

# ITW202: Mobile Application

## Unit IV: Developing for Android

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology  
Royal University of Bhutan

April 23, 2021

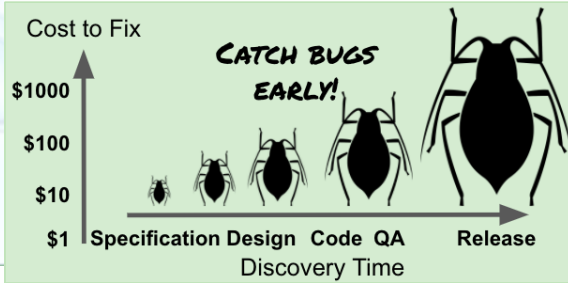
# Mobile Application Development

## App testing



# Why should you test your app?

- Find and fix issues early
- Less costly
- Takes less effort
- Costs to fix bugs increases with time



# Types of tests

- 1. Local Unit Test :

Are tests that are compiled and run entirely on your local machine with the Java Virtual Machine (JVM).

- Use local unit tests to test the parts of your app (such as the internal logic) that do not need access to the Android framework or an Android-powered device or emulator,

# Types of tests

- 2. Instrumented Test:

Are tests that run on an Android-powered device or emulator. You can use instrumented tests for unit testing, user interface (UI) testing, or integration testing, making sure that the components of your app interact correctly with other apps.

- Most commonly, you use instrumented tests for UI testing, which allows you to test that your app behaves correctly when a user interacts with your app or enters a specific input.

# Test-Driven Development (TDD)

## Mobile Application Development

Is a popular software development philosophy that places tests at the core of all software development for an app or service.



# Test-Driven Development (TDD)

- Define a test case for a requirement
- Write tests that assert all conditions of the test case
- Write code against the test
- Iterate on and refactor code until it passes the test
- Repeat until all requirements have test cases, all tests pass, and all functionality has been implemented.

# Unit Testing

## Mobile Application Development

- Unit tests should be the fundamental tests in your app testing strategy.
- A unit test generally exercises the functionality of the smallest possible unit of code (which could be a method, class, or component) in a repeatable way.
- Create unit tests when you need to verify the logic of specific code in your app.



# Unit Testing

## Mobile Application Development

**Note:** A common unit testing framework for Java code is JUnit 4.



# Android Studio source sets

## Mobile Application Development

Source sets are collections of code in your project that are for different build targets or other "flavors" of your app.

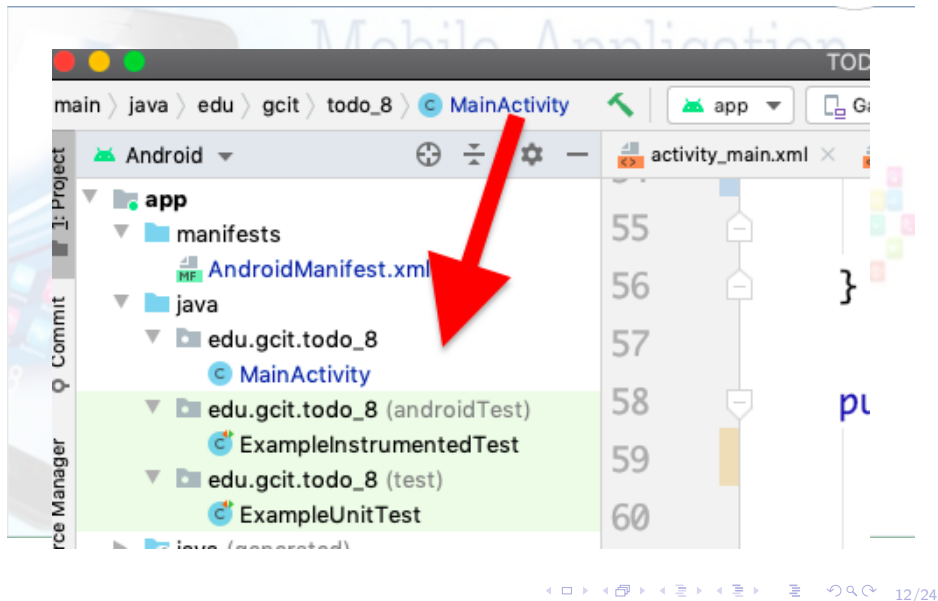


# Android Studio source sets

When Android Studio creates your project, it creates three source sets for you:

- The main source set, for your app's code and resources.
- The (test) source set, for your app's local unit tests. The source set shows (test) after the package name.
- The (androidTest) source set, for Android instrumented tests. The source set shows (androidTest) after the package name

# Android Studio source sets



# Configuring Gradle for test dependencies

## Mobile Application Development

In your app project's build.gradle (Module: app) file, the following dependency should already be included (if not, you should add it):

**testImplementation 'junit:junit:4.12'**

# Creating and running unit tests

## Mobile Application Development

Create your unit tests as a generic Java file using the JUnit 4 APIs, and store those tests in the (test) source set. Each Android Studio project template includes this source set and a sample Java test file called `ExampleUnitTest`.

# Creating a new test class

To add a new test class file, follow these steps:

- Expand the java folder and the folder for your app's test source set. The existing unit test class files are shown.
- Right-click (or Control-click) on the test source set folder and select New > Java Class.
- Name the file and click OK.

# Writing your tests

Use JUnit 4 syntax and annotations to write your tests.

- The `@RunWith` annotation indicates the test runner that should be used for the tests in this class.
- The `@SmallTest` annotation indicates that this is a small (and fast) test.



# Writing your tests

- The `@Before` annotation marks a method as being the setup for the test.
- The `@Test` annotation marks a method as an actual test.
- `setUp()` methods: Sets up environment for testing and Initialize variables and objects used in multiple tests

# Writing your tests

```
ExampleUnitTest.java x build.gradle (:app) x activity_main.xml x dimens.xml x strings.xml
@RunWith(JUnit4.class)
@SmallTest
public class ExampleUnitTest {

    private Calculator mCalculator;

    @Before
    public void setUp(){
        mCalculator = new Calculator();
    }
    /*
     * Test for simple addition
     * */
    @Test
    public void addTwoNumbers() {
        double resultAdd = mCalculator.add(1d, 1d);
```

# Writing your tests

## Mobile Application Development

The key part of a unit test is the **assertion**, which is defined here by the **assertThat()** method.

**Assertions** are expressions that must evaluate and result in a value of true for the test to pass.



# Writing your tests

## Mobile Application Development

JUnit 4 provides a number of assertion methods, but `assertThat()` is the most flexible, as it allows for general-purpose comparison methods called matchers.

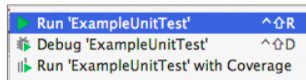


# Writing your tests

**Note:** The general rule for unit tests is to provide a separate test method for every individual assertion. Grouping more than one assertion into a single method can make your tests harder to debug if only one assertion fails, and obscures the tests that do succeed.

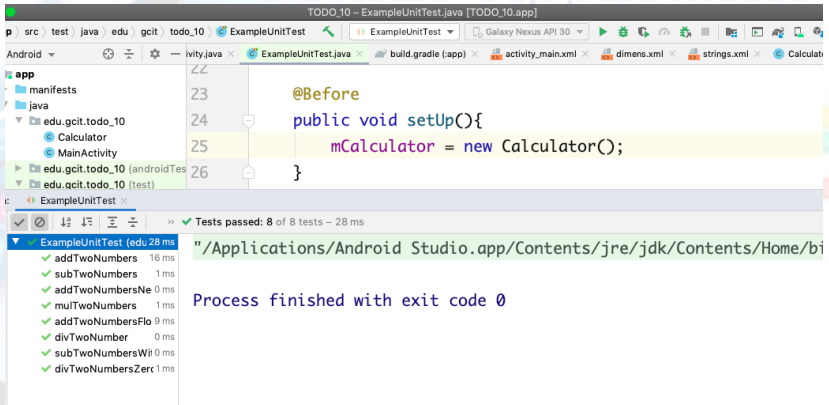
# Running your tests

- **Right-click** test class and select **Run 'app\_name' test**
- **Right-click** test package and select **Run tests in 'package'**



# Running your tests

## Mobile Application



The screenshot displays the Android Studio interface during a unit test run. The top toolbar shows the 'Run' button (a green play icon) being clicked. The editor window shows the `ExampleUnitTest.java` file with the following code:

```
23 @Before
24 public void setUp(){
25     mCalculator = new Calculator();
26 }
```

Below the editor, the 'Test' tab is active, showing the test results. The status bar indicates 'Tests passed: 8 of 8 tests - 28 ms'. The test results list is expanded, showing the following tests and their durations:

- ExampleUnitTest (edu 28 ms)
- addTwoNumbers 16 ms
- subTwoNumbers 1 ms
- addTwoNumbersNe 0 ms
- mulTwoNumbers 1 ms
- addTwoNumbersFlo 9 ms
- divTwoNumber 0 ms
- subTwoNumbersWit 0 ms
- divTwoNumbersZerc 1 ms

The output window at the bottom shows the path `"/Applications/Android Studio.app/Contents/jre/jdk/Contents/Home/bi`.

Process finished with exit code 0

# Mobile Application Development

**THANK YOU**

