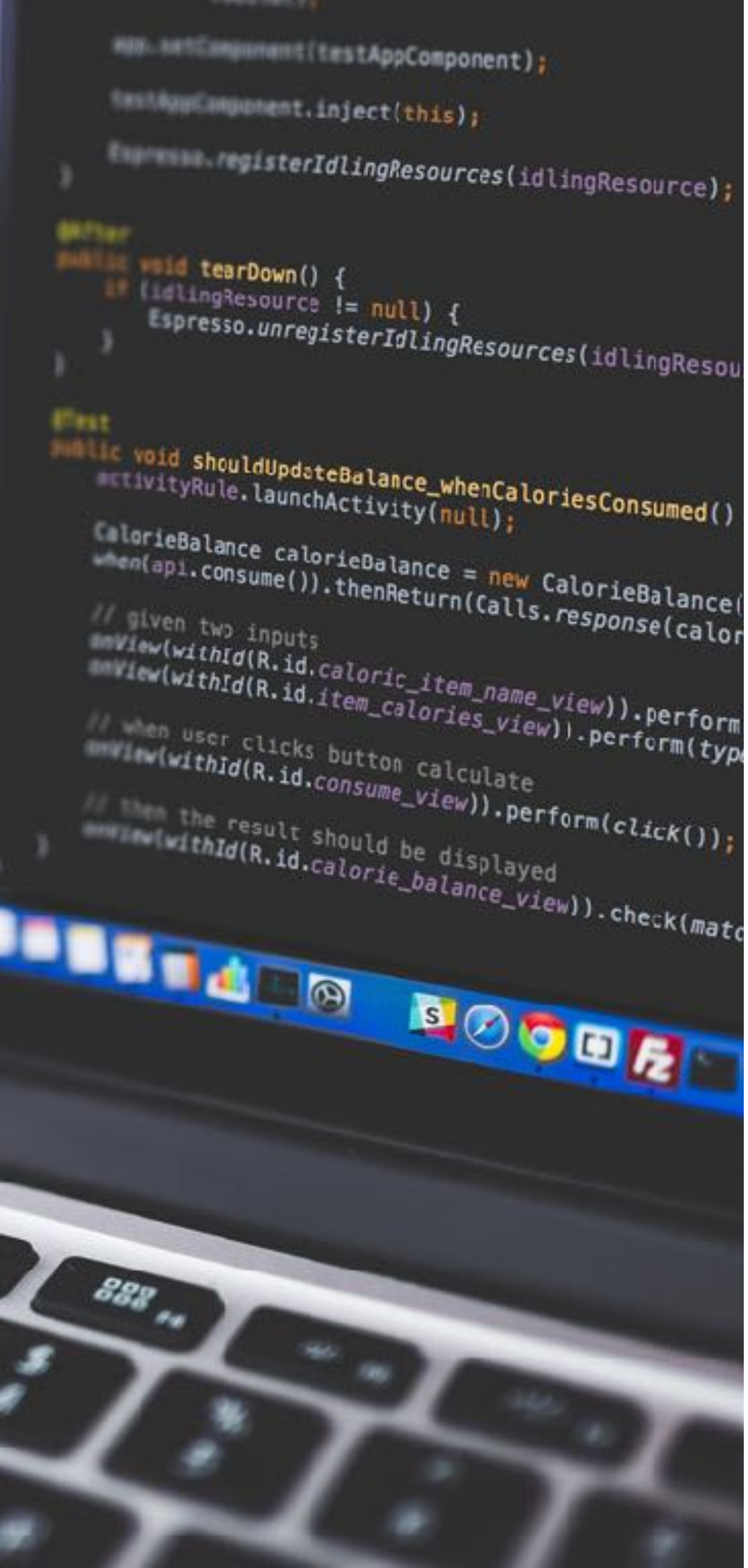


```
@Before
public void setUp() {
    App app = (App) InstrumentationRegistry
            .getInstrumentation()
            .getTargetContext()
            .getApplicationContext();

    TestAppComponent testAppComponent = DaggerTestAppComponent.builder()
            .appModule(new AppModule(app))
            .build();

    app.setComponent(testAppComponent);

    testAppComponent.inject(this);

    Espresso.registerIdlingResources(idlingResource);
}
```

End-to-end test in practice



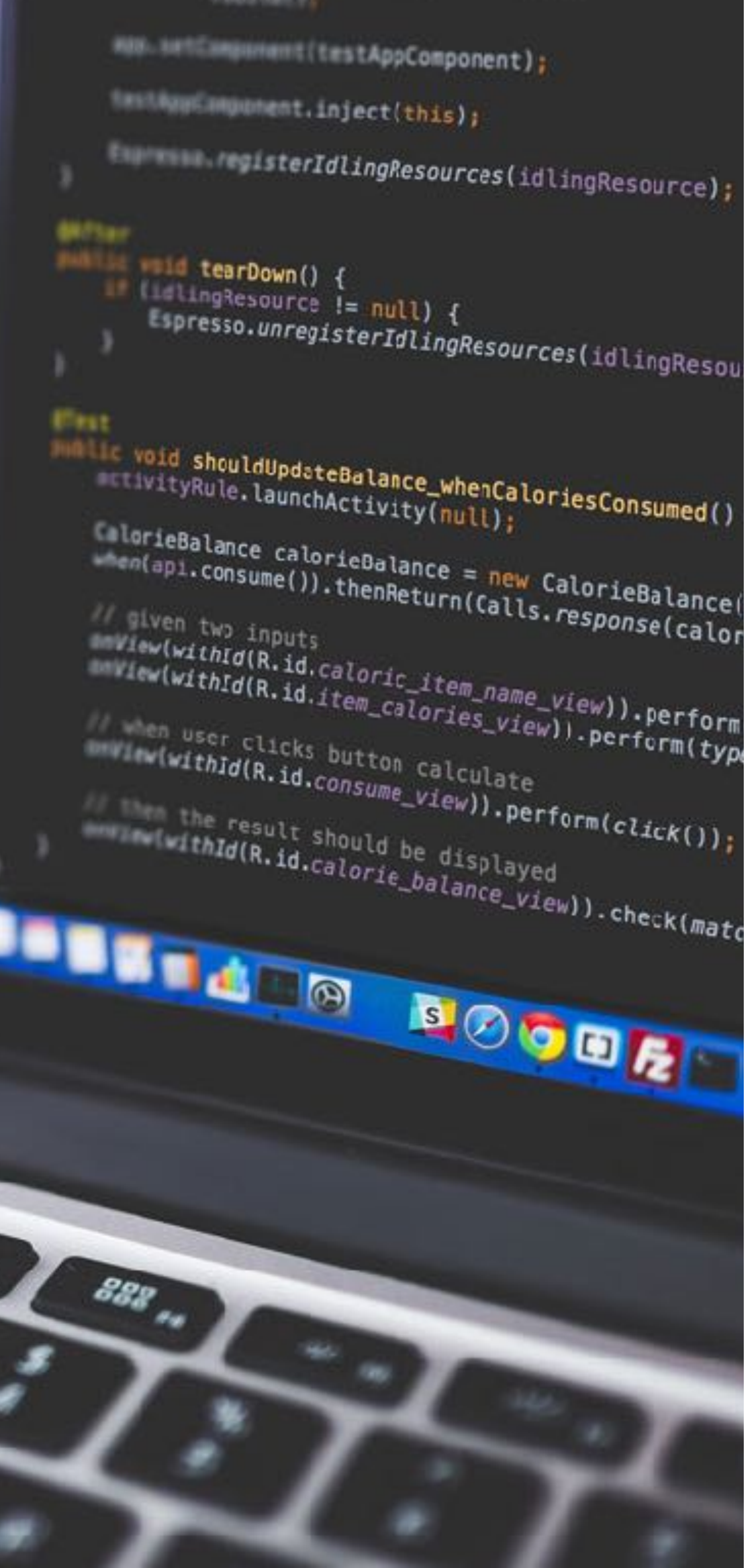
```
@Before
public void setUp() {
    // Initialize UiDevice instance
    device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());

    // Start from the home screen
    device.pressHome();

    // Wait for launcher
    final String launcherPackage = getLauncherPackageName();
    assertThat(launcherPackage, notNullValue());
    device.wait(Until.hasObject(By.pkg(launcherPackage).depth(0)), LAUNCH_TIMEOUT);

    // Launch the blueprint app
    Context context = InstrumentationRegistry.getContext();
    final Intent intent = context.getPackageManager().getLaunchIntentForPackage(BASIC_SAMPLE_PACKAGE);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
    context.startActivity(intent);

    // Wait for the app to appear
    device.wait(Until.hasObject(By.pkg(BASIC_SAMPLE_PACKAGE).depth(0)), LAUNCH_TIMEOUT);
}

@Test
public void should_update_result_on_calculate_click() {
    // given first and second input
    device.findObject(By.res(BASIC_SAMPLE_PACKAGE, "first_item"))
            .setText("100");
    device.findObject(By.res(BASIC_SAMPLE_PACKAGE, "second_item"))
            .setText("200");

    // when user click button calculate
    device.findObject(By.res(BASIC_SAMPLE_PACKAGE, "button_calculate"))
            .click();

    // then the result should be displayed
    UiObject2 result = device .wait(Until.findObject(By.res(BASIC_SAMPLE_PACKAGE, "result")),
                    500 /* wait 500ms */);
    assertThat(result.getText(), is(equalTo("300")));
}
```



```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

JavaCro'17

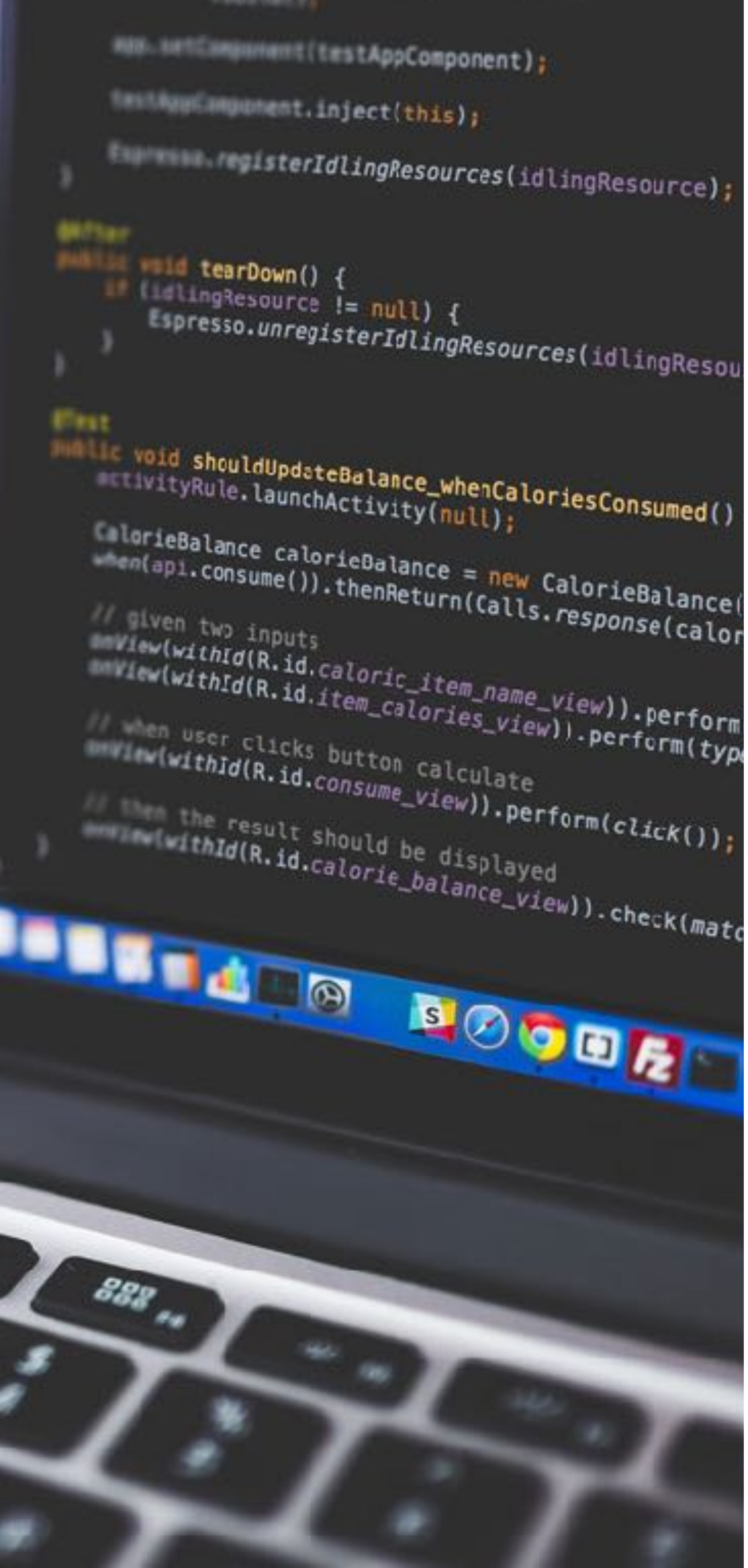
Putting it into practice



```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed() {
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Putting it into practice

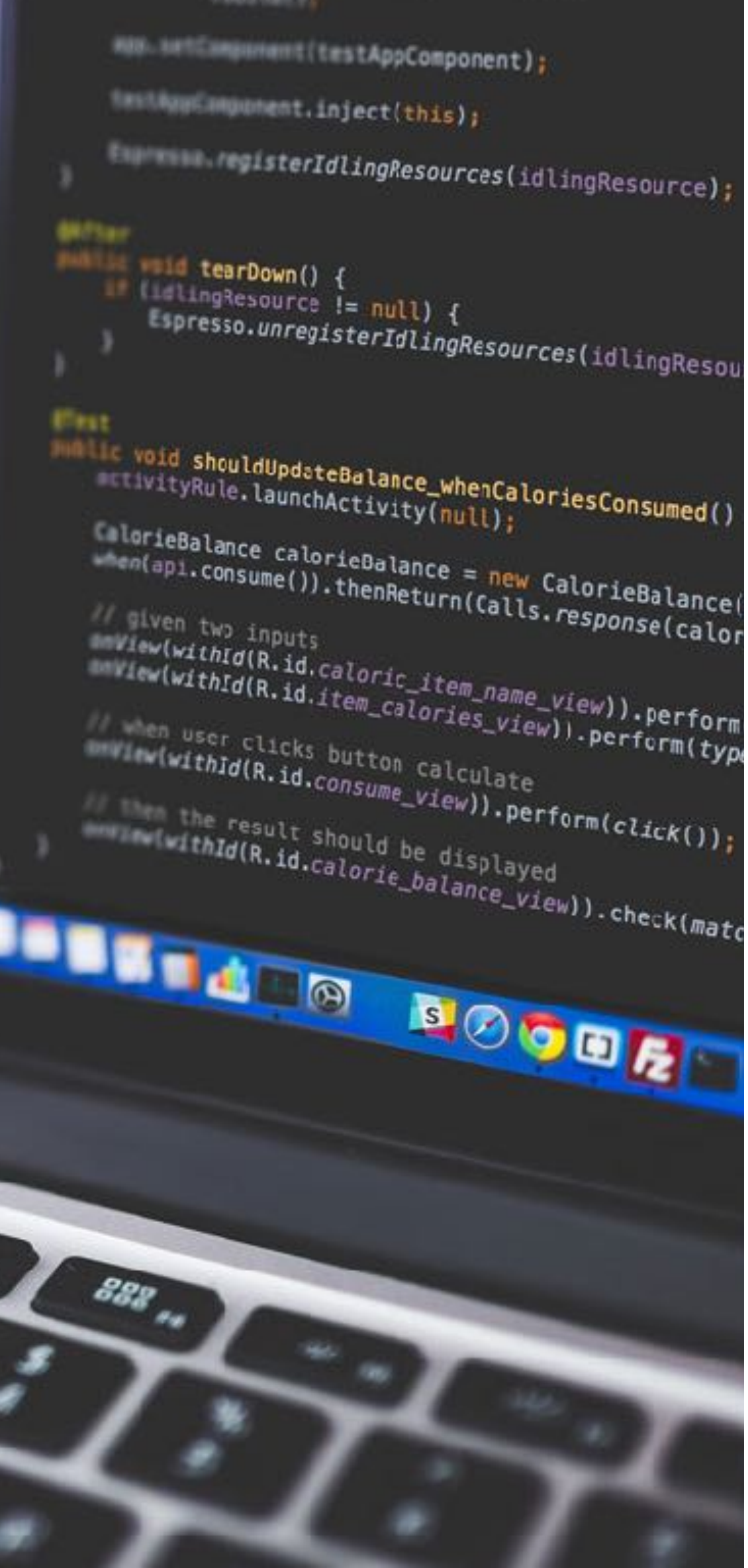
- can be overwhelming



```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed() {
        activityRule.launchActivity(null);
        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor
        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(typ
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

Putting it into practice

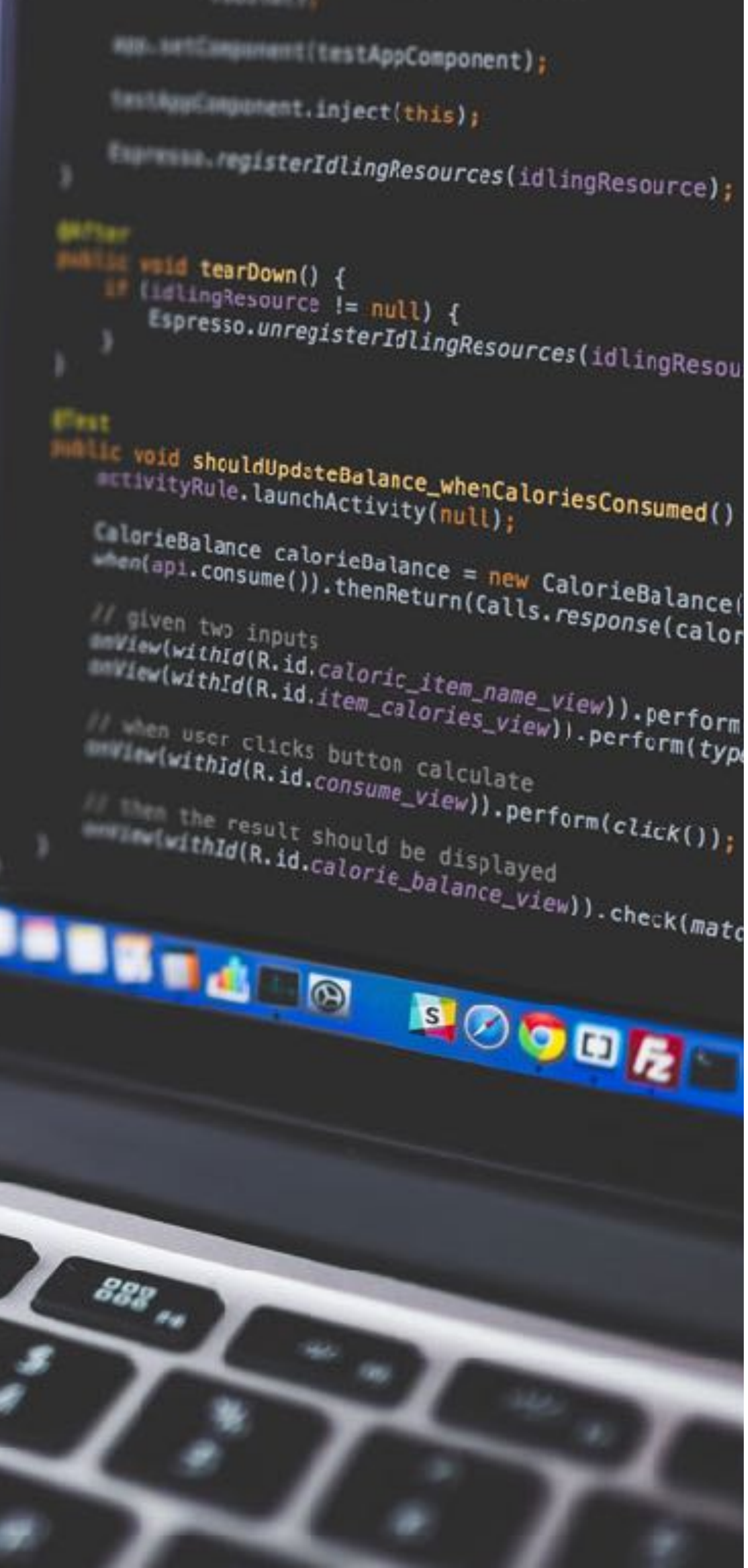
- can be overwhelming
- step-by-step



```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);
        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor
        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(typ
        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());
        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

Putting it into practice

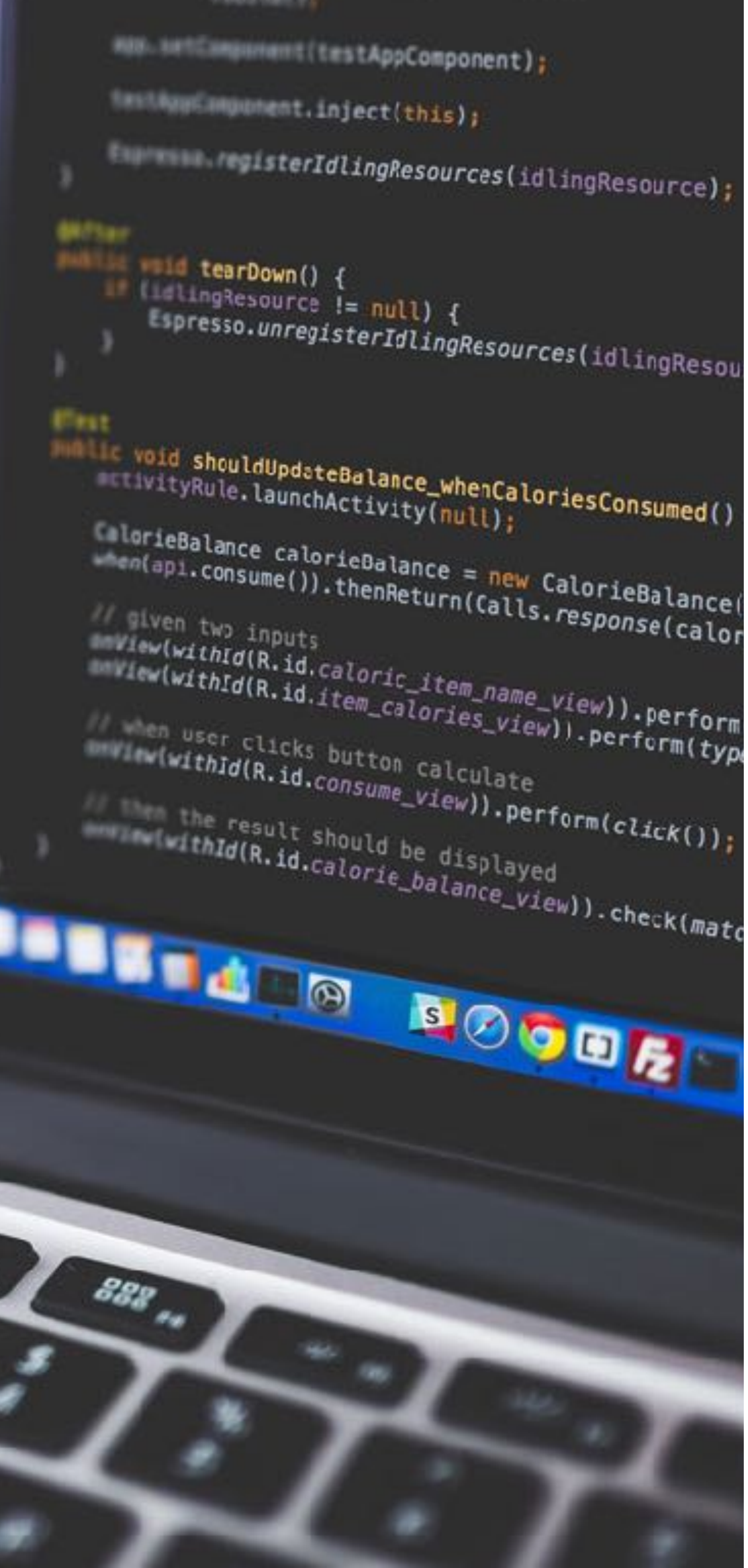
- can be overwhelming
- step-by-step
- unit tests - **immediately**



```
    app.setComponent(testAppComponent);
    testAppComponent.inject(this);
    Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed() {
    activityRule.launchActivity(null);

    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor

    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ

    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());

    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Putting it into practice

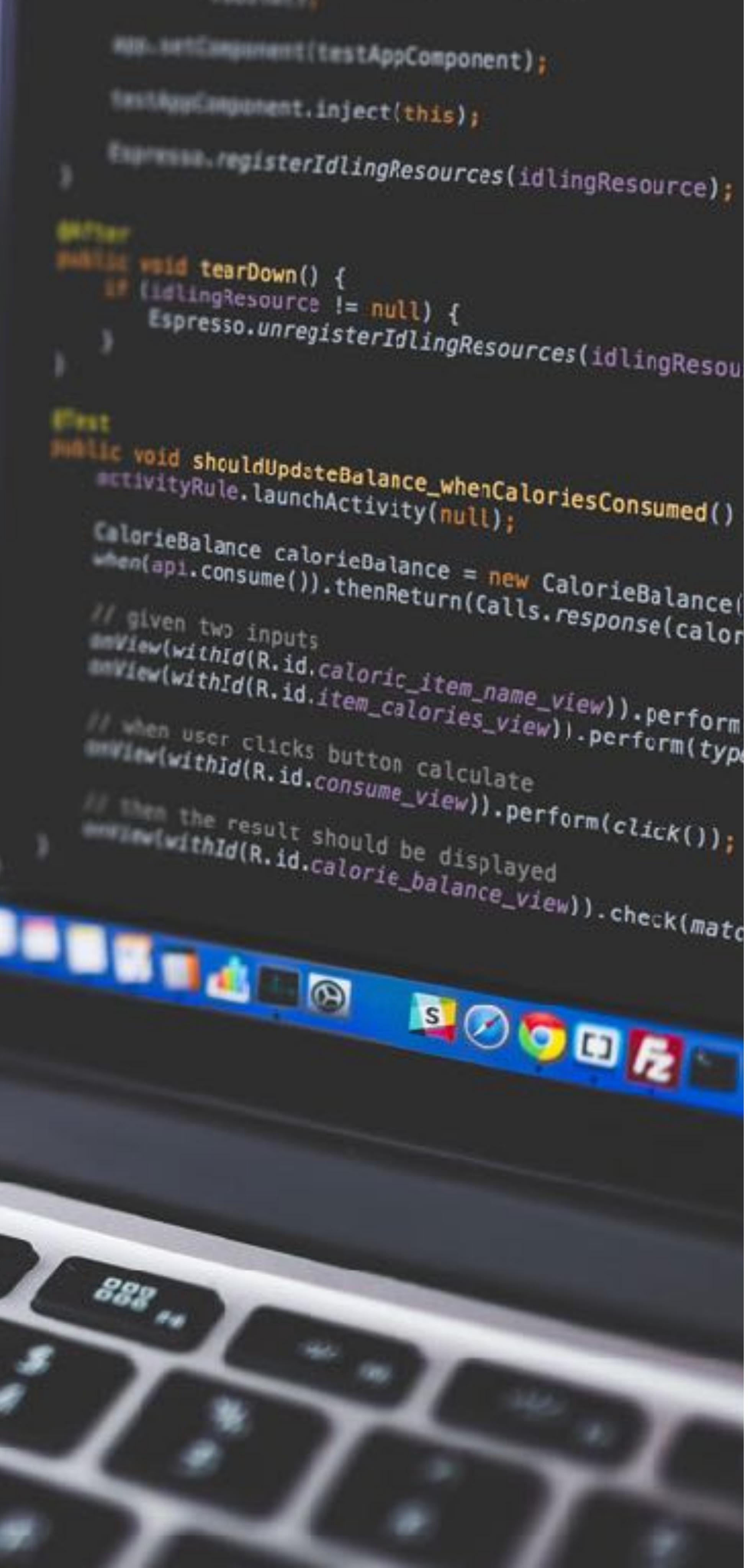
- can be overwhelming
- step-by-step
- unit tests - **immediately**
- functional tests - happy paths and main error path - **short-term**



```
app.setComponent(testAppComponent);
testAppComponent.inject(this);
Espresso.registerIdlingResources(idlingResource);
}

@After
public void tearDown() {
    if (idlingResource != null) {
        Espresso.unregisterIdlingResources(idlingResou
    }
}

@Test
public void shouldUpdateBalance_whenCaloriesConsumed()
    activityRule.launchActivity(null);
    CalorieBalance calorieBalance = new CalorieBalance(
    when(api.consume()).thenReturn(Calls.response(calor
    // given two inputs
    onView(withId(R.id.caloric_item_name_view)).perform
    onView(withId(R.id.item_calories_view)).perform(typ
    // when user clicks button calculate
    onView(withId(R.id.consume_view)).perform(click());
    // then the result should be displayed
    onView(withId(R.id.calorie_balance_view)).check(matc
}
```

Putting it into practice

- can be overwhelming
- step-by-step
- unit tests - **immediately**
- functional tests - happy paths and main error path - **short-term**
- functional tests - all requirement features - **medium-term**



```
        app.setComponent(testAppComponent);
        testAppComponent.inject(this);
        Espresso.registerIdlingResources(idlingResource);
    }

    @After
    public void tearDown() {
        if (idlingResource != null) {
            Espresso.unregisterIdlingResources(idlingResou
        }
    }

    @Test
    public void shouldUpdateBalance_whenCaloriesConsumed()
        activityRule.launchActivity(null);

        CalorieBalance calorieBalance = new CalorieBalance(
        when(api.consume()).thenReturn(Calls.response(calor

        // given two inputs
        onView(withId(R.id.caloric_item_name_view)).perform
        onView(withId(R.id.item_calories_view)).perform(typ

        // when user clicks button calculate
        onView(withId(R.id.consume_view)).perform(click());

        // then the result should be displayed
        onView(withId(R.id.calorie_balance_view)).check(matc
    }
```

JavaCro'17

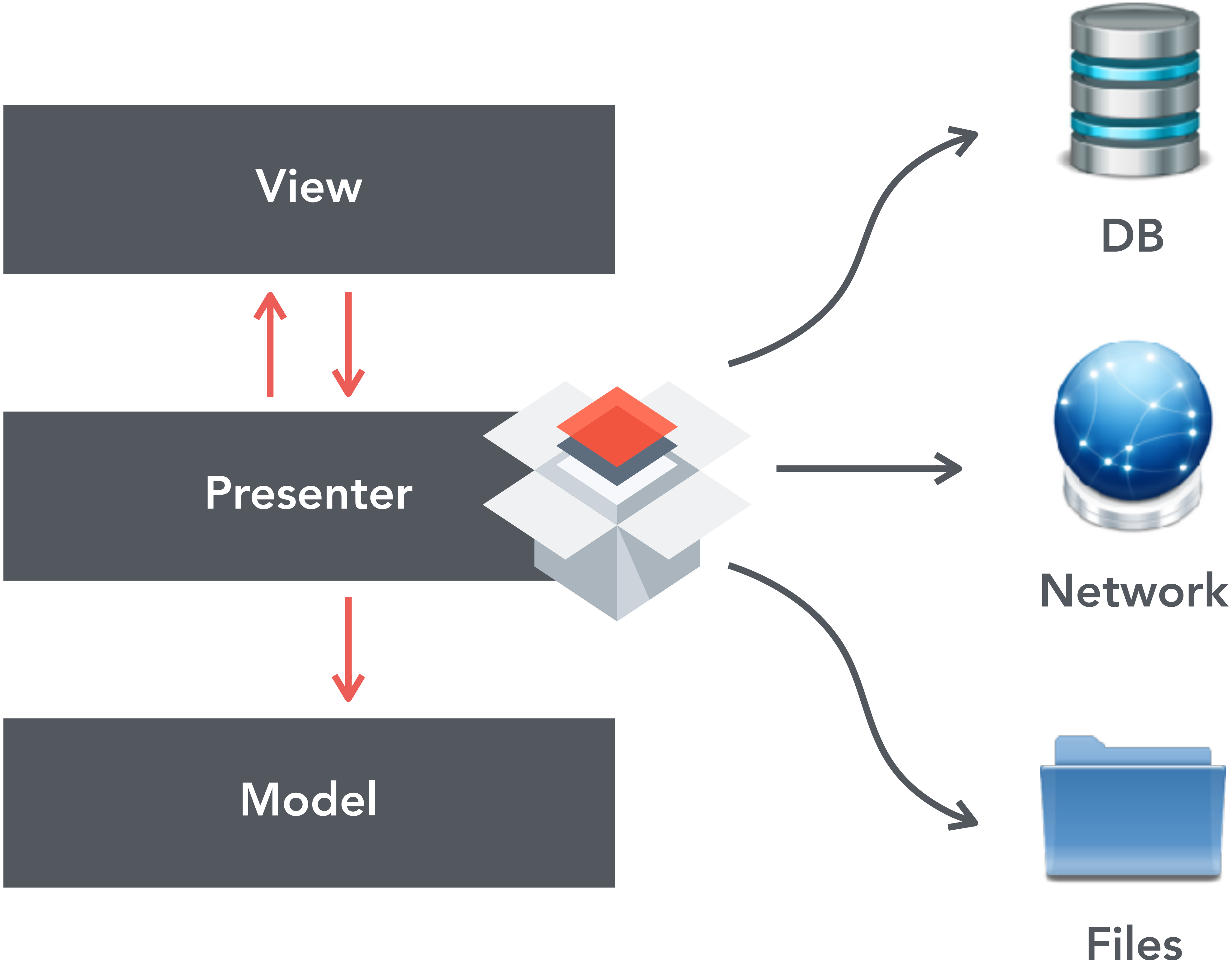
Putting it into practice

- can be overwhelming
- step-by-step
- unit tests - **immediately**
- functional tests - happy paths and main error path - **short-term**
- functional tests - all requirement features - **medium-term**
- end-to-end - **long-term**

Architecture

MVP architecture

MVP architecture



View

Presenter

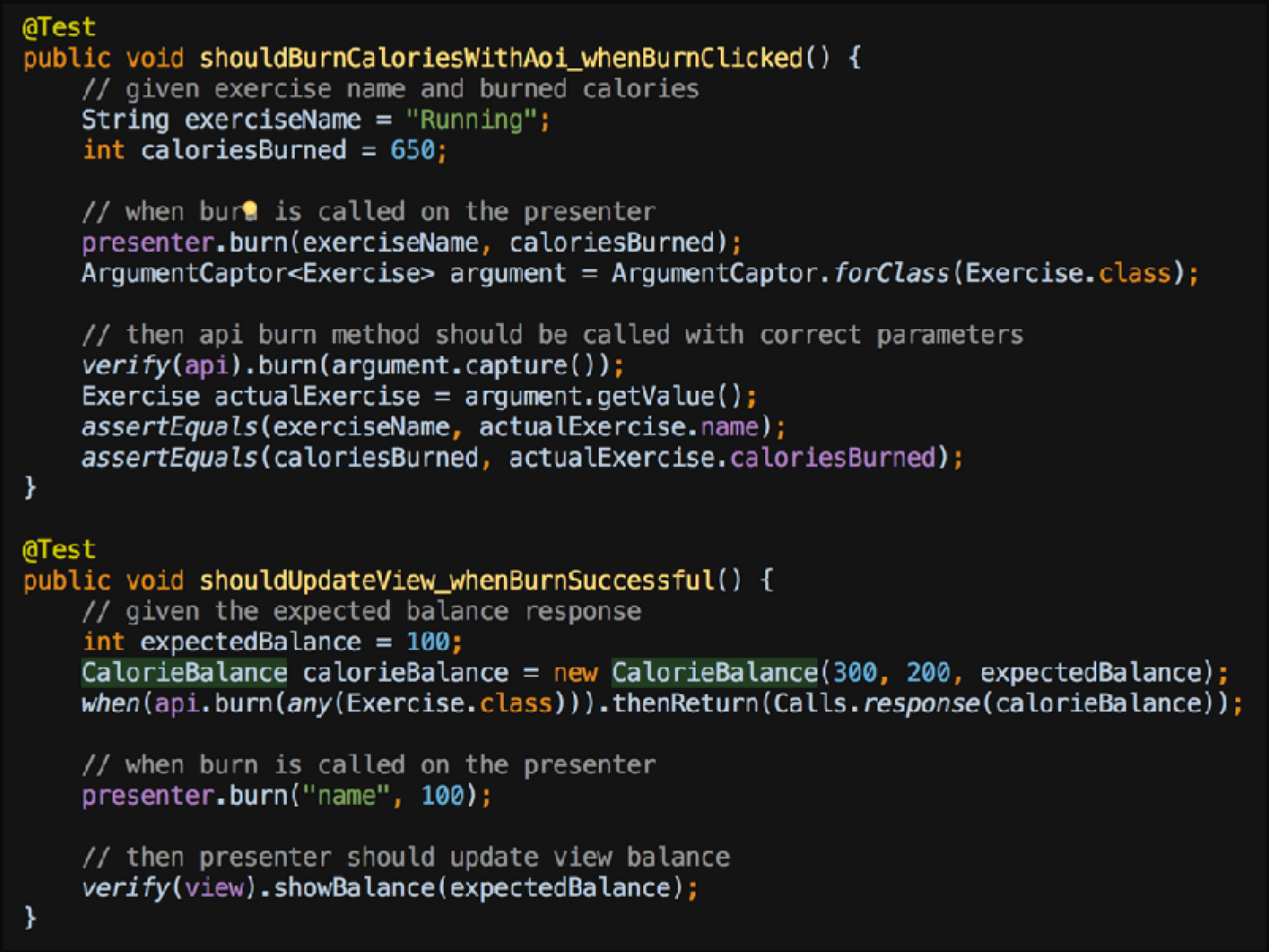
Model

DB

Network

Files

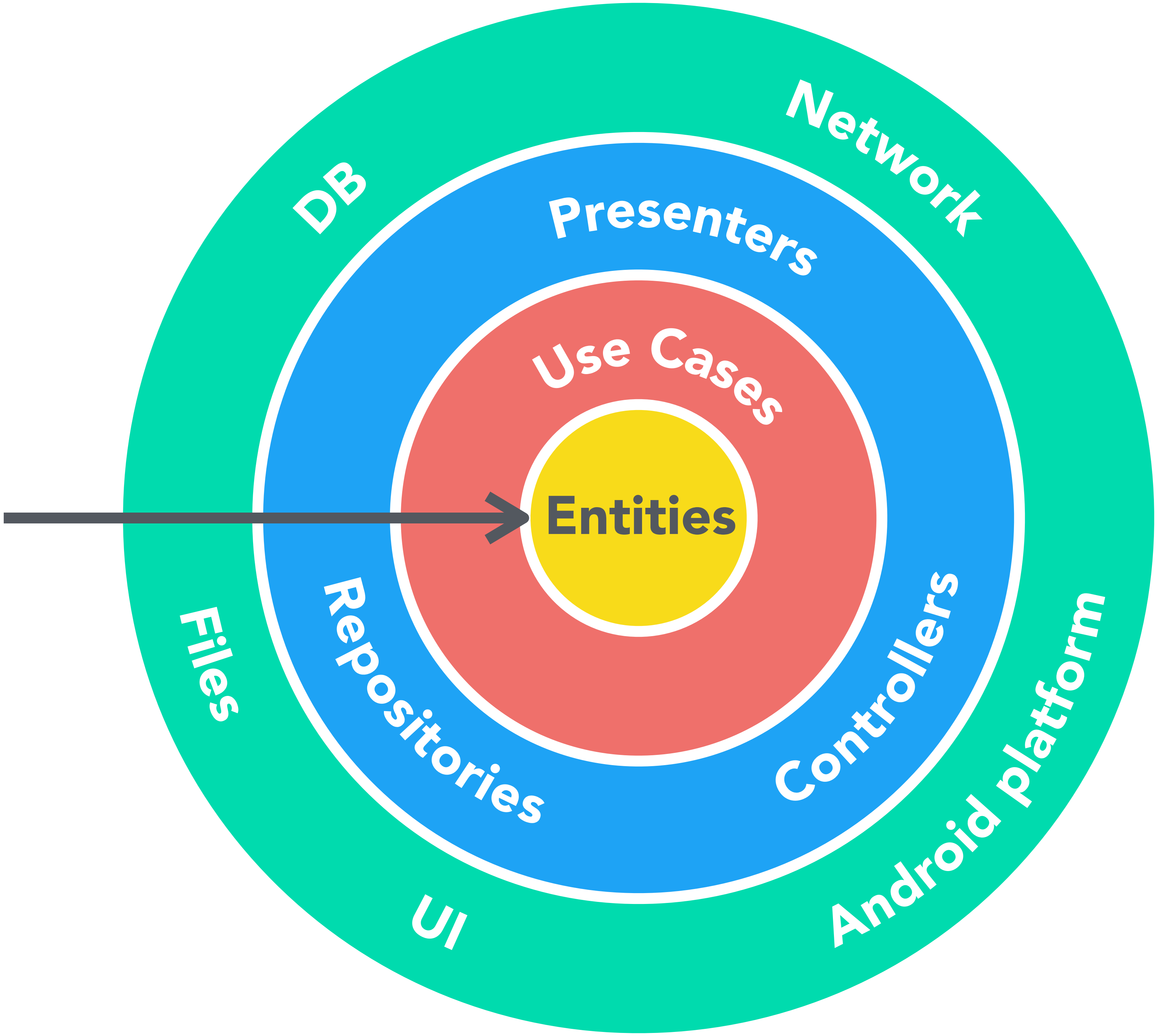
MVP - unit testing presenter interaction



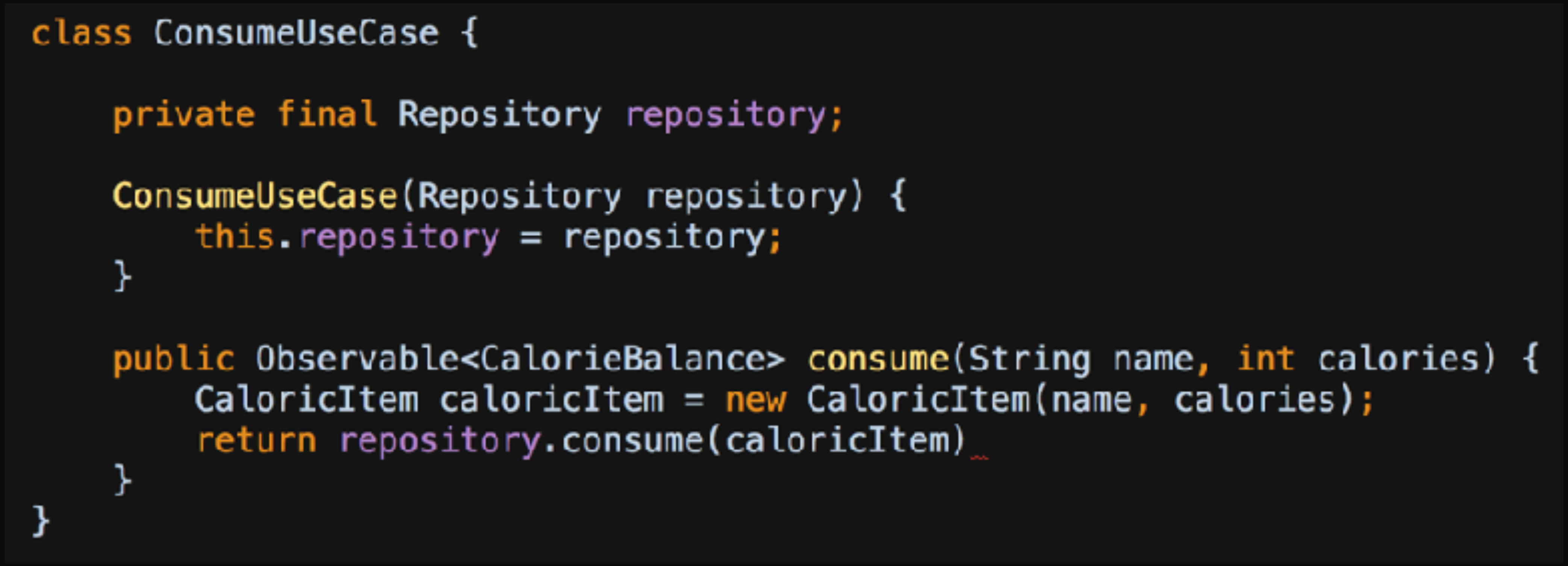
```
@Test
public void shouldBurnCaloriesWithAoi_whenBurnClicked() {
    // given exercise name and burned calories
    String exerciseName = "Running";
    int caloriesBurned = 650;

    // when burn is called on the presenter
    presenter.burn(exerciseName, caloriesBurned);
    ArgumentCaptor<Exercise> argument = ArgumentCaptor.forClass(Exercise.class);

    // then api burn method should be called with correct parameters
    verify(api).burn(argument.capture());
    Exercise actualExercise = argument.getValue();
    assertEquals(exerciseName, actualExercise.name);
    assertEquals(caloriesBurned, actualExercise.caloriesBurned);
}

@Test
public void shouldUpdateView_whenBurnSuccessful() {
    // given the expected balance response
    int expectedBalance = 100;
    CalorieBalance calorieBalance = new CalorieBalance(300, 200, expectedBalance);
    when(api.burn(any(Exercise.class))).thenReturn(Calls.response(calorieBalance));

    // when burn is called on the presenter
    presenter.burn("name", 100);

    // then presenter should update view balance
    verify(view).showBalance(expectedBalance);
}
```

Clean architecture

Clean architecture

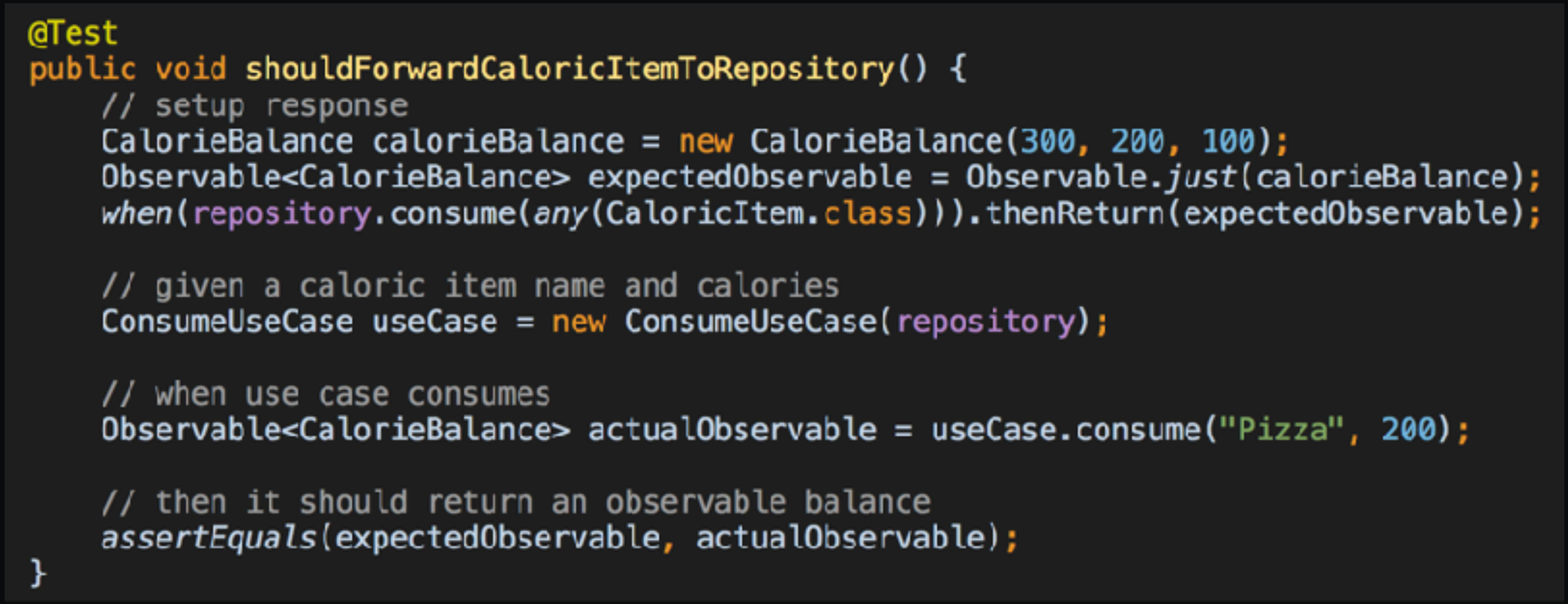


Clean architecture - Use case



```java
class ConsumeUseCase {

    private final Repository repository;

    ConsumeUseCase(Repository repository) {
        this.repository = repository;
    }

    public Observable<CalorieBalance> consume(String name, int calories) {
        CaloricItem caloricItem = new CaloricItem(name, calories);
        return repository.consume(caloricItem)
    }
}
```

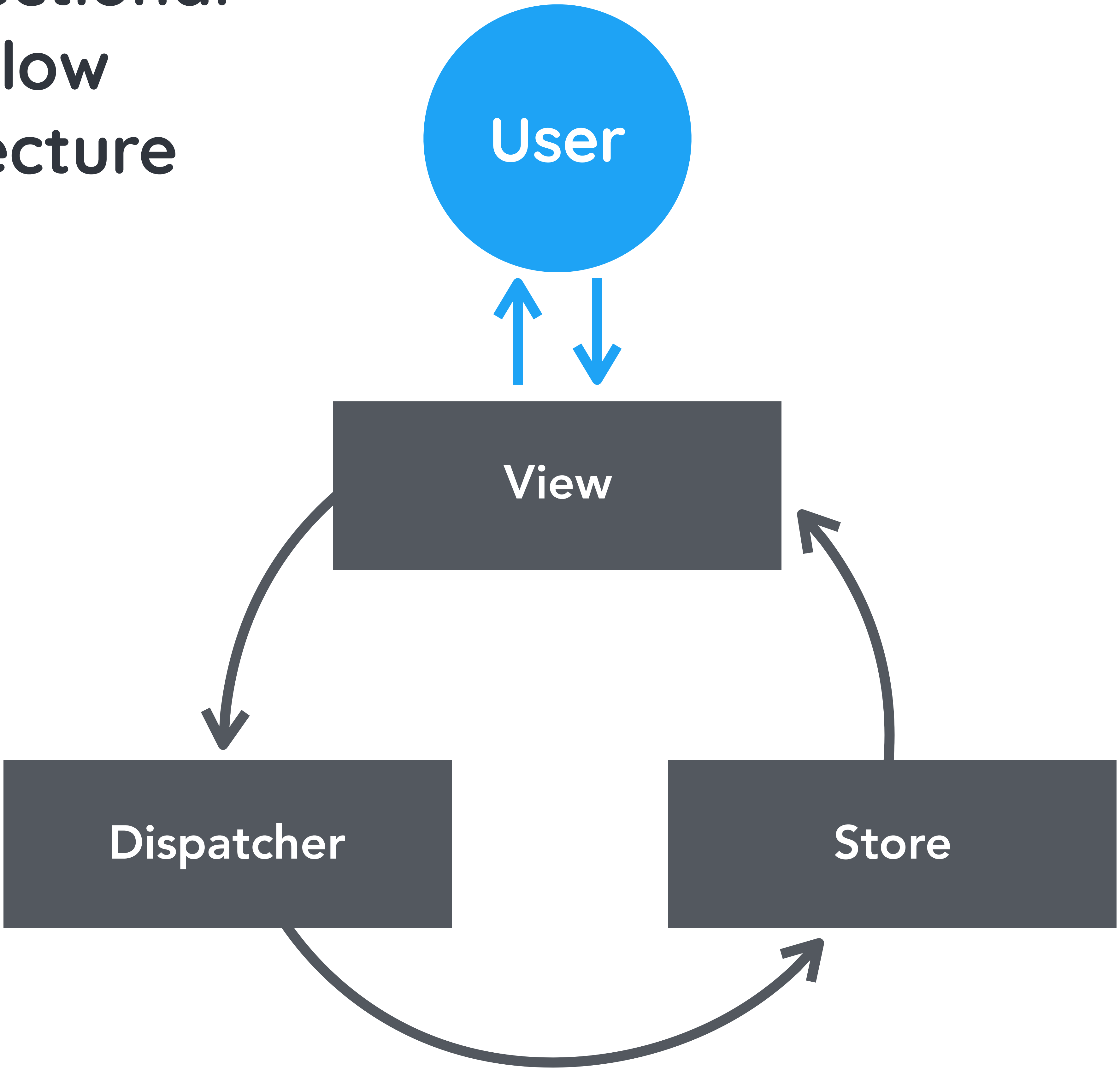
Clean architecture - Unit testing use case



```java
@Test
public void shouldForwardCaloricItemToRepository() {
    // setup response
    CalorieBalance calorieBalance = new CalorieBalance(300, 200, 100);
    Observable<CalorieBalance> expectedObservable = Observable.just(calorieBalance);
    when(repository.consume(any(CaloricItem.class))).thenReturn(expectedObservable);

    // given a caloric item name and calories
    ConsumeUseCase useCase = new ConsumeUseCase(repository);

    // when use case consumes
    Observable<CalorieBalance> actualObservable = useCase.consume("Pizza", 200);

    // then it should return an observable balance
    assertEquals(expectedObservable, actualObservable);
}
```

Unidirectional Data Flow architecture

Unidirectional Data Flow architecture

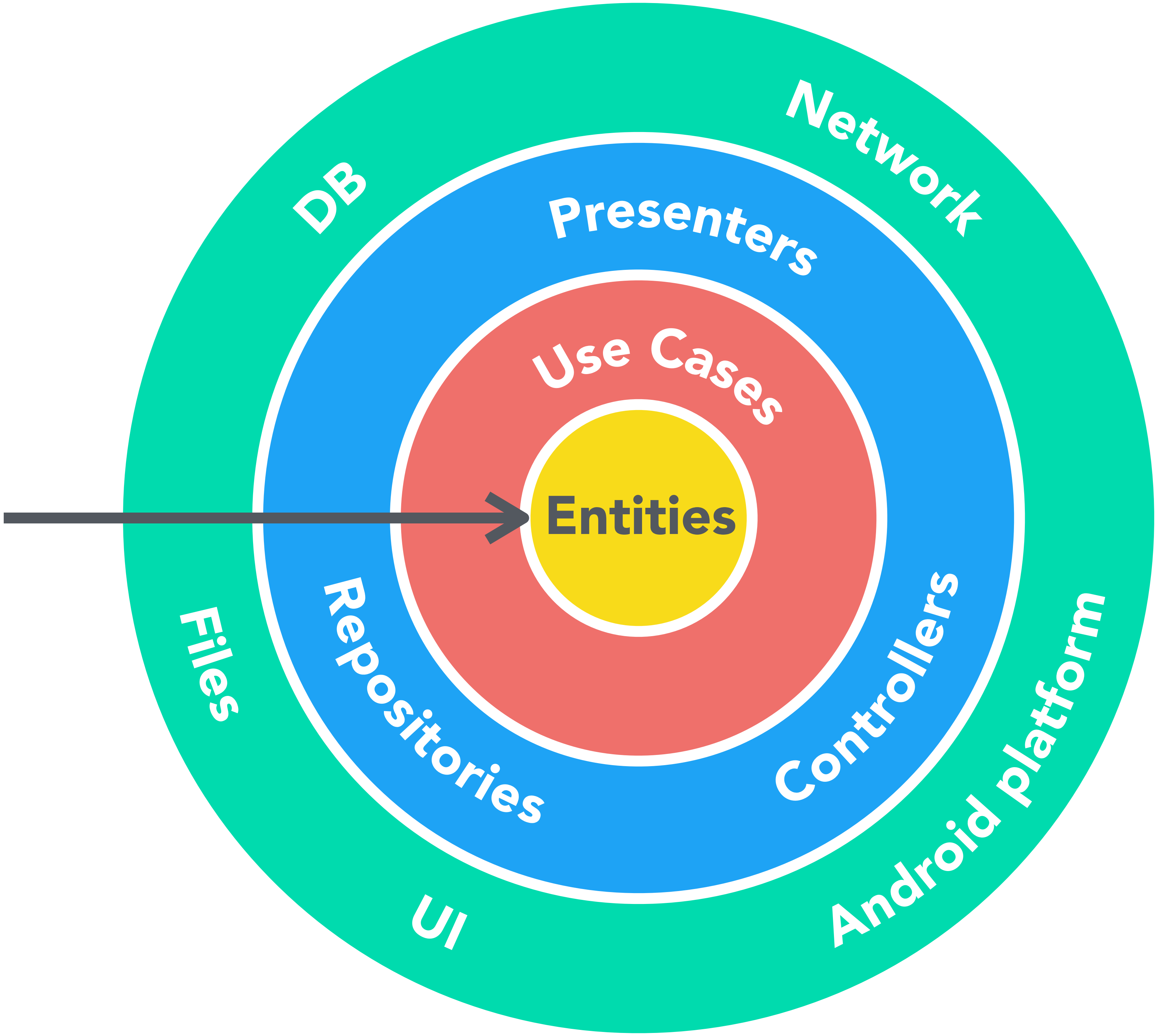


Unidirectional Data Flow architecture tests

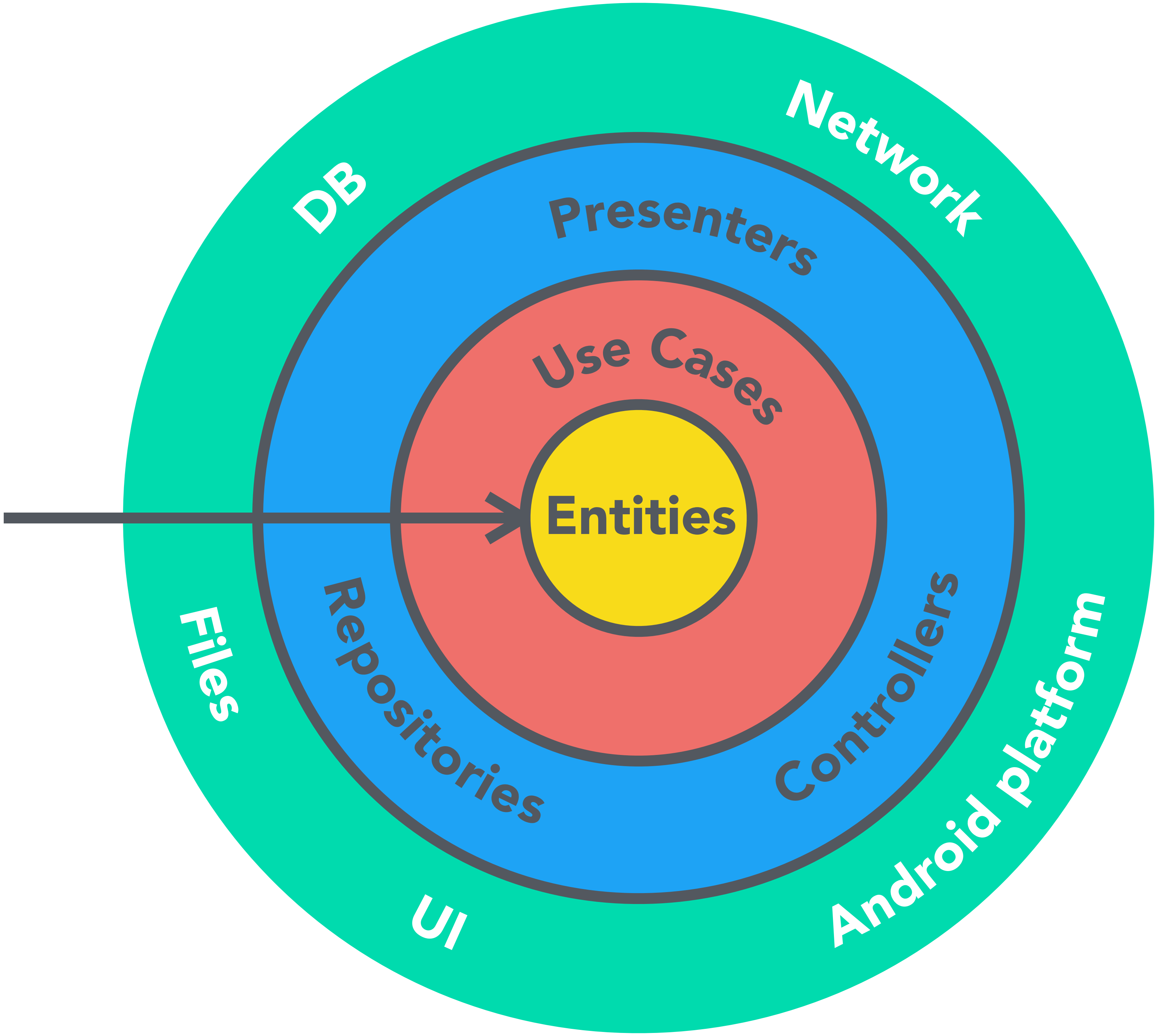
```
// TODO
```

Tips & Tricks

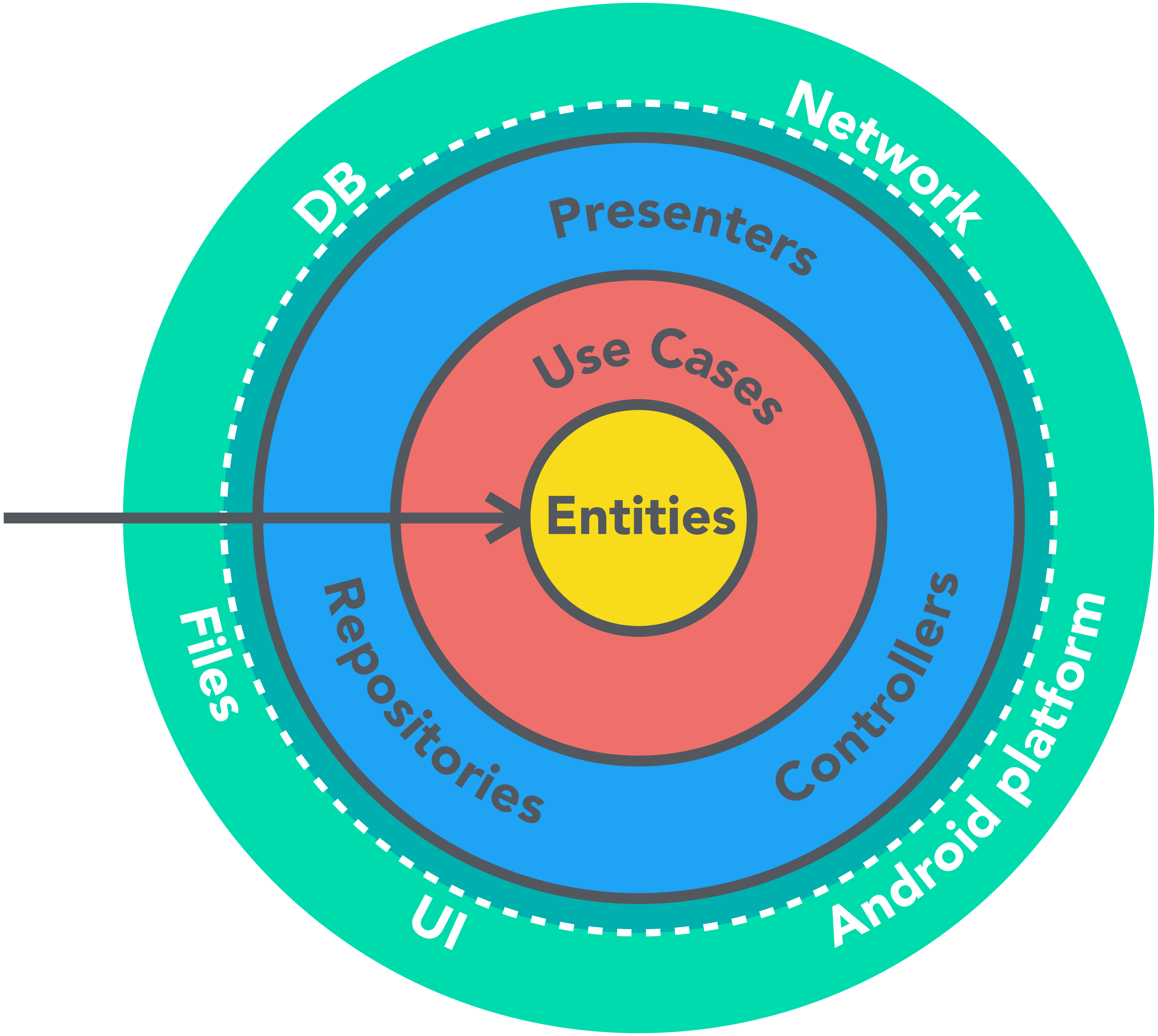
Gain speed


Network
DB
Presenters
Use Cases
Entities
Files
Repositories
Controllers
Android platform
UI

Gain speed - move to POJO module



Expand



Network
DB
Presenters
Use Cases
Entities
Files
Repositories
Controllers
Android platform
UI

Screenshot tests

Test design with screenshot tests for Android

https://facebook.github.io/screenshot-tests-for-android/

https://github.com/facebook/screenshot-tests-for-android

Screenshot tests

Test design with screenshot tests for Android

https://facebook.github.io/screenshot-tests-for-android/

https://github.com/facebook/screenshot-tests-for-android

Immutability

Object is immutable if its state cannot be changed after it is created.

Advantages:

- easier to reason about
- thread-safe
- easier testing

Pure functions

Side effects are produced when a function changes some state outside of its scope or perceived action.

Pure function returns a value based only on its input.

Recap

Recap

Recap

- agree upon a terminology

Recap

- agree upon a terminology
- define types of tests, tools and scope

Recap

- agree upon a terminology
- define types of tests, tools and scope
- utilize JVM as much as possible

Recap

- agree upon a terminology
- define types of tests, tools and scope
- utilize JVM as much as possible
- choose a suitable architecture

Recap

- agree upon a terminology
- define types of tests, tools and scope
- utilize JVM as much as possible
- choose a suitable architecture
- take it step-by-step

Recap

- agree upon a terminology
- define types of tests, tools and scope
- utilize JVM as much as possible
- choose a suitable architecture
- take it step-by-step
- make the process maintainable

Recap

- agree upon a terminology
- define types of tests, tools and scope
- utilize JVM as much as possible
- choose a suitable architecture
- take it step-by-step
- make the process maintainable
- reap the benefits

Q & A Time

Q & A Time

Dejan Tošić

dejan.tosic@undabot.com