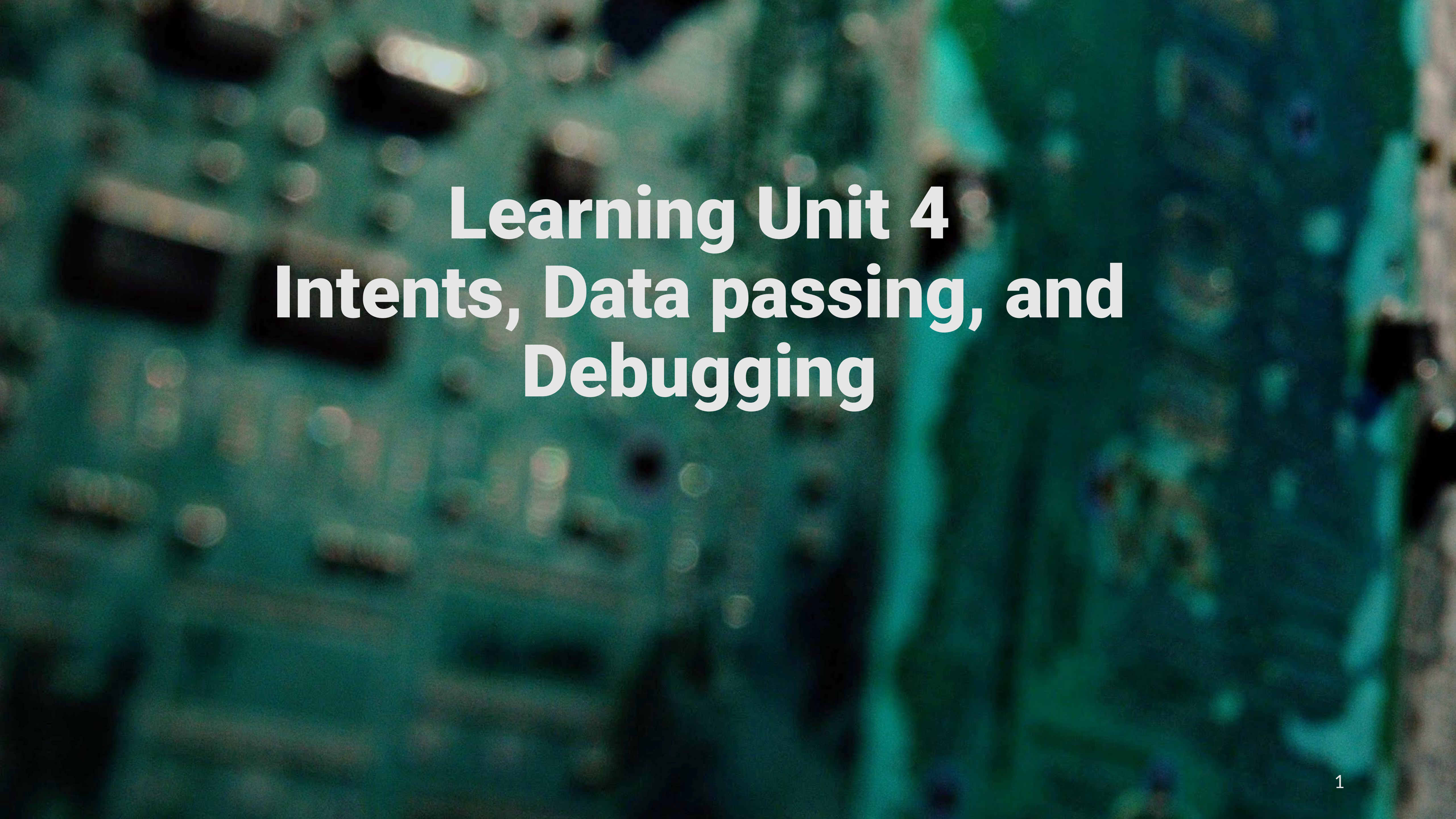
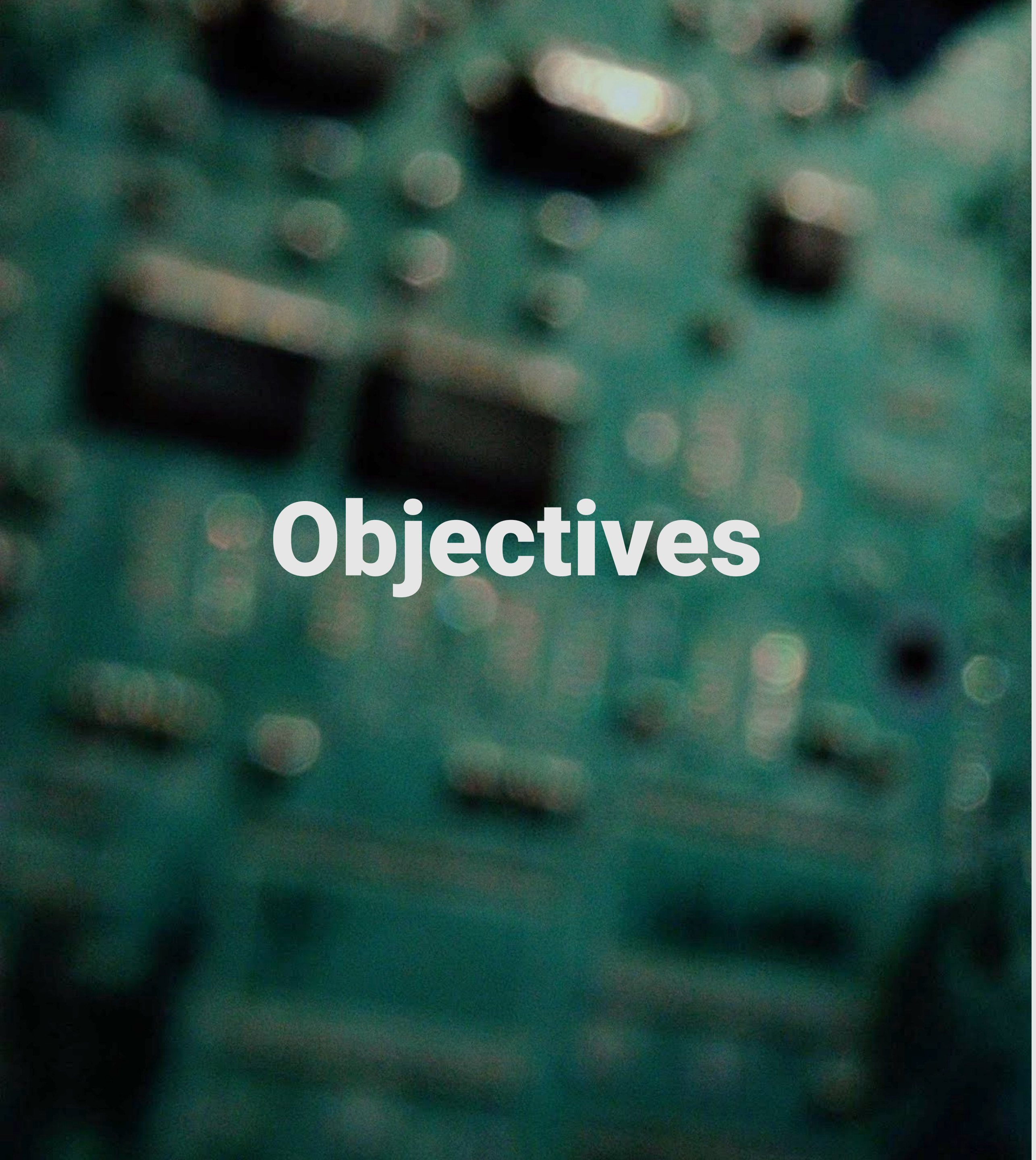


Learning Unit 4
Intents, Data passing, and Debugging



Objectives

- Understand and use Intents
- Passing data between activities
- Use the Android Studio debugger
- Accessibility
- Understand the importance of app performance
- Introducing fragments

Intents and Passing Data

Learning Unit 4

Intent class

- An Intent class is a structure that specifies:
    - Messaging objects
    - An operation to be performed
    - An event that has occurred
- Broadcast by one component
- Received by 0 or more components

Intent Types

There are two types of intents:

Explicit intents:

specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.

Implicit intents:

do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

How both intents work?

Explicit intent:

starts a named activity or service directly suing the Intent object.

Implicit intent:

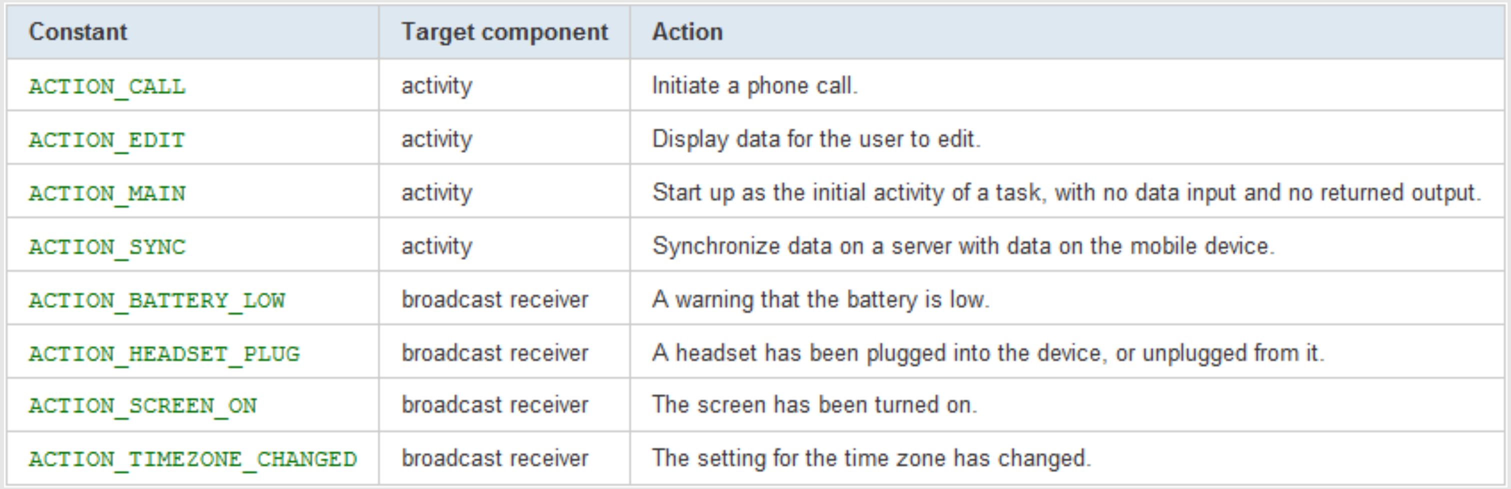
In this case, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device.

If the intent matches an intent filter, the system starts that component and delivers the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

What in the intents?

An Intent is a messaging object

that carries information that the Android system uses to determine which component to start.



Constant	Target component	Action
`ACTION_CALL`	activity	Initiate a phone call.
`ACTION_EDIT`	activity	Display data for the user to edit.
`ACTION_MAIN`	activity	Start up as the initial activity of a task, with no data input and no returned output.
`ACTION_SYNC`	activity	Synchronize data on a server with data on the mobile device.
`ACTION_BATTERY_LOW`	broadcast receiver	A warning that the battery is low.
`ACTION_HEADSET_PLUG`	broadcast receiver	A headset has been plugged into the device, or unplugged from it.
`ACTION_SCREEN_ON`	broadcast receiver	The screen has been turned on.
`ACTION_TIMEZONE_CHANGED`	broadcast receiver	The setting for the time zone has changed.

These are:

1- Component name (optional)

2- Action

A string that specifies the generic action to perform:

Inside the Intent: Actions

Setting the Intent action for viewing:

ACTION_VIEW

Use this action in an intent with startActivity() when you have some information that an activity can show to the user, such as a photo to view in a gallery app, or an address to view in a map app.

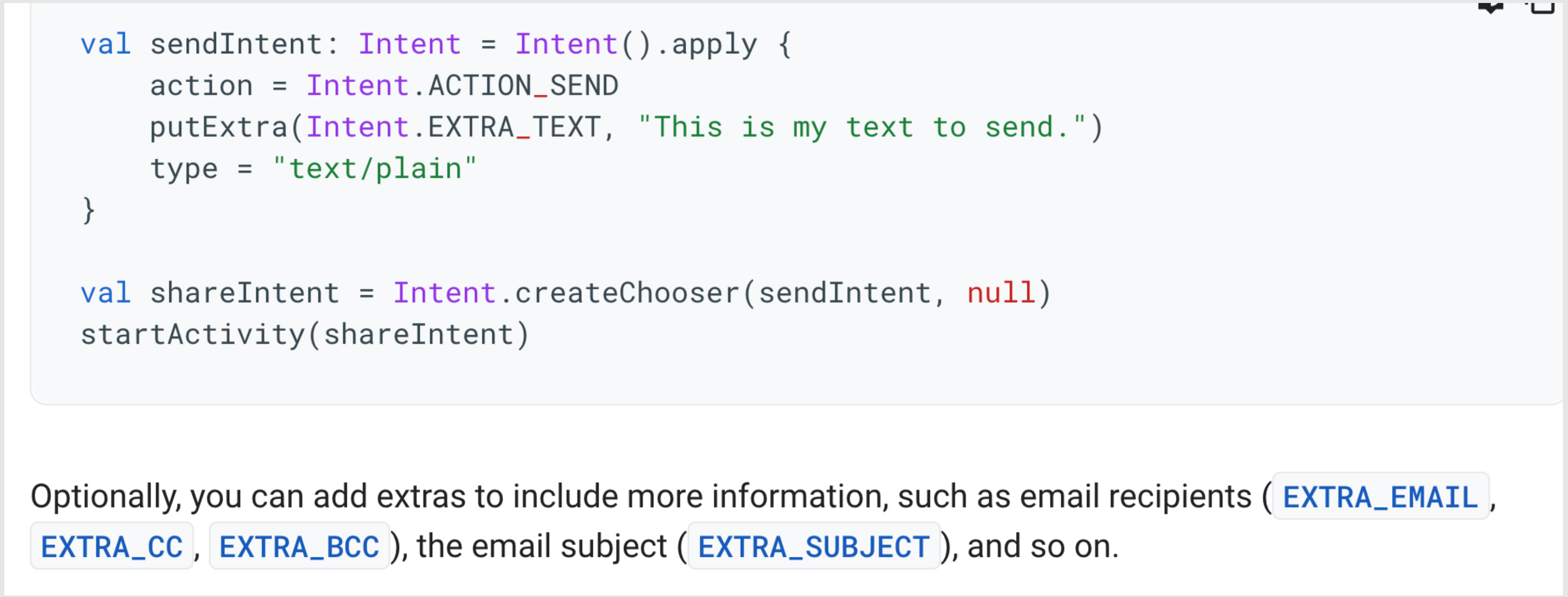
```
intent = Intent(Intent.ACTION_VIEW)
intent.setData(Uri.parse("https://www.google.com/"))
startActivity(intent)

intent= Intent(Intent.ACTION_VIEW, Uri.parse("https://www.google.com/"))
startActivity(intent)
```

More Actions (cont.)

Setting the Intent action for viewing:

ACTION_SEND

Also known as the "share" intent, you should use this in an intent with startActivity() when you have some data that the user can share through another app, such as an email app or social sharing app.



```
val sendIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, "This is my text to send.")
    type = "text/plain"
}

val shareIntent = Intent.createChooser(sendIntent, null)
startActivity(shareIntent)
```

Optionally, you can add extras to include more information, such as email recipients (`EXTRA_EMAIL`, `EXTRA_CC`, `EXTRA_BCC`), the email subject (`EXTRA_SUBJECT`), and so on.

Inside the Intent: Data (cont.)

Example of Data

Data for a map:

- geo:0,0?q=Farmingdale+State+College
  +New York+USA

And

- newInt = Intent(Intent.ACTION_VIEW);
- newInt.setData(Uri.parse("tel:+5160000000"));

Inside the Intent: Extra

Extras

Setting the Extra attribute

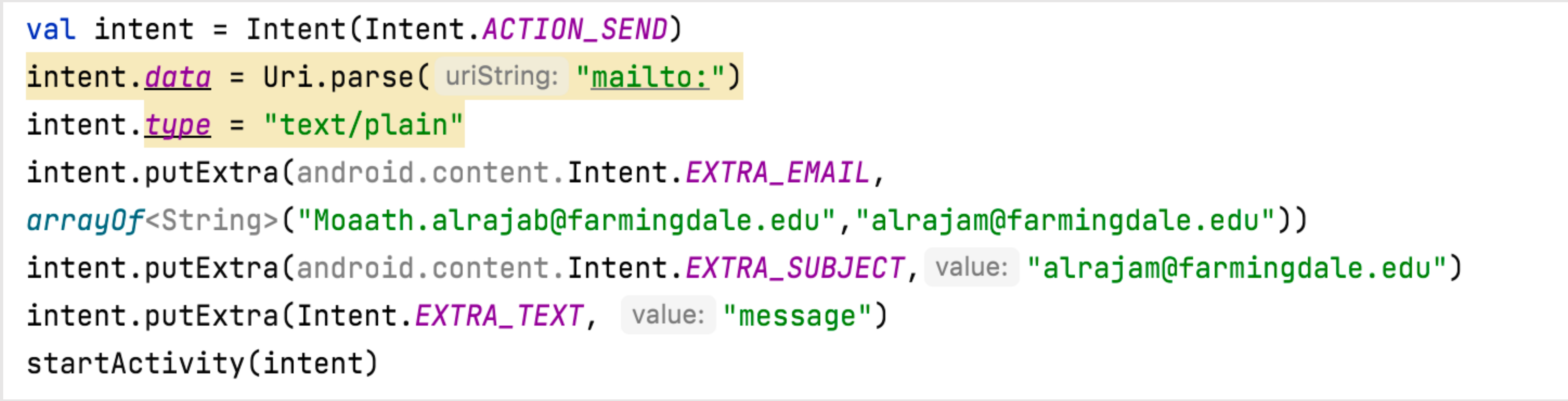
Several forms depending on data types,

e.g.,

- putExtra(String name, String value);
- putExtra(String name, float[] value);

Key-value pairs that carry additional information required to accomplish the requested action. Just as some actions use particular kinds of data URIs, some actions also use particular extras.

Inside the Intent: Extra (cont.)



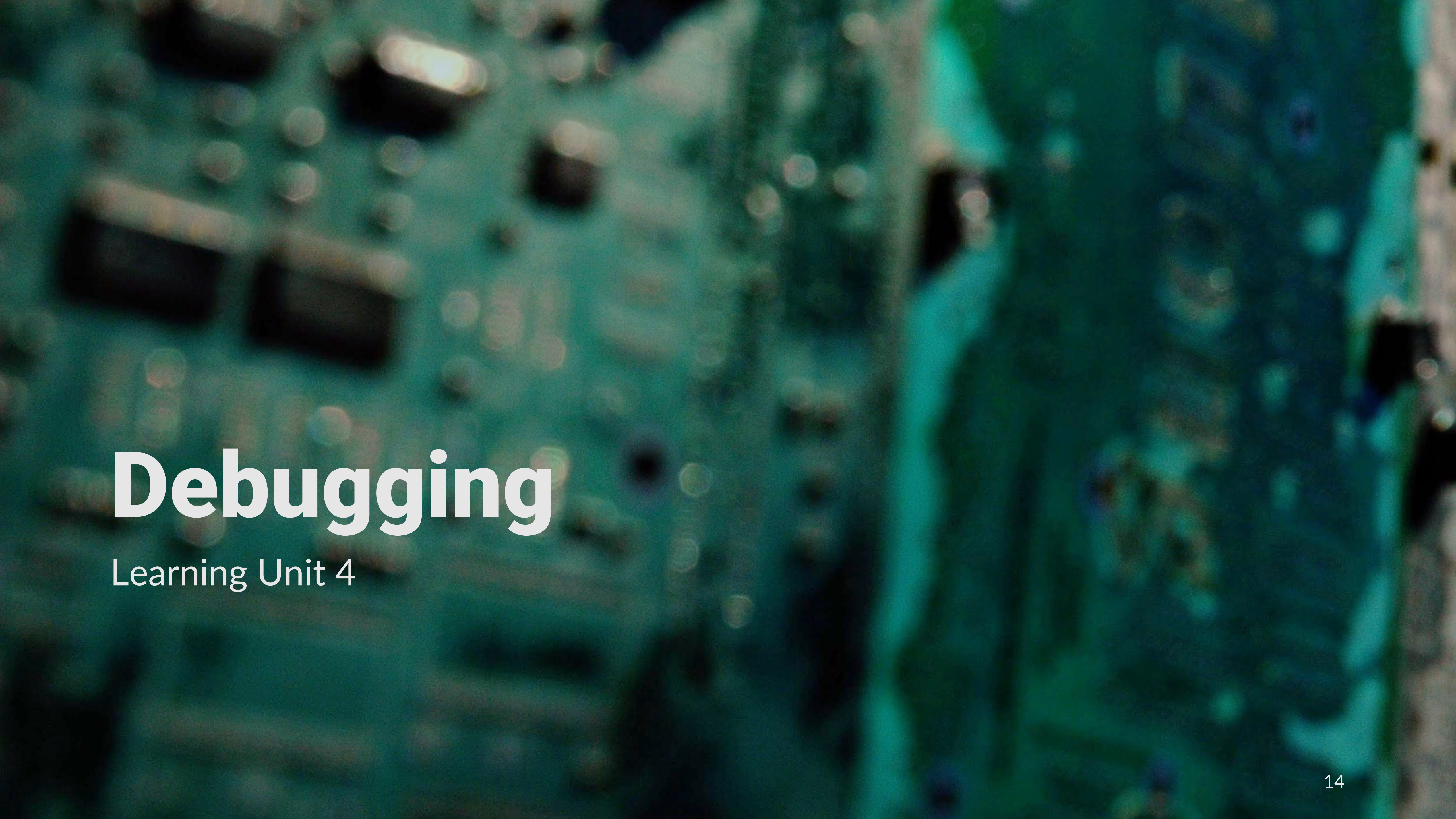
```
val intent = Intent(Intent.ACTION_SEND)
intent.data = Uri.parse( uriString: "mailto:")
intent.type = "text/plain"
intent.putExtra(android.content.Intent.EXTRA_EMAIL,
arrayOf<String>("Moaath.alrajab@farmingdale.edu","alrajam@farmingdale.edu"))
intent.putExtra(android.content.Intent.EXTRA_SUBJECT, value: "alrajam@farmingdale.edu")
intent.putExtra(Intent.EXTRA_TEXT,  value: "message")
startActivity(intent)
```

For example, when creating an intent to send an email with ACTION_SEND, you can specify the "to" recipient with the EXTRA_EMAIL key (above), and specify the "subject" with the EXTRA_SUBJECT key.

Intent Resolution

When the system receives an implicit intent to start an activity, it searches for the best activity for the intent by comparing the intent to **intent filters** based on three aspects:

- The intent action
- The intent data (both URI and data type)
- The intent category



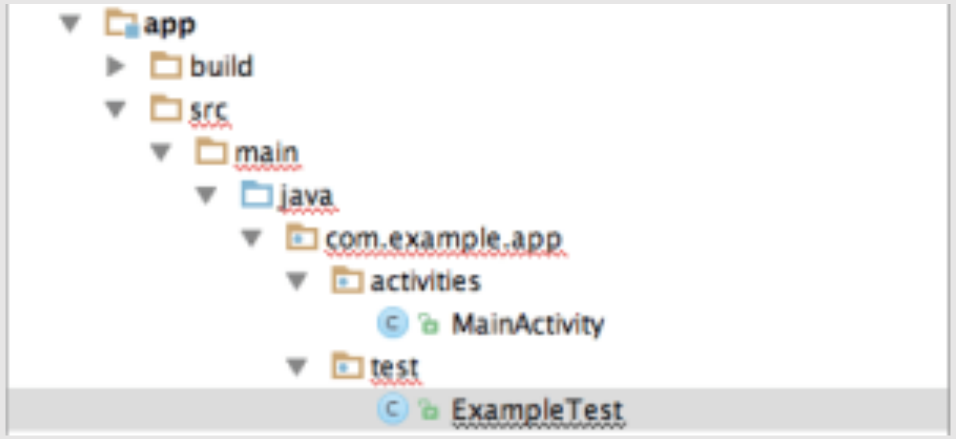
Debugging

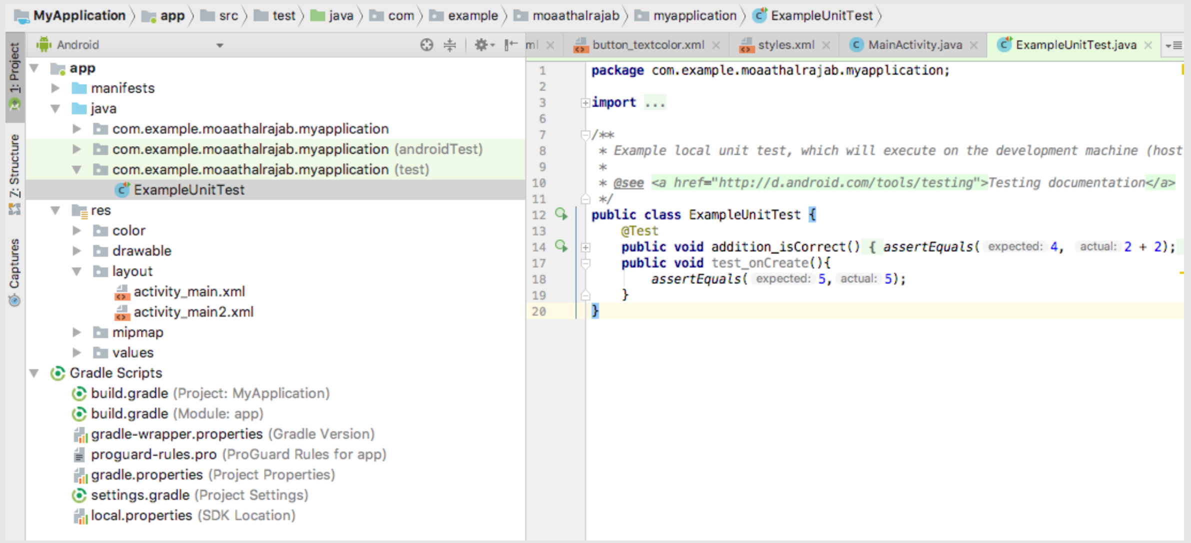
Learning Unit 4

Unit Testing With Android Studio

- Native to Android Studio?
  - Android Studio supports Android Unit Tests natively and you can enable them by setting a few options in your project configuration.

  - You have the option also to create your own tests by adding a new class.




MyApplication
app
src
test
java
com
example
moaathalrajab
myapplication
ExampleUnitTest
Android
button_textcolor.xml
styles.xml
MainActivity.java
ExampleUnitTest.java
1: Project
2: Structure
Captures
app
manifests
java
com.example.moaathalrajab.myapplication
com.example.moaathalrajab.myapplication (androidTest)
com.example.moaathalrajab.myapplication (test)
ExampleUnitTest
res
color
drawable
layout
activity_main.xml
activity_main2.xml
mipmap
values
Gradle Scripts
build.gradle (Project: MyApplication)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

```java
package com.example.moaathalrajab.myapplication;

import ...

/**
 * Example local unit test, which will execute on the development machine (host
 *
 * @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
public class ExampleUnitTest {
    @Test
    public void addition_isCorrect() { assertEquals( expected: 4,  actual: 2 + 2);
    public void test_onCreate(){
        assertEquals( expected: 5, actual: 5);
    }
}
```

Writing tests to run on the Android device

- Instrumentation tests

  Instrumented unit tests are unit tests that run on Android devices and emulators instead of running on the Java virtual machine.

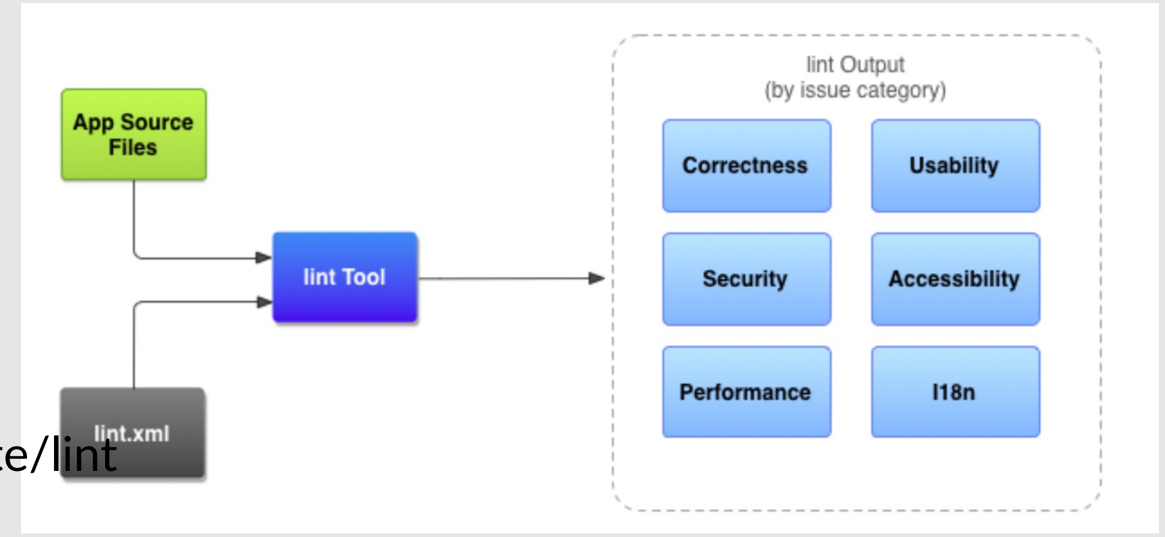
- These tests have access to the real device and its resources and are useful to unit test functionality which cannot be easily mocked by mocking frameworks.
  - An example is a test which validates a Parcelable implementation.

- An instrumentation-based test class allows you to send key events (or touch events) to the application under test.

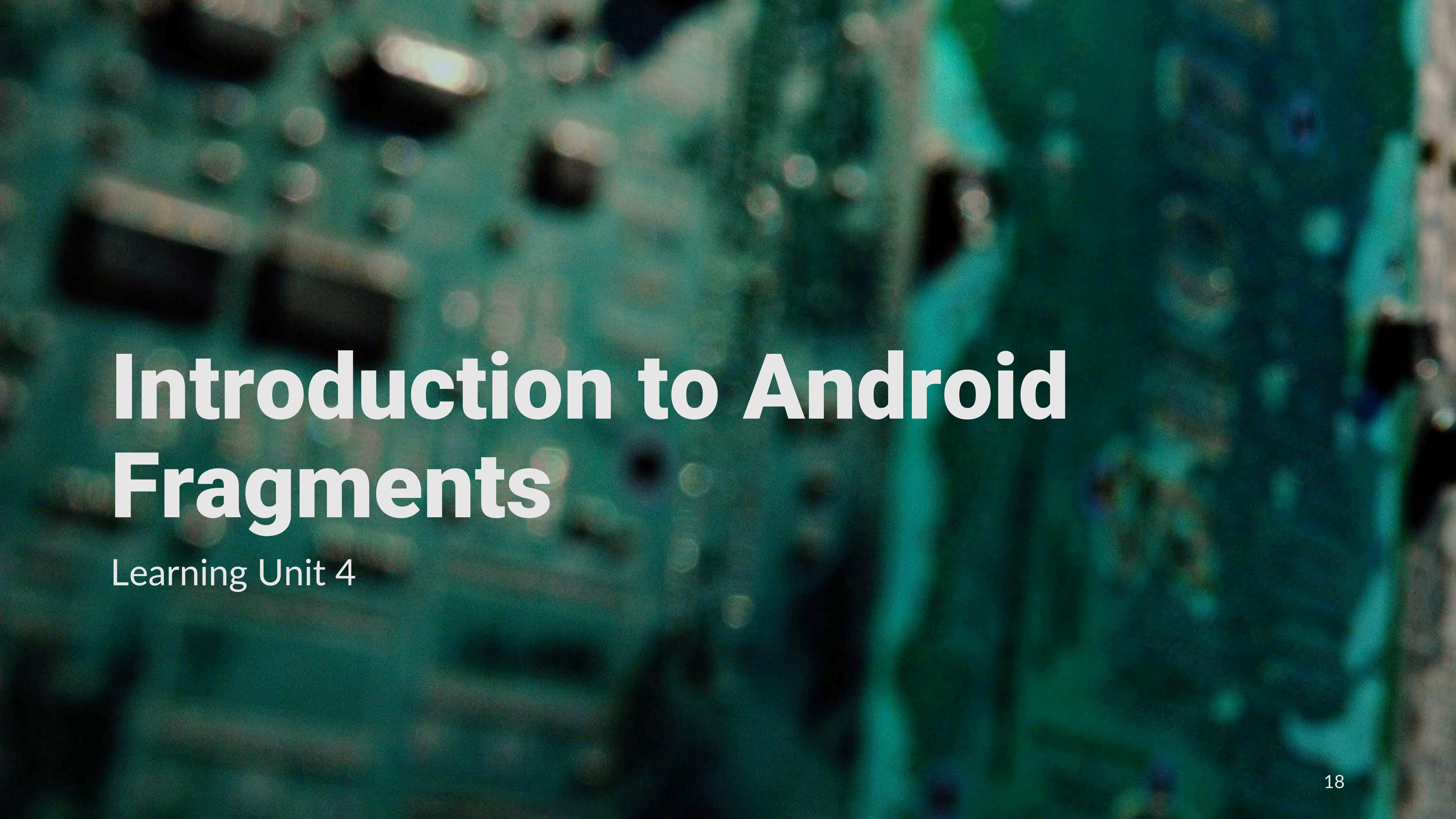
Improve your code with lint checks

You can manually run configured lint and other IDE inspections by selecting **Code** > **Inspect code**. The results of the inspection appear in the **Inspection Results** window.

- Set the inspection scope and profile
- Select the files you want to analyze (inspection scope) and the inspections you want to run (inspection profile), as follows:
- In the **Android** view, open your project and select the project, a folder, or a file that you want to analyze.
- From the menu bar, select **Analyze > Inspect Code**.
- In the **Specify Inspection Scope** dialog, review the settings.
- Under **Inspection profile**, keep the default profile (**Project Default**).
- Click **OK** to run the inspection.



Reference: https://developer.android.com/studio/write/lint



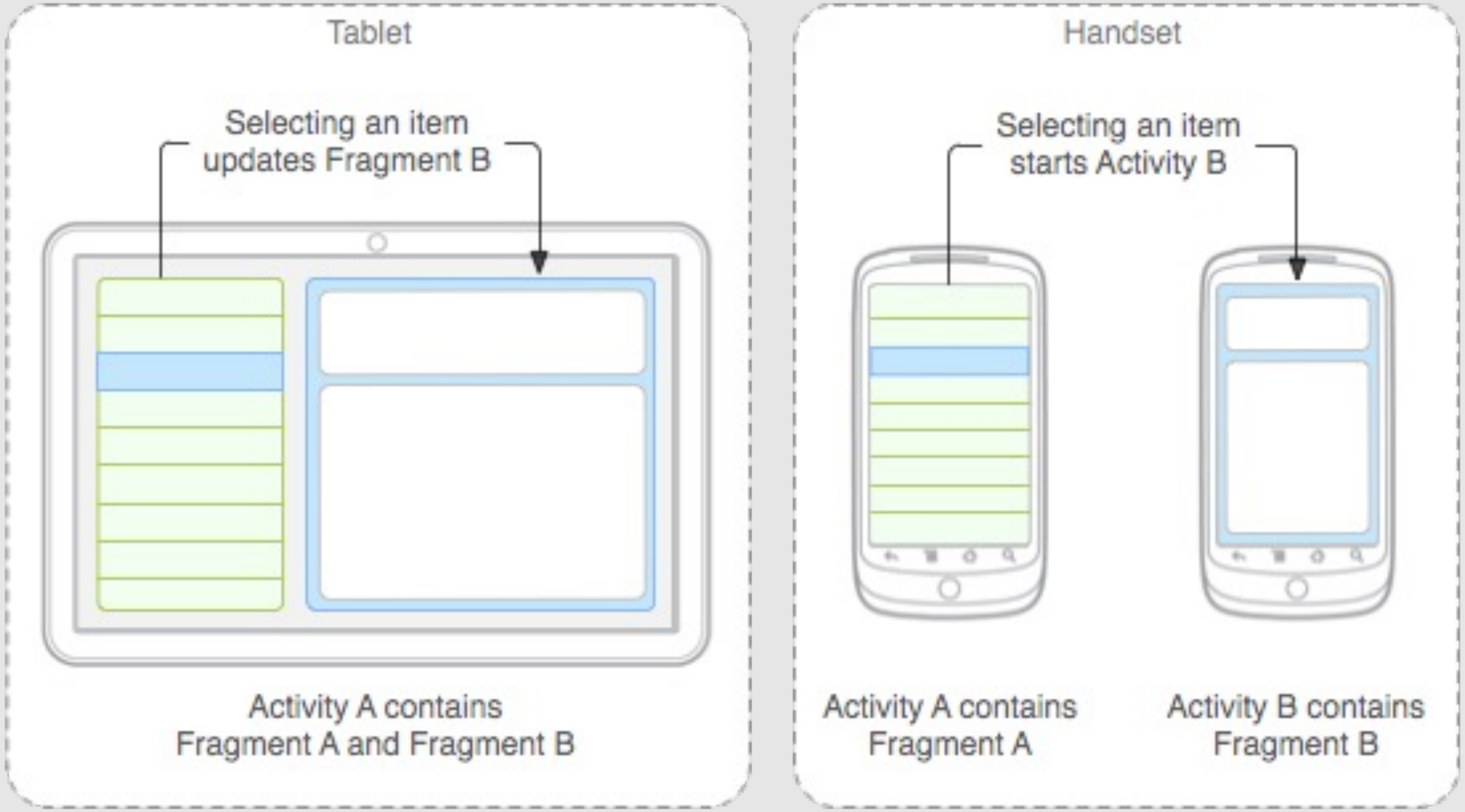
Introduction to Android Fragments

Learning Unit 4

Fragments

- Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets.
- Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components.
- Fragments allow such designs without the need for you to manage complex changes to the view hierarchy.

Fragments: what are they?

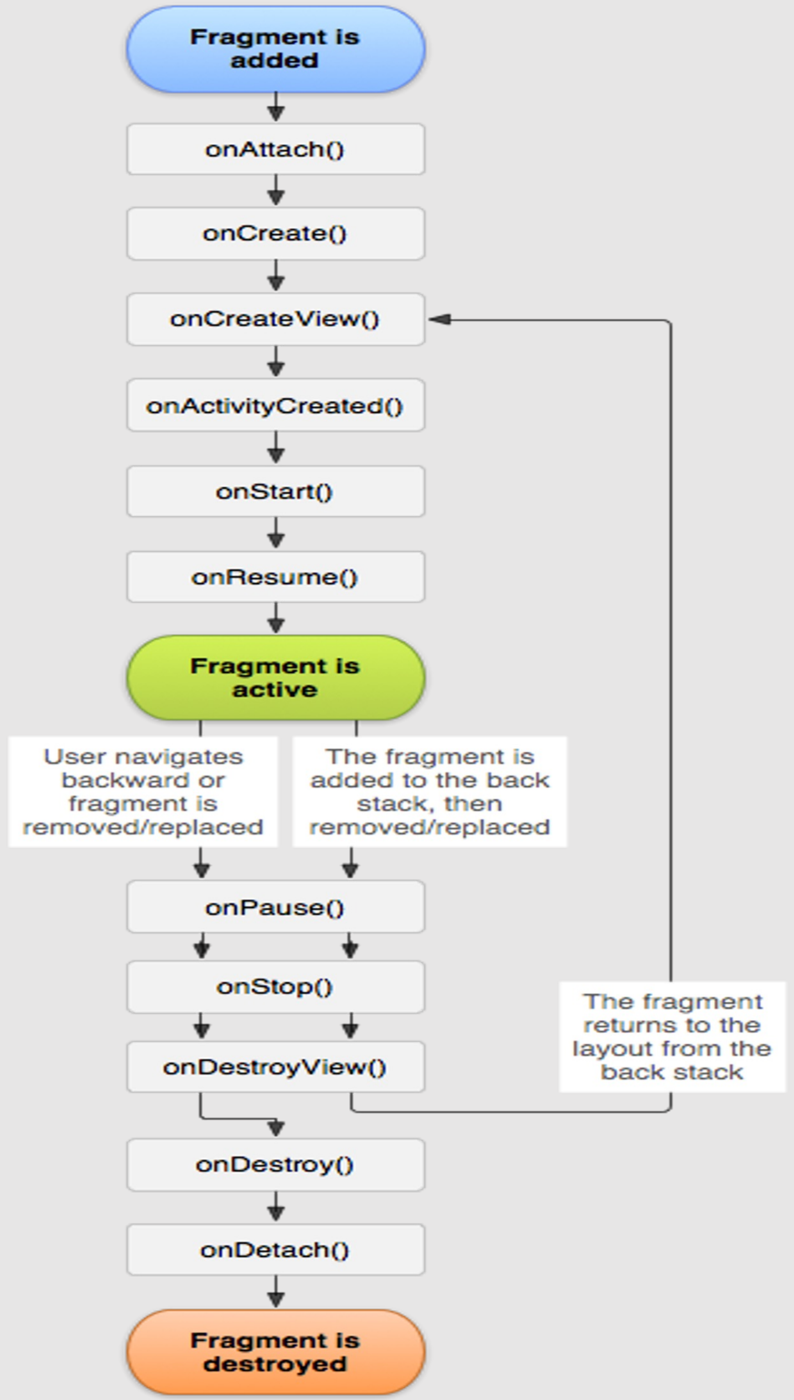

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Fragments: what are they?

- A fragment is used to render and manage a single UI view..
- Fragments are parts of the application's UI or behavior that are integrated into activities -- a fragment needs an activity to run.
- Consider each fragment as a **lightweight mini-activity**.
- Fragments are **flexible and reusable components** defined by the programmer.
- Fragments present a **consistent UI across different apps** and devices; fragments are flexible in adapting the user experience across these different environments (phone, tablet); therefore, allowing to group UI views.
- Each fragment has **its own lifecycle** and associated UI (for example, a fragment for the user to enter profile information, another one to render maps).
- How many fragments do you need? Rule of thumb is one per different type of data displayed (e.g., preferences, profile, maps, list of exercises) or user input.

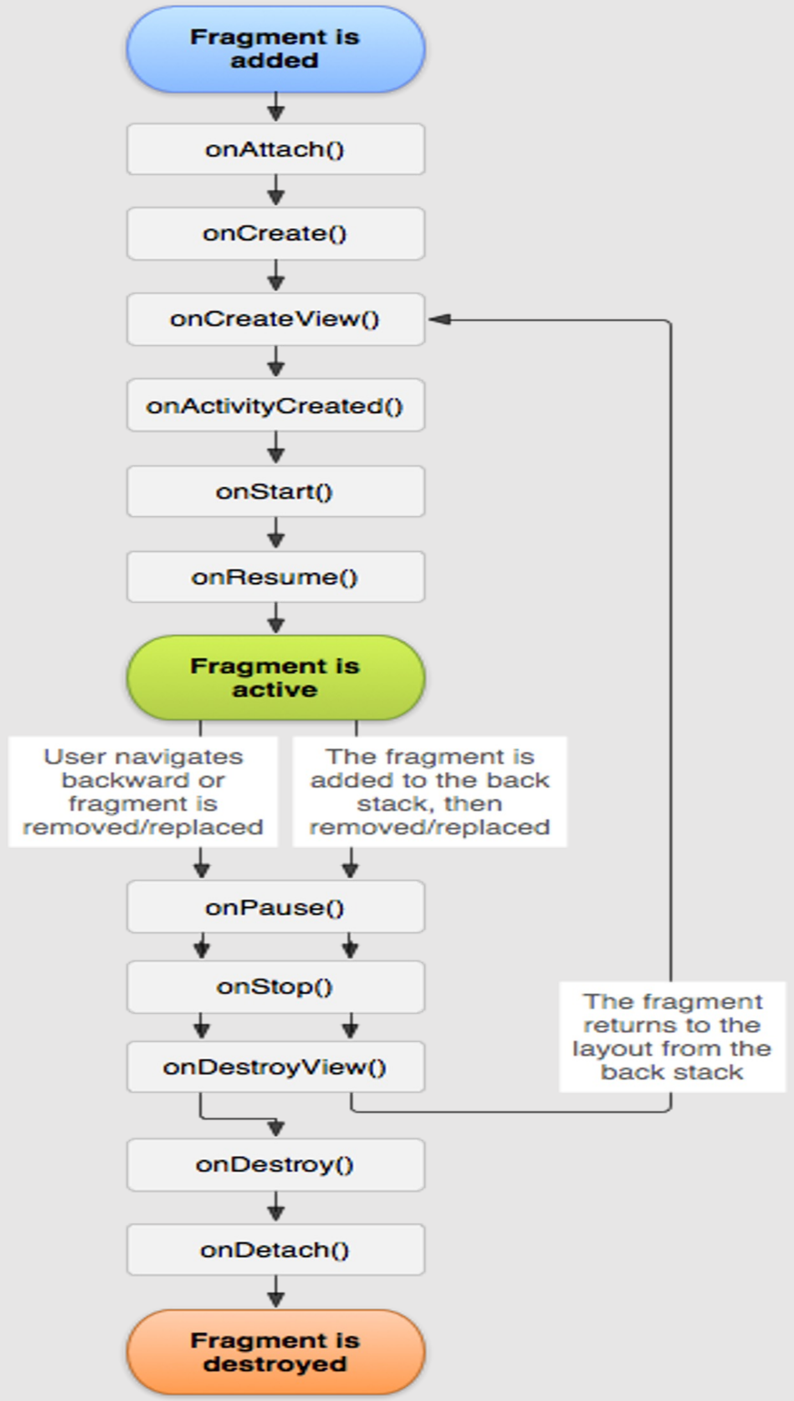
**Treat a fragment as a mini-activity with its
own independent UI and Lifecycle (to manage its internal state).**

Fragments should be considered as part of the controller layer in the MVC architecture



Fragment Lifecycle

- onAttach() is called when the Fragment is attached to its parent activity
- **onCreate()** is called to initially create the fragment
- onCreateView() is called to created its user interface -- it inflates the fragment UI
- onActivityCreated() is called once the parent activity and the fragment UI are created.
- **onStart()** is called the start of the visible lifetime, any UI changes can be applied before the fragment goes visible
- **onResume()** is called at the start of the active lifetime such as resuming any paused UI updates needed by the fragment that were suspended when it became inactive.



Fragment Lifecycle

- **onPause()** called at the end of the active lifetime; persist all edits or state changes, suspend UI updates, threads, or CPU intensive processes that don't need to be updated when the Activity isn't the active foreground activity
- **onSaveInstanceState()** called to save UI state changes at the end of the active lifecycle
- void onStop() called at the end of the visible lifetime
- onDestroyView() called when the fragment's view is detached
- **onDestroy()** called at the end of the full lifetime
- onDetach() called when the Fragment has

What have we learnt this unit?

- The use of Intents
- Passing data between activities
- Use the Android Studio debugger
- Accessibility
- Understand the importance of app performance
- Introducing fragments