20CSI605 - Mobile Application Development

Testing your UI

Testing your UI

Properly testing a user interface(UI)

- Writing and running tests is an important part of the Android app-development cycle.
- Well-written tests can help you catch bugs early in your development cycle, where bugs are easier to fix.
- In *user interface (UI) testing*, you focus on aspects of the UI and the app's interactions with users. Recognizing and acting on user input is a high priority in UI testing and validation.
- As a developer, you should get into the habit of testing user interactions to ensure that users don't encounter unexpected results or have a poor experience when interacting with your app.

Testing your UI

An app's UI contains View elements such as buttons, menus, and text fields, each with a set of properties.

To properly test a UI, you need to do the following:

- ✓ Exercise all UI events that use View elements. To do this, tap a View in the app and enter data or make a choice. Then examine the values of the properties of each View—referred to as the state of the UI—at different times during execution.
- ✓ Provide inputs to all UI View elements. Use this opportunity to test improper inputs. For example, input letters in a view where numbers are expected.
- ✓ Check the outputs and UI representations of data—such as strings and integers—to see if they are consistent with what you expect.

Testing your UI

- In addition to functionality, UI testing evaluates design elements such as layout, colors, fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links, and content.

Manual testing

To summarize, the problems inherent with manual testing fall into two categories:

- ✓ **Domain size:** A UI has many operations that need to be tested. Even a relatively small app can have hundreds of possible UI operations.
- ✓ **Sequences:** Some functionality of the app may only be accomplished with a sequence of UI events.

Testing your UI

Automatic testing

When you automate tests of user interactions, you free yourself and your resources for other work

For testing Android apps, you typically create these types of automated UI tests:

UI tests that work within a single app: Verifies that the app behaves as expected when a user performs a specific action or enters a specific input

Example: **Espresso**

UI tests that span multiple apps: Verifies the correct behavior of interactions between different user apps or between user apps and system apps

Testing your UI

Using Espresso for tests that span a single app

You can use Espresso to create UI tests to automatically verify the following:

- ✓ The app returns the correct UI output in response to a sequence of user actions on a device.
- ✓ The app's navigation and input controls bring up the correct Activity and View elements.
- ✓ The app responds correctly with mocked-up dependencies, such as data from an outside server, or can work with stubbed out backend methods to simulate real interactions with backend components which can be programmed to reply with a set of defined responses.

Testing your UI

Using UI Automator for tests that span multiple apps

- The UI Automator testing framework in the Android Testing Support Library can help you verify the correct behavior of interactions between different user apps or between user apps and system apps.
- It can also show you what is happening on the device before and after an app is launched.
- The UI Automator APIs let you interact with visible elements on a device.
- Your test can look up a UI component by using descriptors such as the text displayed in that component or its content description

Testing your UI

The following are important functions of UI Automator:

- Like Espresso, UI Automator has access to system interaction information so that you can monitor all of the interaction the Android system has with the app.
- Your test can send an Intent or launch an Activity (without using shell commands) by getting a Context object through getContext().
- You can simulate user interactions on a collection of items, such as songs in a music album or a list of emails in an inbox.
- You can simulate vertical or horizontal scrolling across a display.
- You can use standard JUnit Assert methods to test that UI components in the app return the expected results.

Testing your UI

Setting up your test environment

To use the Espresso and UI Automator frameworks, you need to store the source files for instrumented tests at module-name/src/androidTests/java/

This directory already exists when you create a new Android Studio project. In the **Project > Android** view of Android Studio, show this directory by navigating to **app > java >** ***module-name*** **(androidTest)**.

You also need to do the following:

- Install the Android Support Repository and the Android Testing Support Library.
- Add dependencies to the project's build.gradle (Module: app) file.
- Create test files in the androidTest directory.

Testing your UI

Installing the Android Support Repository and Testing Support Library

You may already have the Android Support Repository and its Android Testing Support Library installed with Android Studio.

To check for the Android Support Repository, follow these steps:

1. In Android Studio select Tools > Android > SDK Manager.
2. Click the SDK Tools tab and look for the Support Repository.
3. If necessary, update or install the library.

Testing your UI

Adding dependencies:

When you start a project for the Phone and Tablet form factor using API 15: Android 4.0.3 (Ice Cream Sandwich) as the minimum SDK, Android Studio automatically includes the dependencies you need to use Espresso.

To ensure that you have these dependencies, follow these steps:

- Open the build.gradle (Module: app) file.
- Check if the following is included (along with other dependencies) in the dependencies section:

```
testImplementation 'junit:junit:4.12'

androidTestImplementation 'com.android.support.test:runner:1.0.1'

androidTestImplementation

            'com.android.support.test.espresso:espresso-core:3.0.1'
```

Testing your UI

Adding dependencies:

The following annotations are useful for testing:

@RunWith

@SmallTest, @MediumTest, and @LargeTest

@Rule

@Test

@Before and @After

Testing your UI

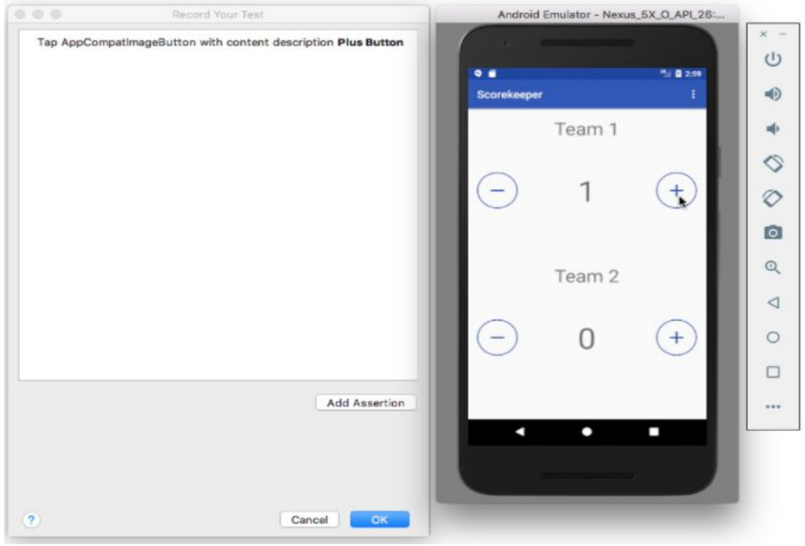
Recording a test:

- ✓ An Android Studio feature (in version 2.2 and newer) lets you record an Espresso test, creating the test automatically.
- ✓ After choosing to record a test, use your app as a normal user would. As you click through the app UI, editable test code is generated for you. Add assertions to check if a View holds a certain value.
- ✓ You can record multiple interactions with the UI in one recording session. You can also record multiple tests, and edit the tests to perform more actions, using the recorded code as a snippet to copy, paste, and edit.

Testing your UI

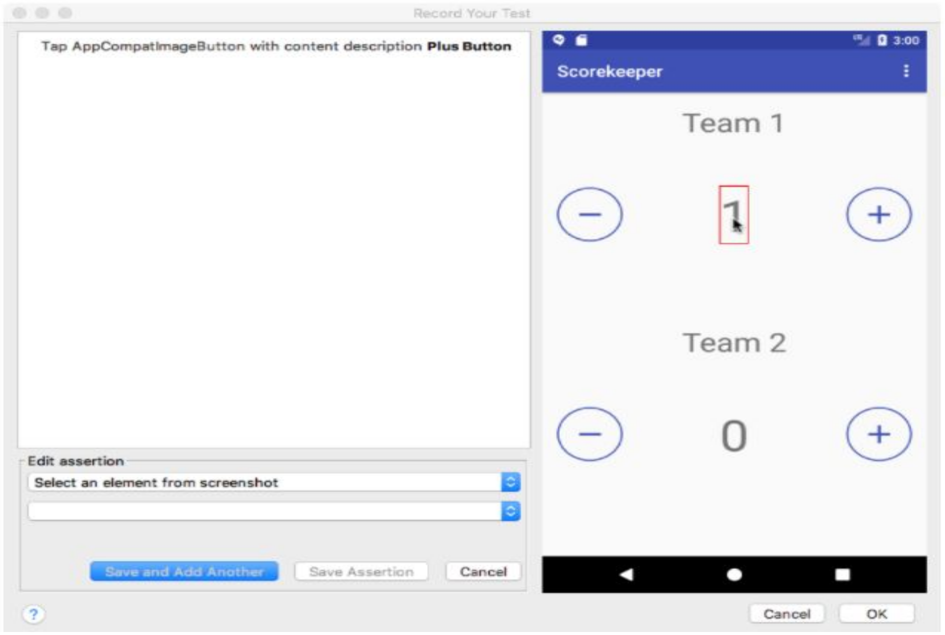
Follow these steps to record a test:

1. Select **Run > Record Espresso Test**, select your deployment target (an emulator or a device), and click **OK**.
2. Interact with the UI to do what you want to test. In this case, tap the plus (+) ImageButton for Team 1 in the app. The Record Your Test window shows the action that was recorded.


Record Your Test
Tap AppCompatImageButton with content description **Plus Button**
Add Assertion
Cancel
OK
Android Emulator - Nexus_5X_O_API_26:...
Scorekeeper
Team 1
1
Team 2
0

Testing your UI

3.Click Add Assertion in the Record Your Test window. A screenshot of the app's UI appears in a pane on the right side of the window, and the Select an element from screenshot option appears in the dropdown menu. Select the score (1) in the screenshot as the UI element you want to check, as shown in the figure below.

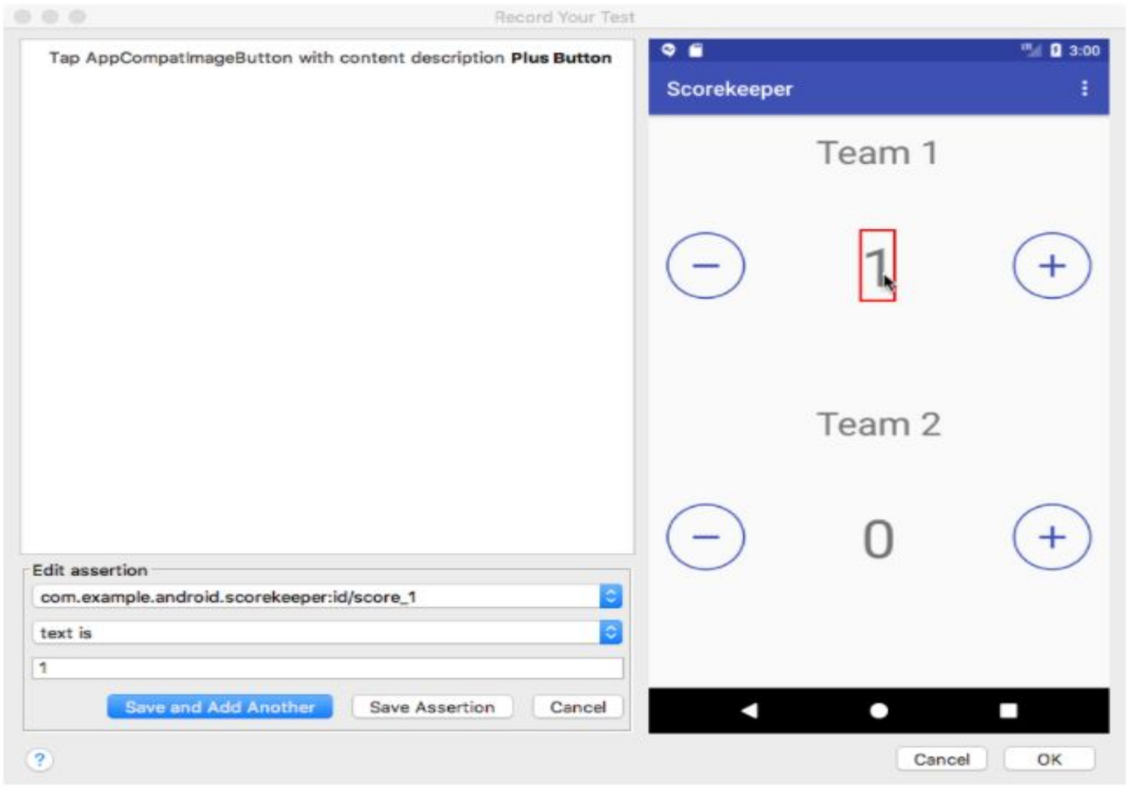


Tap AppCompatImageButton with content description **Plus Button**

Select an element from screenshot

Testing your UI

4. Select text is from the second dropdown menu, as shown in the figure below. The text you expect to see (1) is already entered in the field below the dropdown menu.



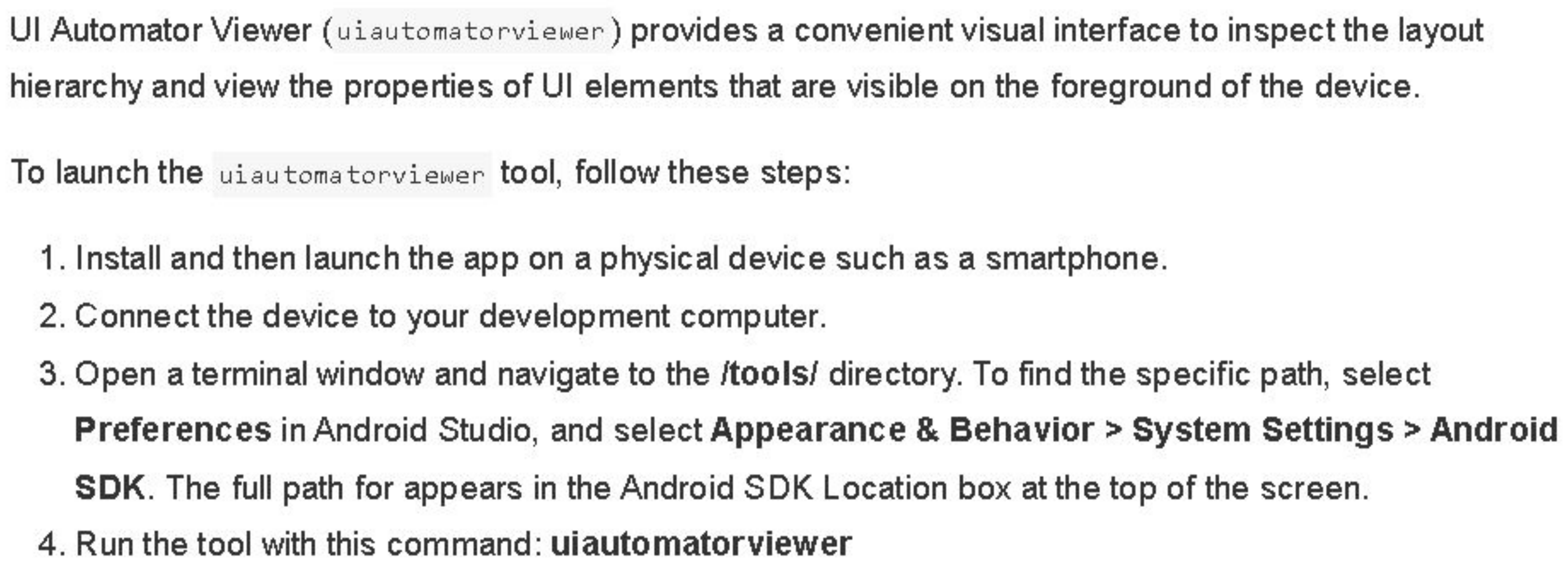
Testing your UI

1. Click Save Assertion, and then click OK.
2. In the dialog that appears, edit the name of the test so that it is easy for others to understand the purpose of the test.
3. Android Studio may display a request to add more dependencies to your Gradle Build file. Click Yes to add the dependencies. Android Studio adds the following to the dependencies section of the build.gradle (Module: app) file:

```
androidTestCompile
        'com.android.support.test.espresso:espresso-contrib:2.2.2', {
    exclude group: 'com.android.support', module: 'support-annotations'
    exclude group: 'com.android.support', module: 'support-v4'
    exclude group: 'com.android.support', module: 'design'
    exclude group: 'com.android.support', module: 'recyclerview-v7'
}
```

Testing your UI

Use UI Automator Viewer to inspect the UI on a device



UI Automator Viewer (`uiautomatorviewer`) provides a convenient visual interface to inspect the layout hierarchy and view the properties of UI elements that are visible on the foreground of the device.

To launch the `uiautomatorviewer` tool, follow these steps:

1. Install and then launch the app on a physical device such as a smartphone.
2. Connect the device to your development computer.
3. Open a terminal window and navigate to the ***/tools/*** directory. To find the specific path, select **Preferences** in Android Studio, and select **Appearance & Behavior > System Settings > Android SDK**. The full path for appears in the Android SDK Location box at the top of the screen.
4. Run the tool with this command: **uiautomatorviewer**

THANK YOU