



Espresso

1

User Interface Testing: Goals and Approach

- Focuses on writing functional **end-to-end UI tests**, which is the **closest way to replicate end user behavior** and catch potential issues before a product goes live.
- Can be much slower than unit or integration tests, but they usually discover issues that were not caught during the unit and integration testing stages.

2


Espresso for Android

- Espresso is a lightweight, fast, customizable testing framework for reliable **automated UI tests**
- First announced at 2013 by Google

3

Espresso for Android

- Part of Android Test Supporting Library (now, AndroidX Library)
- Simulate user interactions
- **Automatic synchronization** of test actions with app UI
 - The framework ensures that your activity is started before the tests run.
 - It also let the test wait until all observed background activities have finished.



```
nextButton.click();  
// Wait until button is rendered  
Thread.sleep(2000);  
previousButton.click();
```

4

Demo

```
// navigate to second activity
onView(withId(R.id.button_next_activity))
    .perform(click());

// check the second activity visible
onView(withId(R.id.secondary))
    .check(matches(isDisplayed()));

// navigate back to main activity
onView(withId(R.id.button_back))
    .perform(click()); // method 1

// check the main activity visible
onView(withId(R.id.main))
    .check(matches(isDisplayed()));
```



5

Espresso Packages

- **espresso-core**
 - Contains core and basic view matchers, actions, and assertions.
- **espresso-contrib**
 - External contributions that contain DatePicker, **RecyclerView**, and Drawer actions, accessibility checks, and the CountingIdlingResource.
- **espresso-intents**
 - Extensions to validate and stub intents for hermetic testing.
- **espresso-idling-resource**
 - Espresso's mechanism for synchronizing background jobs.
- **espresso-remote**
 - Location of Espresso's multi-process functionality.
- **espresso-web**
 - Contains resources for WebView support.

6

Add Dependencies to build.gradle

- Android Studio templates include dependencies
- If needed, add the following dependencies:

```
dependencies {  
    ...  
    testImplementation 'junit:junit:4.13'  
    androidTestImplementation 'androidx.test:runner:1.1.0'  
    androidTestImplementation 'androidx.test:core:1.1.0'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
    androidTestImplementation 'androidx.test:rules:1.2.0'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
    androidTestImplementation 'androidx.test.espresso:espresso-contrib:3.2.0'  
    androidTestImplementation 'androidx.test.espresso:espresso-intent:3.2.0'  
    androidTestImplementation 'androidx.test.espresso:espresso-idling-resource:3.2.0'  
    implementation 'androidx.test.espresso:espresso-idling-resource:3.2.0'  
}
```

7

Add defaultConfig to build.gradle

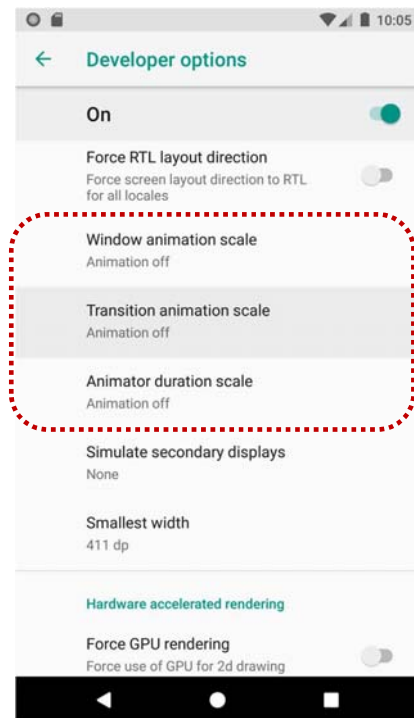
- Android Studio templates include **defaultConfig** setting.
- If needed, add the following to **defaultConfig** section:

```
defaultConfig {  
    ...  
    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
}
```

8

Prepare your device

1. Turn on USB Debugging
2. Turn off all animations in **Developer Options > Drawing**
 - Window animation scale
 - Transition animation scale
 - Animator duration scale



9

Create Tests

- Store in *module-name/src/androidTests/java/*
 - In Android Studio: app > java > *module-name* (androidTest)
- Create tests as JUnit classes



10

Five Components of Espresso

- JUnit Runner - runs the espresso test cases
- JUnit Rules - launches an activity
- *ViewMatchers* - allows to find view in the current view hierarchy
- *ViewActions* - allows to perform actions on the views
- *ViewAssertions* - allows to assert state of a view (i.e. validation)

11

JUnit Runner (*AndroidJUnitRunner*)

- Android testing framework provides a runner, *AndroidJUnitRunner* to run the espresso test cases (JUnit3 and JUnit4 test cases).
- It transparently handles
 - loading test cases and SUT both in actual device or emulator,
 - execute the test cases, and
 - report the result of the test cases.

```
@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
    ...
}
```

12

JUnit Rules

- Espresso needs a rule of type **ActivityScenarioRule** to specify the activity.
- **ActivityScenarioRule** launches an activity before executing the test cases.
 - launches the activity before *@Before* and
 - will terminate it after *@After*.

```
@Rule
public ActivityScenarioRule<MainActivity> scenarioRule =
    new ActivityScenarioRule<>(MainActivity.class);
```

13

@Rule specifies the context of testing

- **@ActivityScenarioRule** - Testing support for a single specified activity
- **@ActivityTestRule** - Testing support for a single specified activity
 - deprecated
- **@IntentsTestRule** – Subtype of ActivityTestRule for intents
- **@ServiceTestRule** - Testing support for starting, binding, shutting down a service

14

Formula

```
onView(ViewMatcher)           // find a View by id, content, focus, hierarchy
    .perform(ViewAction)       // Perform an action
    .check(ViewAssertion)      // Verify action was taken, assert result
```

```
onView(withId(R.id.greet_button))
    .perform(click())
    .check(matches(not(isEnabled())))
```

- **onView()** methods return an object of type **ViewInteraction**.

15

Espresso Matchers

```
onView(withId(R.id.next_button))
    .perform(click())
    .check(matches(not(isDisplayed()
)))
```

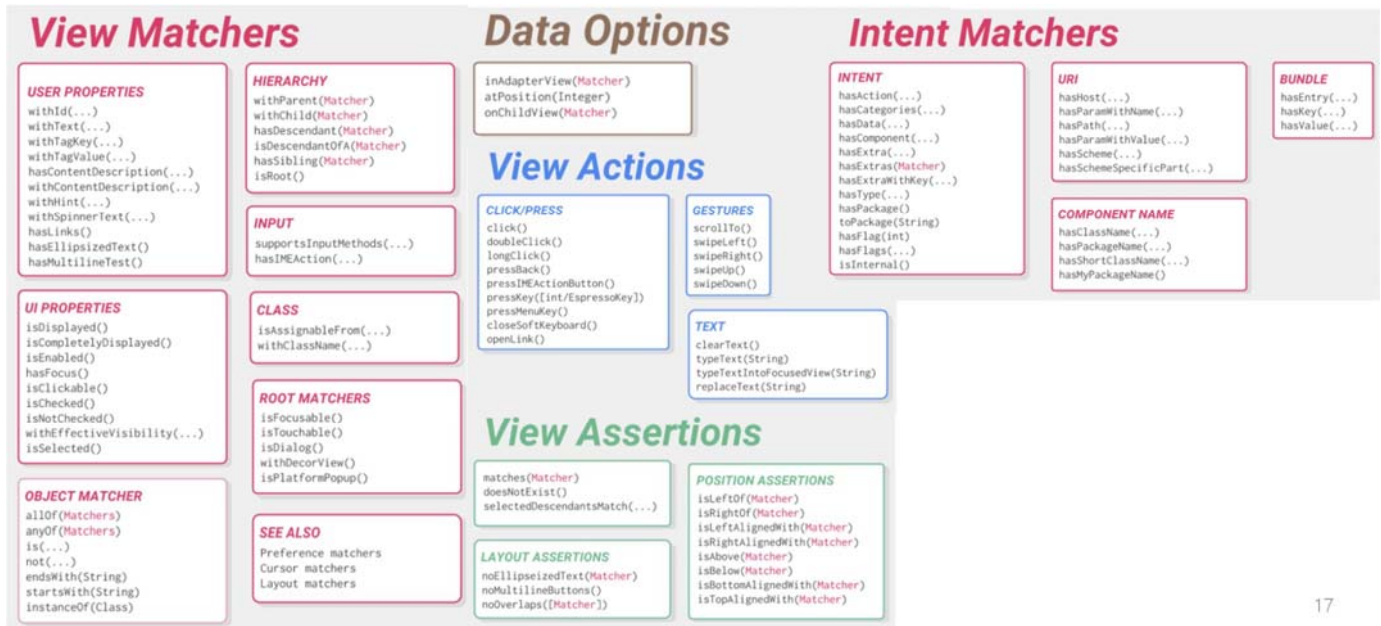
Hamcrest Matchers

```
onView(withId(R.id.next_button))
    .perform(click())
    .check(matches(not(isDisplayed()
)))
```

16

Espresso cheat sheet

(<https://android.github.io/android-test/downloads/espresso-cheat-sheet-2.1.0.pdf>)



17

"Hamcrest" simplifies tests

“Hamcrest” an anagram of “Matchers”

Framework for creating custom matchers and assertions

Match rules defined declaratively

Enables precise testing

The Hamcrest Tutorial

<http://hamcrest.org/JavaHamcrest/tutorial>

18

Hamcrest 1.3 Quick Reference

General purpose

```
is(T)
equalTo(T)
not(T) : Matcher<T>

anything()
anything(String) : Matcher<Object>

any(Class<T>)
instanceOf(Class<?>)
isA(Class<T>) : Matcher<T>

nullValue() : Matcher<Object>
nullValue(Class<T>) : Matcher<T>
notNullValue() : Matcher<Object>
notNullValue(Class<T>) : Matcher<T>

sameInstance(T)
theInstance(T) : Matcher<T>

isIn(Collection<T>)
isIn(T[])
isOneOf(T...)
hasToString(String)
hasToString(Matcher<? super String>) : Matcher<T>
```

Iterables

```
everyItem(Matcher<U>) : Matcher<Iterable<U>>

hasItem(T)
hasItem(Matcher<? super T>) : Matcher<Iterable<? super T>>

hasItems(T...)
hasItems(Matcher<? super T>...) : Matcher<Iterable<T>>

emptyIterable() : Matcher<Iterable<? extends E>>
emptyIterableOf(Class<E>) : Matcher<Iterable<E>>

contains(E...)
contains(Matcher<? super E>...)
contains(Matcher<? super E>)
contains(List<Matcher<? super E>>)
: Matcher<Iterable<? extends E>>

containsInAnyOrder(T...)
containsInAnyOrder(Collection<Matcher<? super T>>)
containsInAnyOrder(Matcher<? super T>...)
containsInAnyOrder(Matcher<? super E>)
: Matcher<Iterable<? extends E>>

iterableWithSize(Matcher<? super Integer>)
iterableWithSize(int) : Matcher<Iterable<E>>
```

19

Basic example test

```
@Test
public void changeText_sameActivity() {
    // 1: Find view by id
    onView(withId(R.id.editTextUserInput))

    // 2: Perform action-type string and click button
    .perform(typeText(mStringToBeTyped), closeSoftKeyboard());

    onView(withId(R.id.changeTextButton)).perform(click());

    // 3: Check that the text was changed
    onView(withId(R.id.textToBeChanged))
        .check(matches(withText(mStringToBeTyped)));
}
```

20

Finding views with onView

withId() - find a view with the specified Android id

```
onView(withId(R.id.editTextUserInput))
```

withText() - find a view with specific text

```
onView(withText("Espresso"))
```

21

onView returns ViewInteraction object

If you need to reuse the View returned by onView

Make code more readable or explicit

check() and perform() methods

```
ViewInteraction textView = onView(  
    allOf(withId(R.id.word), withText("Clicked! Word 15"),  
    isDisplayed()));  
textView.check(matches(withText("Clicked! Word 15 "))));
```

22

Perform actions

Perform an action on the View found by a ViewMatcher

Can be any action you can perform on the View

```
// 1: Find view by id  
onView(withId(R.id.editTextUserInput))  
// 2: Perform action - type string and click button  
.perform(typeText(mStringToBeTyped), closeSoftKeyboard());  
onView(withId(R.id.changeTextButton)).perform(click());
```

23

Check result

Asserts or checks the state of the View

```
// 3: Check that the text was changed  
onView(withId(R.id.textToBeChanged))  
    .check(matches(withText(mStringToBeTyped)));
```

24

When a test fails

Test

```
onView(withId(R.id.text_message))
    .check(matches(withText("This is a failing test.")));
```

Result snippet

```
androidx.test.espresso.base.DefaultFailureHandler$Assertion
FailedWithCauseError: 'with text: is "This is a failing test."'
does't match the selected view.
Expected: with text: is "This is a failing test."
Got: "AppCompatTextView{id=2131165359, res-name=text_message, ...
```

25

Access to the Instrumentation API

- Via the `ApplicationProvider.getApplicationContext()` you have access to the target context of your application.

```
@Test
public void buttonShouldUpdateText(){
    onView(withId(R.id.update)).perform(click());
    onView(withId(getResourceId("Click"))).check(matches(withText("Done")));
}

private static int getResourceId(String s) {
    Context targetContext = ApplicationProvider.getApplicationContext();
    String packageName = targetContext.getPackageName();
    return targetContext.getResources().getIdentifier(s, "id", packageName);
}
```

26

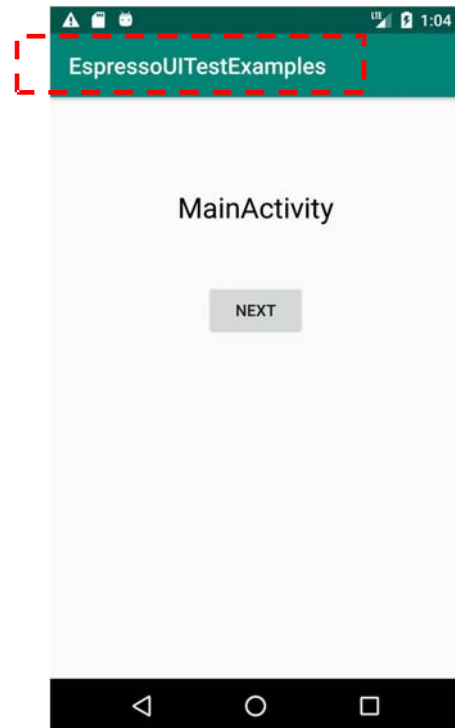
Combining Matchers

- **Example:** Find a visible list item with the given text:

allOf() - find a view to that matches multiple conditions

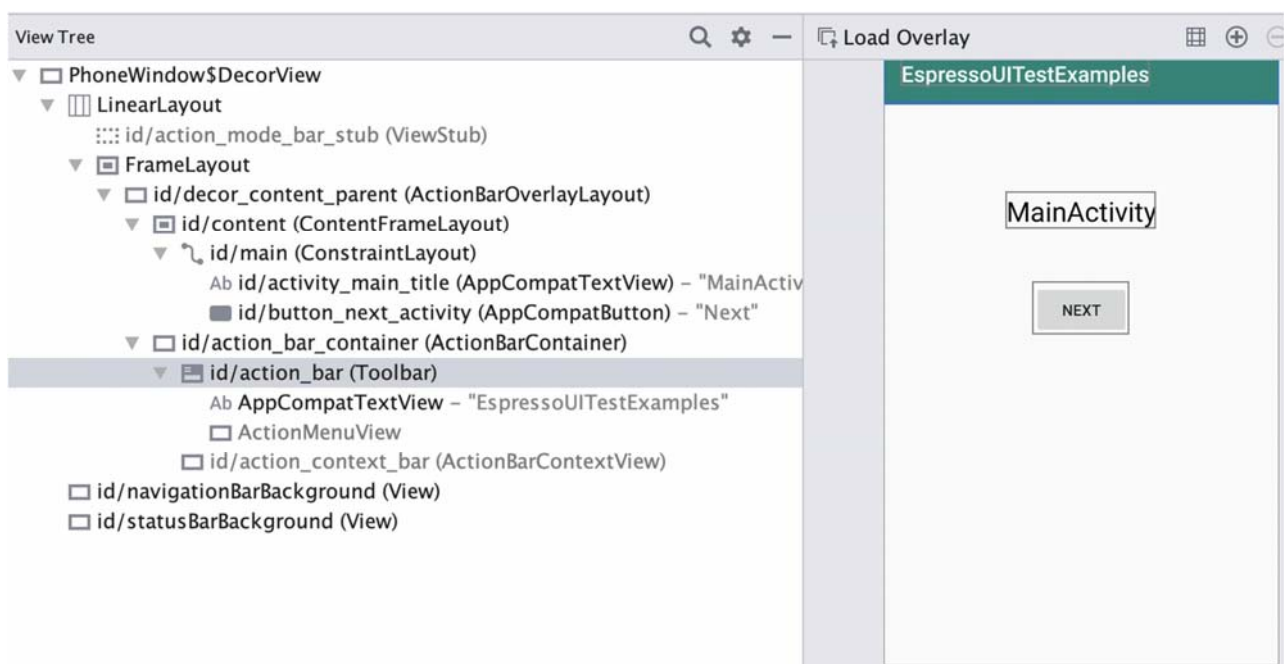
```
onView(allOf(withId(R.id.word),  
              withText("Clicked! Word 15"),  
              isDisplayed()));
```

- **Another Example:** Toolbar title



27

Layout Inspector (Hierarchy Viewer)



Combining Matchers

```
@Test
public void testToolbarTitle() throws Exception {
    CharSequence title =
        InstrumentationRegistry.getInstrumentation()
            .getTargetContext().getString(R.string.app_name);
    matchToolbarTitle(title);
}

private ViewInteraction matchToolbarTitle(CharSequence title) {
    return onView(
        allOf(
            isAssignableFrom(Textview.class),
            withParent(isAssignableFrom(Toolbar.class)))
        ).check(matches(withText(title.toString())));
}
```

29

Custom Matchers

```
private ViewInteraction matchToolbarTitle(CharSequence title) {
    return onView(isAssignableFrom(Toolbar.class))
        .check(matches(withToolbarTitle(is(title))));
}
```

30

Custom Matcher (using Toolbar.getTitle())

```
public static Matcher<View> withToolbarTitle(CharSequence title) {
    return withToolbarTitle(is(title));
}

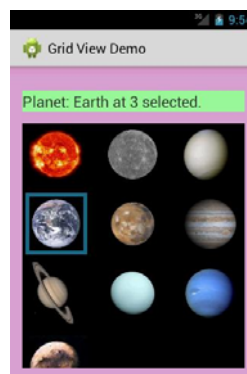
public static Matcher<View> withToolbarTitle(Matcher<CharSequence> textMatcher)
{
    return new BoundedMatcher<View, Toolbar>(Toolbar.class) {
        @Override public boolean matchesSafely(Toolbar toolbar) {
            return textMatcher.matches(toolbar.getTitle());
        }

        @Override public void describeTo(Description description) {
            description.appendText("with toolbar title: ");
            textMatcher.describeTo(description);
        }
    };
};
```

31

Adapter Views

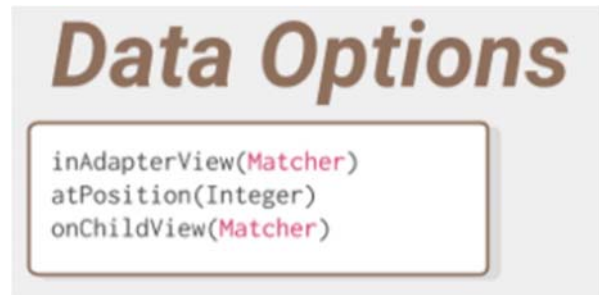
- Use **onData()** instead of **onView()** when working with **AdapterViews**.
 - ListView, GridView and Spinner



32

Formula

```
onData(ObjectMatcher)  
    .dataOptions  
    .perform(ViewAction)  
    .check(ViewAssertion)
```



- If you are using an **AdapterView**, use the `onData()` instead of `onView()`.
- The `onData()` method returns an object of type **DataInteraction**.

33

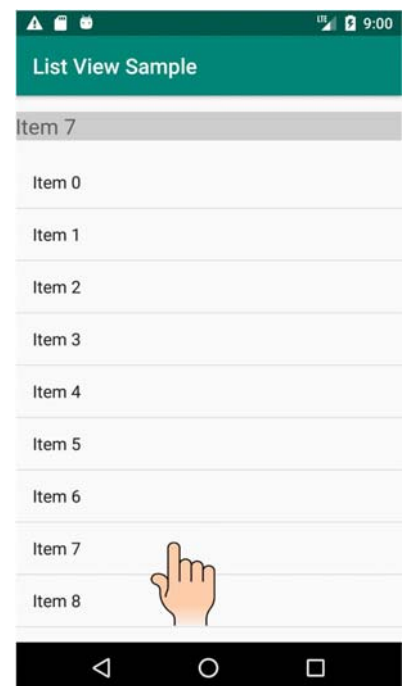
App

```
TextView textMsg = findViewById(R.id.textMsg);  
textMsg.setBackgroundColor(Color.LTGRAY);  
textMsg.setVisibility(View.GONE);  
ListView listView = findViewById(R.id.myList);
```

```
String[] items = new String[COUNT];  
for (int i = 0; i < COUNT; i++) {  
    items[i] = "Item " + i;  
}
```

```
listView.setAdapter(new ArrayAdapter<>(this,  
    android.R.layout.simple_list_item_1,  
    items));
```

```
listView.setOnItemClickListener(  
    (parent, view, position, id) -> {  
        textMsg.setText(items[position]);  
        textMsg.setVisibility(View.VISIBLE);  
    });
```



34

Test

```
onView(withId(R.id.textMsg))
    .check(matches(not(isDisplayed())));

// select view with text "Item 15"
onData(is("Item 15"))
    .inAdapterView(withId(R.id.myList))
    .perform(click());
// or ...
// onData(allOf(is(instanceOf(String.class)),
//             is("Item 15"))).perform(click());

onView(withId(R.id.textMsg))
    .check(matches(withText("Item 15")))
    .check(matches(isDisplayed()));
```



35

Useful Data Interactions

```
onData(withBookTitle("Effective Java"))
    .inAdapterView(allOf(
        isAssignableFrom(AdapterView.class),
        isDisplayed()))
    .perform(click());

onData(anything()).atPosition(2).perform(click());

onData(withBookTitle("Espresso Tutorial"))
    .onChildView(withId(R.id.book_delete)).perform(click());
```

36

Custom Data Matchers

```
public static Matcher<Object> withBookTitle(final String bookTitle)
{
    return new BoundedMatcher<Object, Book>(Book.class) {
        @Override
        protected boolean matchesSafely(Book book) {
            return bookTitle.equals(book.getTitle());
        }

        @Override
        public void describeTo(Description description) {
            description.appendText("with title: " + bookTitle);
        }
    };
}
```

37

Recycler Views

- You cannot use `onData()` for `RecyclerView`.
- Use `onView(R.id.recyclerview_id)` together with **`RecyclerViewActions`**.

```
java.lang.Object
└─ android.view.View
    └─ android.view.ViewGroup
        └─ androidx.recyclerview.widget.RecyclerView
```

38

RecyclerViewActions

```
onView(withId(R.id.textMsg))
    .check(matches(not(isDisplayed())));

// select view at position 15
onView(withId(R.id.recycler_view))
    .perform(
        RecyclerViewActions.actionOnItemAtPosition(
            15, click())
    );

onView(withId(R.id.textMsg))
    .check(matches(withText("Item 15")))
    .check(matches(isDisplayed()));
```



39

RecyclerViewActions (**espresso-contrib**)

- `actionOnItemAtPosition(int, ViewAction)`
- `actionOnItem(Matcher<View>, ViewAction)`
- `scrollToPosition(int)`
- `scrollTo(Matcher<View>)`
- `actionOnHolderItem(Matcher<ViewHolder>, ViewAction)`
- `scrollToHolder(Matcher<ViewHolder>)`

40

Comments on `actionOnItem()`

- You have to create a matcher that matches the **whole item**.



```
onView(withId(R.id.RecyclerView))  
    .perform(actionOnItem(withText( "Espresso"), click()));
```



```
onView(withId(R.id.RecyclerView))  
    .perform(actionOnItem(hasDescendant(withText( "Espresso")),  
        click()));
```



41

Mocking intents with Espresso Intents

- Espresso's intents are provided by the `com.android.support.test.espresso:espresso-intents` library.
- Use the `IntentsTestRule` instead of `ActivityTestRule`.

```
androidTestCompile 'com.android.support.test.espresso:espresso-intents:2.2.1'
```

42

(1) Intent Stub

Intending(Matcher<Intent> matcher)

- A fake response to an intent call during a test.

Intending(Matcher<Intent> matcher)

1. *@Rule* - **IntentsTestRule**
2. *@before* - stubbing an intent must be set up and we need to make sure all external intents are blocked. In android 6 (M) and later on we also need to grant for permission.
3. *@Test*

```
intending(hasComponent(hasShortClassName(".ContactsActivity")))
    .respondWith(new ActivityResult(Activity.RESULT_OK,
        ContactsActivity.createResultData(VALID_PHONE_NUMBER)));
```

43

(2) Intent Verification

- Make sure that the information that was intended to be sent was what actually sent, by using a hardcoded matcher.

*intended(Matcher<Intent> matcher,
VerificationMode verification)*

- *@Test*

```
intended(allOf(hasAction(Intent.ACTION_CALL),
    hasData(INTENT_DATA_PHONE_NUMBER),
    toPackage(PACKAGE_ANDROID_DIALER)));
```

44

IdlingResource Basics

On each time invocation of **onView()** or **onData()**, Espresso waits until the following synchronization conditions are met:

1. The message queue is empty.
2. There are no instances of **AsyncTask** currently executing a task.
3. All developer-defined idling resources are idle.

45

Espresso doesn't know about:

- Animations
- Background operations
- Other mechanisms to schedule updates
 - For example, Data Binding uses the Choreographer to post updates instead of the main Looper queue (what Espresso monitors).

46

Registration and Unregistration

@Before

```
public void registerIdlingResource() {  
    ...  
    IdlingRegistry.getInstance().register(myIdlingResource);  
}
```

@After

```
public void unregisterIdlingResource() {  
    ...  
    IdlingRegistry.getInstance().unregister(myIdlingResource);  
}
```

```
dependencies {  
    implementation "androidx.test.espresso:espresso-idling-resource:3.2.0"  
}
```

47

IdlingResource Basics

The common use cases in which **IdlingResource** can be used are when your app is:

- **Loading data** from the internet or a local data source.
- **Establishing connections** with databases and callbacks.
- **Managing services**, either using a system service or an instance of `IntentService`.
- **Performing complex business logic**, such as bitmap transformations.

It is important to register **IdlingResource** when these operations update the application UI you would like to further validate.

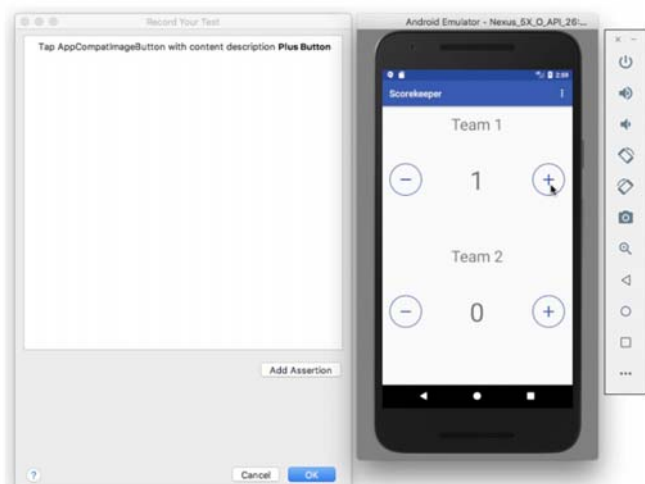
48

Recording Tests

49

Start recording an Espresso test

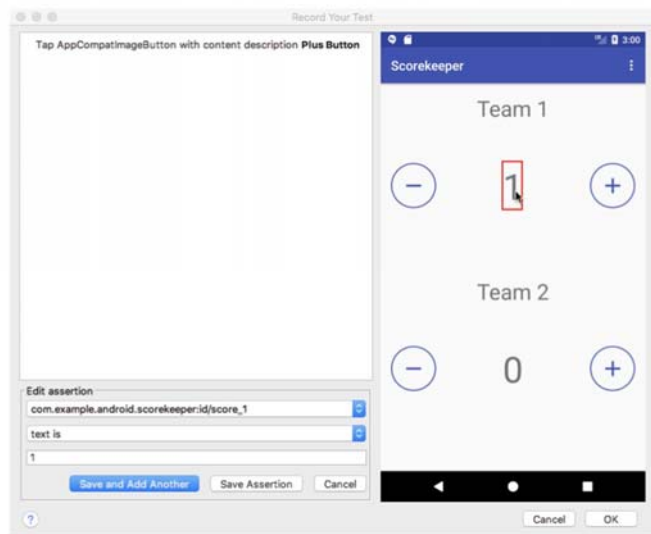
1. **Run > Record Espresso Test**
2. Click **Restart app**, select target, and click **OK**
3. Interact with the app to do what you want to test



50

Add assertion to Espresso test recording

4. Click **Add Assertion** and select a UI element
5. Choose **text is** and enter the text you expect to see
6. Click **Save Assertion** and click **Complete Recording**



51

52