

Android UI Testing

Hui Chen ^a

^aCUNY Brooklyn College, Brooklyn, NY, USA

March 15, 2022

Outline

- 1 Review for Last Class
- 2 Android UI Testing
 - Introduction to Espresso
 - Espresso Example
- 3 Summary
- 4 References

Outline

- 1 Review for Last Class
- 2 Android UI Testing
 - Introduction to Espresso
 - Espresso Example
- 3 Summary
- 4 References

Recall: Requirements

1. Discussed
 - ▶ Overview of requirement engineering
 - ▶ Agile vs. traditional (plan & document)
2. An agile approach of requirement analysis
 - ▶ Design user stories for/as requirements
 - ▶ In Behavior-Driven Development (BDD), map a user story to one or more scenarios
 - ▶ Question: How do we ensure that our “stories” are acceptable by the users? → acceptance tests?
 - ▶ Each scenario can be an acceptance test
3. Your project
 - ▶ User stories, storyboards, scenarios
 - ▶ How do we do acceptance testing here?

Scenarios

Question: How do we ensure that our “stories” are acceptable by the users? → acceptance tests?

Map user stories to multiple testable scenarios

Format: Given/When/Then.

- ▶ Given: some specific starting condition(s),
- ▶ When: I take specific action X,
- ▶ Then: one or more specific thing(s) should happen

But, how?

Outline

- 1 Review for Last Class
- 2 Android UI Testing
 - Introduction to Espresso
 - Espresso Example
- 3 Summary
- 4 References

Android UI Testing

Android UI testing can serve multiple purposes, our goal is:

- ▶ Automate the interaction behavior of an imaginary user of your app (e.g. clicks, typing, swiping, etc.)
- ▶ with which, we can test the acceptance of a given scenario.

There are several libraries that can help you do this efficiently, we discuss

- ▶ Espresso

Espresso

Part of Android Support library, three basic components

- ▶ ViewMatchers
- ▶ ViewActions
- ▶ ViewAssertions

Note that `View` is the base class for most UI widgets

Testing Scenarios with Espresso

- ▶ *Given* steps represent state of the app before event, i.e., *preconditions*
 - ▶ use ViewMatchers and ViewActions to create the desired state
 - ▶ can check preconditions via ViewAssert if necessary
- ▶ *When* steps represent event
 - ▶ e.g., simulate user pushing a button
 - ▶ use ViewMatchers and ViewActions
- ▶ *Then* steps represent expected *postconditions*; check if true
ViewAsserts in Espresso

Basic Espresso Test

```
onView( ViewMatcher )  
    . perform ( ViewAction )  
    . check ( ViewAssertion );
```

1. Finds the view
2. Performs an action on the view
3. Validates an assertion

```
@Test  
public void greeterSaysHello() {  
    onView(withId(R.id.name_field)).perform(typeText("Steve"));  
    onView(withId(R.id.greet_button)).perform(click());  
    onView(withText("Hello Steve!")).check(matches(isDisplayed()));  
}
```

Using Espresso: Preparation and Steps

- ▶ Generally, add some references to Gradle build
- ▶ In the test part of the Java code of your project (where?)
 1. Subclass one of the Android test classes (e.g. `ActivityInstrumentationTestCase2`)
 2. Write your test
 3. Run tests by right-clicking test class and selecting Run from the context menu

Reference: Tutorial on the Android Developer site (show)

<https://developer.android.com/training/testing/espresso/setup>

A Simple User Story

```
1 public class MainActivity extends Activity {
2     @Override
3     protected void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.activity_main);
6
7         Button buttonClear = (Button) findViewById(R.id.button_clear);
8         buttonClear.setOnClickListener(
9             new View.OnClickListener() {
10                public void onClick(View v) {
11                    TextView viewHello = (TextView) findViewById(R.id.
12                    textview_hello);
13                    if (viewHello.getText().toString() == getString(R.string.
14                    hello_word)) {
15                        viewHello.setText("");
16                    } else {
17                        viewHello.setText(getString(R.string.hello_world));
18                    }
19                }
20            }
21 );
22 }
```

Be aware that the above implementation may be buggy

User Story and Scenarios?

What is the user story?

Can you elaborate 3 scenarios for this user story?

Testing a Scenario

First attempt

```
1 public void testChangeTestOnMainActivity() {  
2     onView(withId(R.id.button_clear).perform(click()));  
3     onView(withId(R.id.textview_hello)).check(matches(withText(R.  
4         string.hello_word)));  
5 }
```

- ▶ onView — finds the view (See [Espresso](#))
- ▶ perform — performs an action in the view
- ▶ check — validates the assertion

Any problem with this test?

ViewMatchers

To find a view, use the `onView()` method with a [view matcher](#) which selects the correct view

ViewMatcher	Description
<code>withText("SomeText")</code>	Searches for view with the specified text.
<code>withId()</code>	Searches for view with the ID
Hamcrest Matchers	Can use Hamcrest matchers, e.g., <code>containsString</code> or <code>instanceOf</code> , etc. e.g., <code>onView(allOf(withId(R.id.button_login), not(withText("Logout")))</code>

ViewActions

Allow to specify an `action` for test via an object of type `ViewAction` via the `perform()` method

- ▶ `ViewActions.click()`
- ▶ `ViewActions.typeText()`
- ▶ `ViewActions.pressKey()`
- ▶ `ViewActions.clearText()`

ViewAssertions

Call the `check()` method to `assert` a view state. This method expects a `ViewAssertion` object as input.

- ▶ `matches` – Hamcrest matcher
- ▶ `doesNotExist` – asserts that the select view does not exist

Outline

- 1 Review for Last Class
- 2 Android UI Testing
 - Introduction to Espresso
 - Espresso Example
- 3 Summary
- 4 References

Summary

Android UI testing

Use Espresso

Can use it as acceptance test of user stories

Outline

- 1 Review for Last Class
- 2 Android UI Testing
 - Introduction to Espresso
 - Espresso Example
- 3 Summary
- 4 References

References

[UIAutomator with Espresso](#)

[Espresso Cheat Sheet](#)