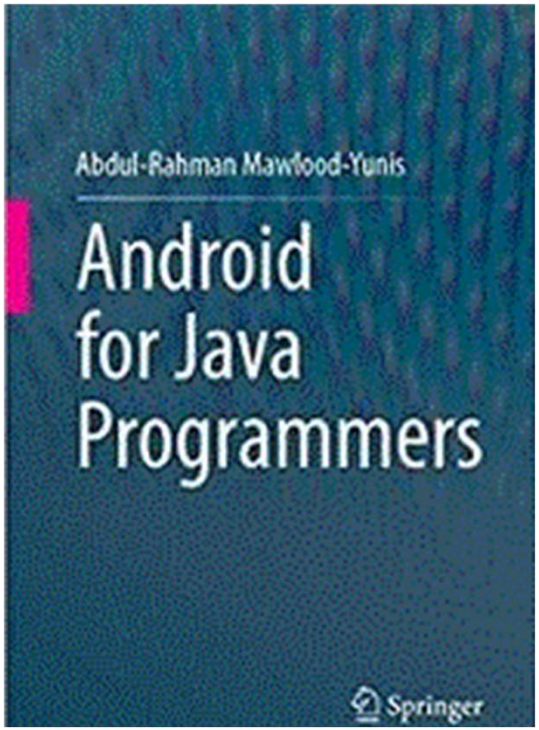
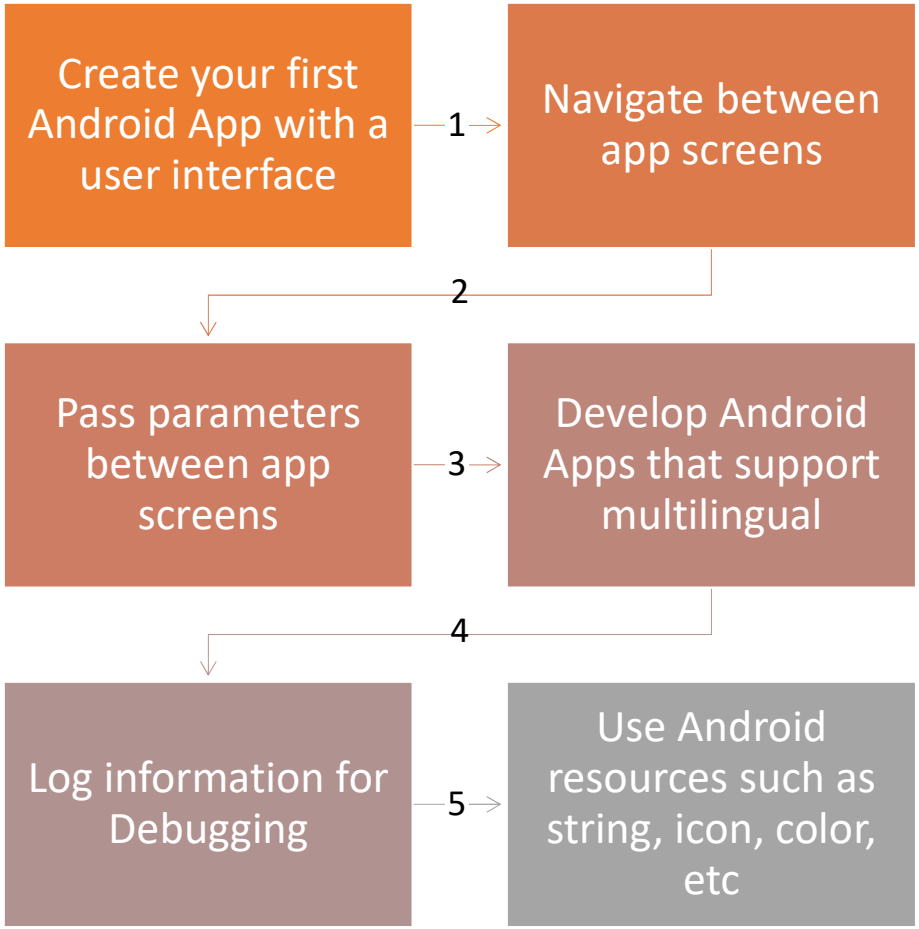
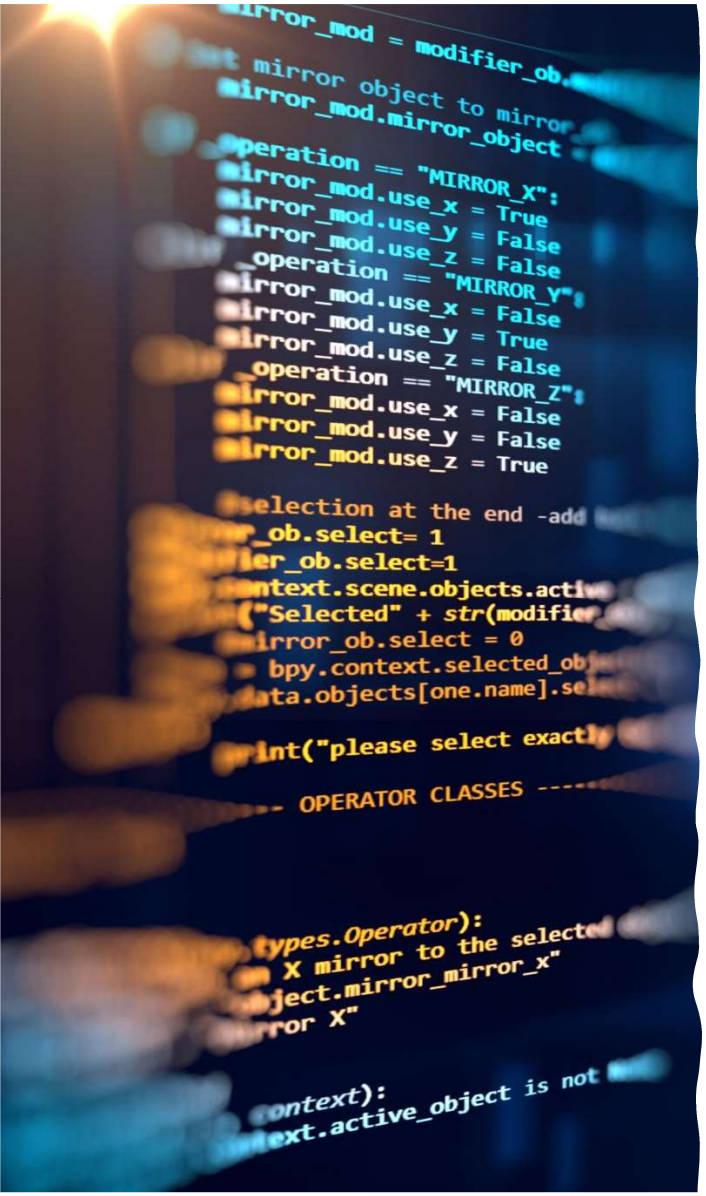
Chapter 03: Your First Android Application



Learning outcome





Check out the demo project

Download the demo app, **MyFirstApplication**.zip, specifically developed to go with this lesson.

How to run the code: unzip the code in a folder of your choice, then in Android Studio click File->import->Existing Android code into the workspace

3.1 Introduction

You may have noticed that there were several new ideas in the HelloWorld app presented in chapter two that you usually don't see when running the HelloWorld Java programs

These include using activities, intents, and bundle objects, starting the program from the manifest file, creating a User Interface for the app using an XML file, and using resources such as layouts and strings

In the HelloWorld app, there is a clear separation between the UI implementation, which is written in the XML files, and the program, the Java code

In this lesson, we will consider a more complex program with multiple UI elements, activities, and intent objects

Part 1: Introduction to Android App Development

In this part, we cover a general introduction to Android app development

We describe Android application characteristics and introduce major concepts and components like Android activities, R Files, Android context, AndroidManifest.xml files, opening android projects in Android Studio, and cleaning Android project builds

3.2 Early Android development

At first, Android development was done in Eclipse with some plugins to enable Android app development

Coding was done in Java and then installed on Android-enabled devices or emulators powered with the Java virtual machine for running

Now, Android Studio is used for app development and there is more than one emulator and physical device for running

In the past, the Jack toolchain needed to be enabled to support Java 1.8 language features.

Now, Jack is deprecated, and native Java 1.8 features are supported as part of the build system

The compilation, packaging, and installation of Android apps are different from typical Java applications.

These steps were described in chapter two, and you are encouraged to review them.

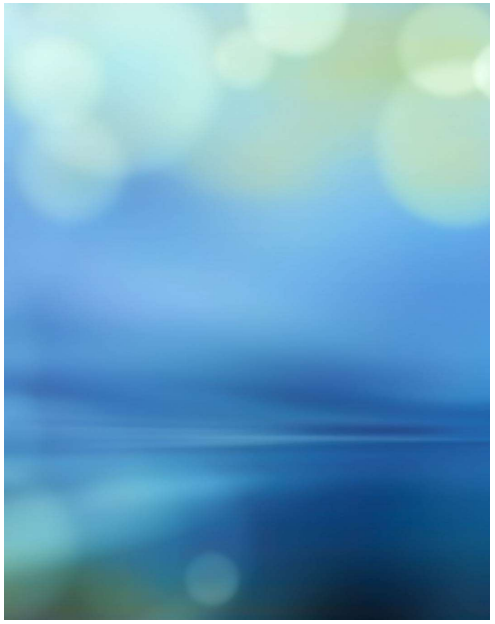
3.3 Android Versions

Android versioning reveals the fact that the Android API is continually growing and improving, and this growth is expected to continue for many years to come.

Both the users and the developer can feel these improvements. However, their views on the change are different.

As a developer, you express these changes as going from API level 25 to API 27 to API 28, etc. However, the user is not aware of these terms. Instead, they see it as going from Nougat to Oreo, to Pie, etc.

Only recently Google started to name its APIs as Android 10, Android 11, etc.




List of Android Versions and Initial Stable Release Dates
Android 1.0
September 23, 2008
1.5 - Cupcake
April 27, 2009
1.6 - Donut
September 15, 2009
2.0/2.1 - Éclair
October 26, 2009
2.2 - Froyo
May 20, 2010
2.3 - Gingerbread
December 6, 2010
3.0 - Honeycomb
February 22, 2011
4.0 - Ice Cream Sandwich
October 18, 2011
4.1/4.3 - Jelly Bean
July 9, 2012
4.4 - KitKat
October 31, 2013
5.0 - Lollipop
November 12, 2014
6.0 - Marshmallow
October 5, 2015
7.0 - Nougat
August 22, 2016
8.0 - Oreo
August 21, 2017
9.0 - Pie
August 6, 2018
Android 10
September 3, 2019
Android 11
September 8, 2020
Android 12
October 17, 2021
CellPhoneDeal

3.3 Android Versions

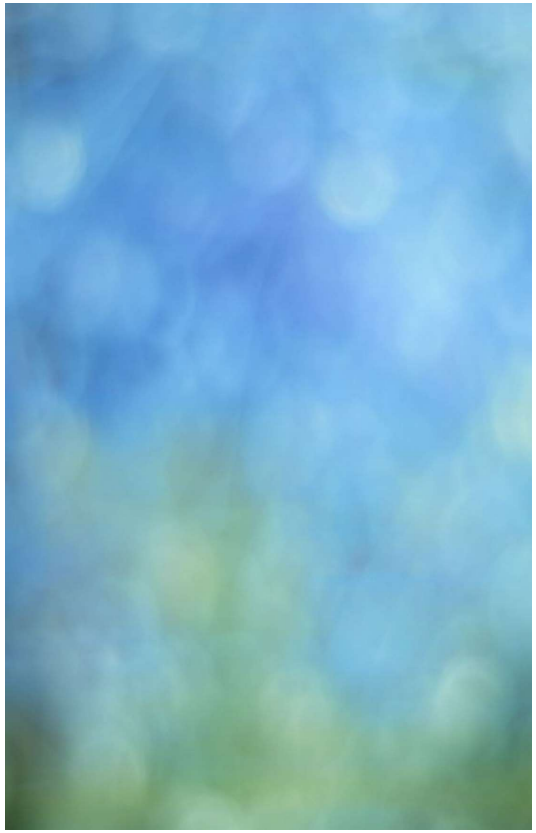
Examples of the Android API levels, or versions, and the features they support.

AsyncTask was introduced in API 3.

Download Manager was introduced in API 9.

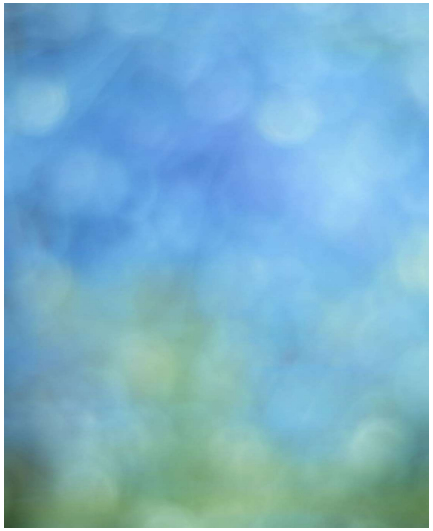
GridLayout was introduced in API 14 and is depreciated in the current API version.

NetworkServiceDiscovery was introduced in API 16.



3.3 Android Versions

Examples of the Android API levels, or versions, and the features they support.

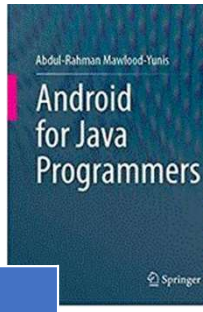


- You can find out when a feature has been introduced by checking the top right-hand side of the Android reference page for the feature

- For example, the reference page for the AsyncTask feature shows that it has been introduced in level 3

https://developer.android.com/reference/android/os/AsyncTask

Examples of New Android versions

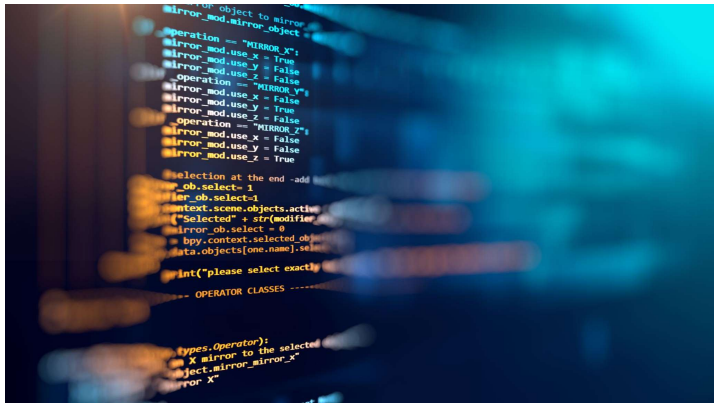


Codename	Version	Released	API Level
Android 12	12	Sept 2021	31
Android 11	11	Sept 2020	30
Android 10	10	Sept 2019	29
Pie	9.0	Aug 2018	28
Oreo	8.0 -8.1	Sept 2017	26 - 27
Nougat	7.0 - 7.1	Sept 2016	24 - 25
Marshmallow	6.0 - 6.0.1	Oct 2015	23

Older Android versions

Codename	Version	Released	API Level
Honeycomb	3.0 - 3.2.6	Feb 2011	11 - 13
Ice Cream Sandwich	4.0 - 4.0.4	Oct 2011	14 - 15
Jelly Bean	4.1 - 4.3.1	July 2012	16 - 18
KitKat	4.4 - 4.4.4	Oct 2013	19 - 20
Lollipop	5.0 - 5.1.1	Nov 2014	21 - 22

- For earlier versions before 2011 see: Android History and Platform Versions

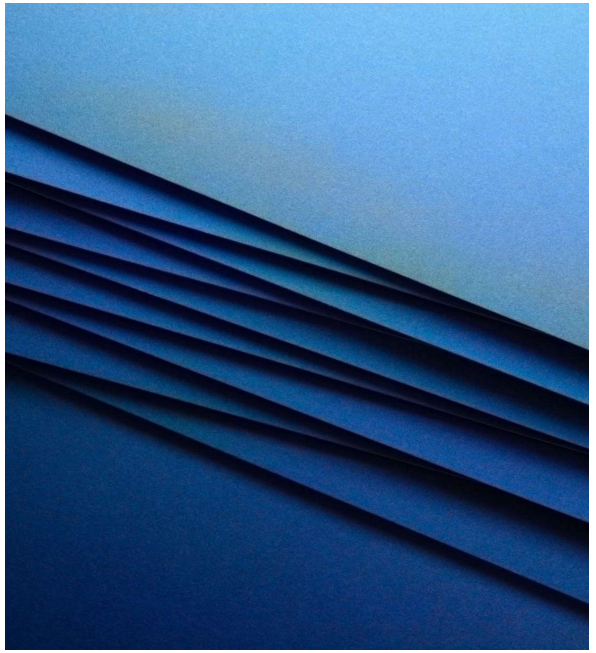


App Development

Chapter 03: Your First Android Application

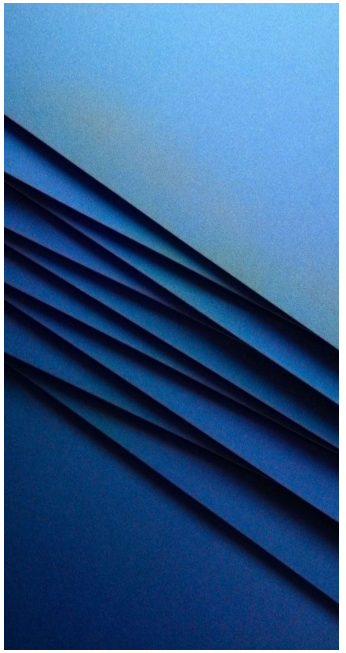
What is an Android app?

- One or more interactive screens
- Written using Java Programming Language and XML
- Uses the Android Software Development Kit (SDK) (tool
- Uses Android libraries and Android Application Framework
- Executed by Android Runtime Virtual machine (ART)



The Android SDK

- The Android SDK is a collection of tools, libraries, and resources provided by Google for developing Android applications.
- It includes the Android Studio IDE (Integrated Development Environment), which is the official development environment for Android app development.
- The SDK also includes essential tools like the Android Debug Bridge (ADB) for debugging and testing on Android devices, as well as emulators for simulating Android devices on your computer.
- Android SDK provides the necessary APIs and documentation to access device features, such as the camera, GPS, sensors, and more.



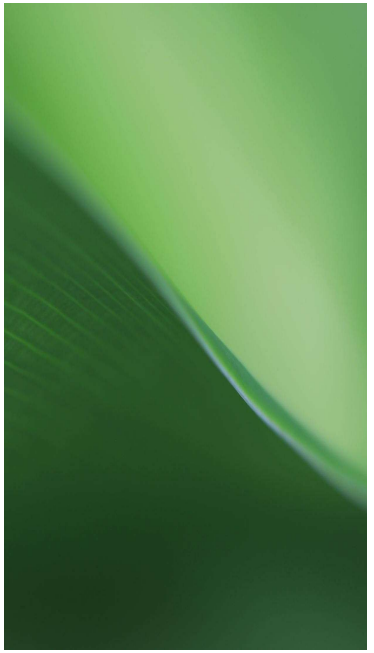
Frameworks

Android framework refers to the underlying structure and architecture of the Android operating system. It includes the core components of Android, such as Activity, Fragment, Service, Content Provider, and Broadcast Receiver.

Developers use the Android framework to create the structure of their applications and define how different components interact with each other.

The Android framework enforces the Model-View-Controller (MVC) or Model-View-ViewModel (MVVM) architectural patterns, depending on your preference, to organize the code and user interface of your app.

Android framework also handles the lifecycle management of activities and services, user interface rendering, and communication between different app components.



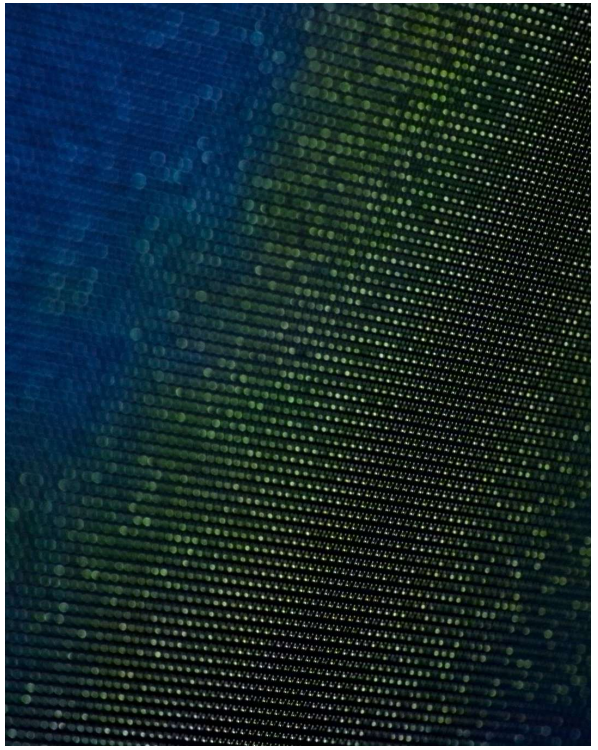
Challenges of Android development

Multiple screen sizes and resolutions

Performance: make your apps responsive and smooth

Security: keep source code and user data safe

Compatibility: run well on older platform versions



3.4 Android Applications Characteristic

Useful Android apps are typically made of multiple interactive screens

Two of the main technologies used to create Android apps are Java and XML files

Java files are used to define the app's behaviour or business logic

Android devices come in different kinds of screen sizes and resolutions that you need to account for when developing apps

Android is evolving continuously, and your app needs to be backward compatible as well

3.4 App building blocks

Resources: layouts, images, strings, colors as XML and media files

Components: activities, services, Broadcast Receivers, Content Providers

Helper classes as Java code

Manifest file: Information about app for the runtime, and an entry to point to the app, same as the main method in Java applications

Build configuration: APK versions in Gradle config files

3.5 Android Activity

Defines a screen, or a window, in your app, just like an HTML file of a website or a JFrame in a Java Swing application

Activities for Android are the same as classes in Java and more

Activities, as the name implies, are activities and are created for a reason, for example, to display information, enable transactions, process business forms, etc

They enable user interactions such as clicking a button or entering text and can start other activities in the same or different apps

setContentView is a method in the Activity class that takes a layout object, an XML file declared inside the layout folder, as an input to draw the content of the Activity

3.6 R File

When compiling an application, the XML files are parsed to generate the R.java class

For each folder in your res directory, there will be a static final class with the same name in the R.java file

The XML files inside the app folders are also represented in the R.java file

In the early versions of the Android framework, you were able to locate and open the R.java file by searching the generated folder of your app

New versions of the Android framework transform bytecode into machine code during app installation using ahead-of-time compilation and android runtime , and there is no need to save R.java files anymore to be saved into your device

3.7 Android Context

Context is an interface to the app

If you need the context of the entire application and not just the current Activity, use getApplicaitonContext() instead of getContext().

You can use context to refer to components of your app, for example, your Activity

The Activity class has a method called getContext , and when called, it returns the reference to the current Activity

For example, within an Activity, you can pass ***this*** in places where Context is needed

Activity is a subclass of the Context class. . Hence, when it is appropriate, you can use Activity in places where Context is expected.

3.8 Application Manifest Files

The manifest file is the entry point to the app, it acts as the main method in Java.

Every application must have a manifest file

Things that are declared inside the manifest file include:

- The parent-child relationship declaration between activities
- The app's interaction behavior with other apps on the same device
- Interaction with components within the same app

3.8 Application Manifest Files

You declare application permissions such as

- defining permission to use the internet, read files, make calls, etc inside the manifest file.

In the manifest file, you declare Activities that are a part of your application. The Activity that launches the app becomes the first screen of your app.

The information above is needed for your build tools, Android OS, and Google play store

3.9 Opening Android Project in Android Studio

You should have downloaded the MyFirstApplicatoin.zip by now. If you did not, please do so.

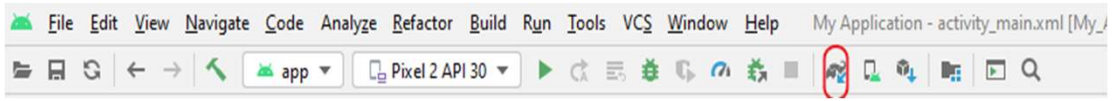
Unzip the MyFirstApplication.zip. You will have the MyFirstApplication project.

In Android Studio: Click *File* → *Open* if you have already opened a project. Otherwise, Click Open *an existing Android Studio project* at the "Welcome to Android Studio" screen.

Find the MyFirstApplication directory and click "Choose". The project should be opened and ready to run.

3.10 Cleaning Android Project Builds

You might need to synchronize your code with Gradle files



You can do that by clicking on the Sync Project with Gradle button on the **File menu**, or just click on the sync button on the menu bar

Cleaning builds and synchronizing your project with Gradle is oftentimes a helpful way to clear out project errors

Part 2: Create Your First Mobile App

In this part of the lesson, we will describe how to create your first mobile app

We use our MyFirstApplication example to describe the steps

This chapter, along with the previous one, is all that it takes to create your first app

3.11 Your App Specification

Before creating an app, you need to know its specification, i.e., what you are going to build

The MyFirstApplication app is made of two activities, or two screens:

A. The **MainActivity** which starts the second Activity, called **MessageDisplayActivity**.

B. Use an **Intent** object as a passing parameter between the two Activities

3.11 Your App Specification

When the user:

types a message into the input box, or EditText object, on the first Activity and

clicks send button, the message is bundled up in an intent object and is passed to the second Activity

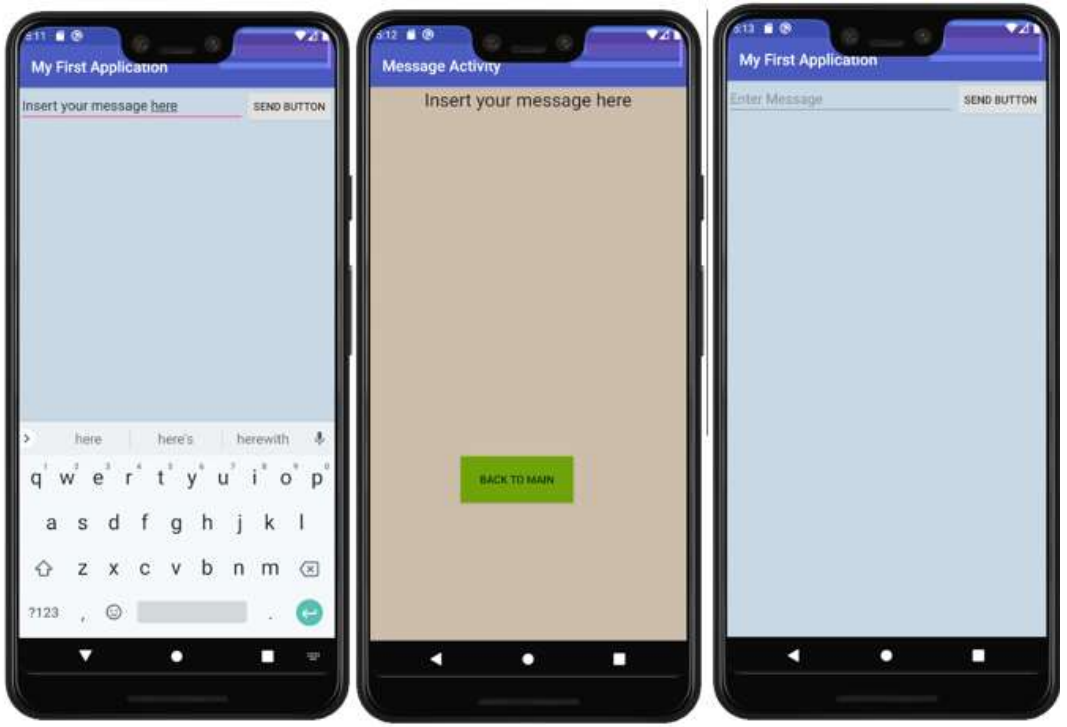
The DisplayMessageActivity or the second activity starts and displays the message

3.11
Your App Specification

The figures are the screenshots of the app.

The first screen from the left is rendered by the MainActivity, the user enters a message and clicks the send button.

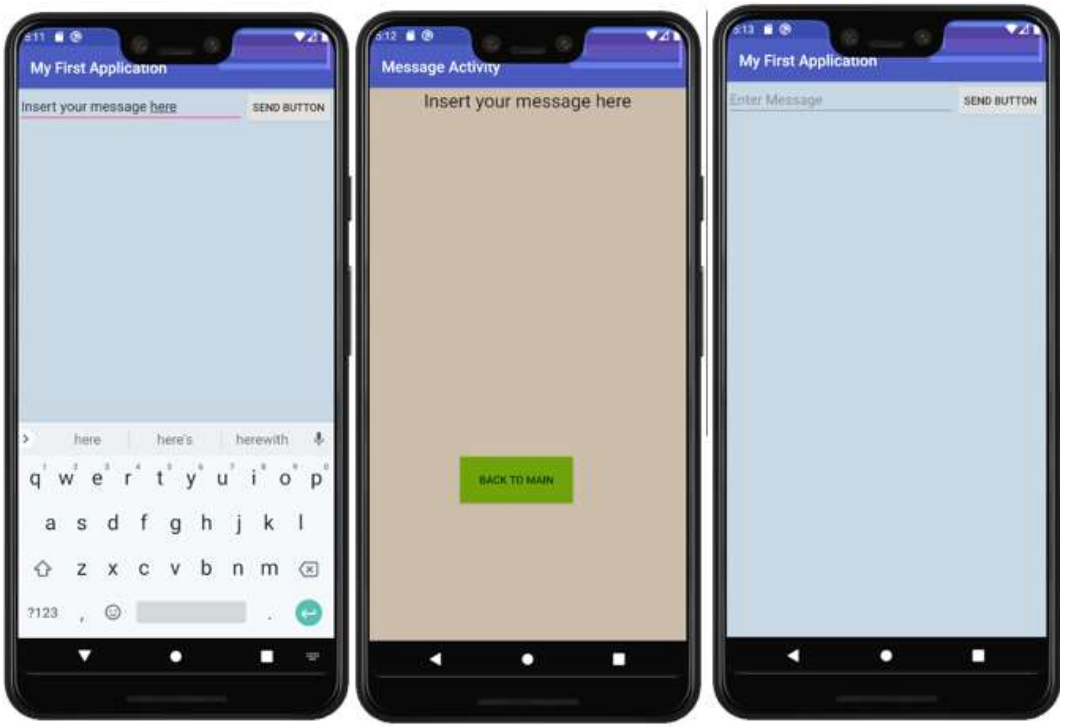
The DisplayMessageActivity starts to display the message (the middle screen).



3.11
Your App Specification

You can navigate back to the MainActivity view using the Back to Main button from the DisplayMessageActivity View.

To see the application code, open the *app/java* folder and look at the MainActivity.java code.



3.12 Create Activity Layout

Earlier in this lesson, we described how to create an Android Activity, specifically, how to create an EmptyActivity

When an Activity is created using the Android Studio, a window pops up and you will be asked if you want to create a layout file for it

If you checked the *Generate a Layout File* box, a layout file is created. The location of the file would be at *app/res/layout/.*

The XML code snippet Listed in next slide is the layout for the MainActivity when you create a project following the default steps for creating an Empty View Activity

The layout for the MainActivity

activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android=http://schemas.android.com/apk/res/android

    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools=http://schemas.android.com/tools

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

</androidx.constraintlayout.widget.ConstraintLayout>
```

3.12
Create Activity Layout

- For Empty View Activity, there are NO entries for widget elements between the opening and closing ConstraintLayout element, i.e., the layout is empty.
- To make it more interesting, we need to add some widgets to it. For example, we may need to add a **Button**, **TextView**, and labels to make it more appealing.
- **EditText** is a class with many properties in the Android SDK.
- In Android, you can use an XML file to declare and initialize classes

tools:context=".DisplayMessageActivity"

3.12.1
Adding an EditText Field to the Layout File

- In the code snippet below, the EditText class is declared and four of its properties are initialization using tags

- <EditText

```
android:id="@+id/edit_message"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="@string/edit_message" />
```

- The name of the class becomes an XML element and class fields are represented using XML element properties.

3.12.2 Add String to Resources File

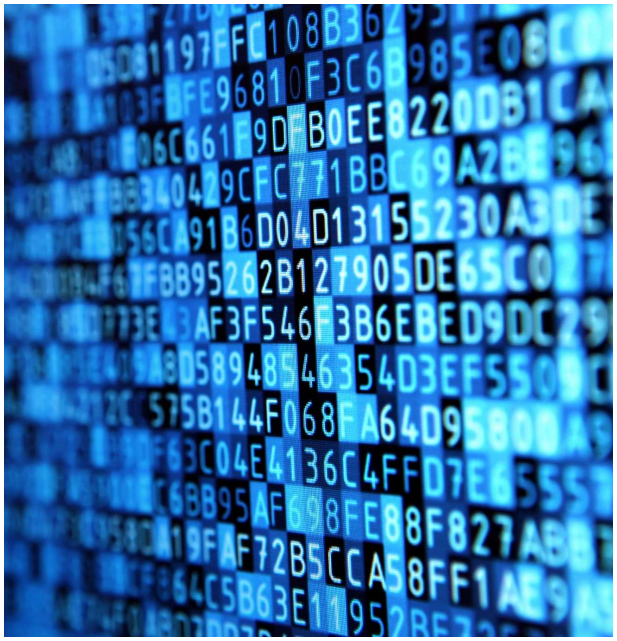
- By default, an Android project has a **strings.xml** resource file located *at **app/res/values/*** directory inside the project structure
- Open the file insert a new string key named "insert_message" and set the value to *"Enter a message."* See the example below:

<string name="insert_message"> Enter Message </string>

In general :

<string name =“key”> value</string> or

<string name =“key” value=“value” />



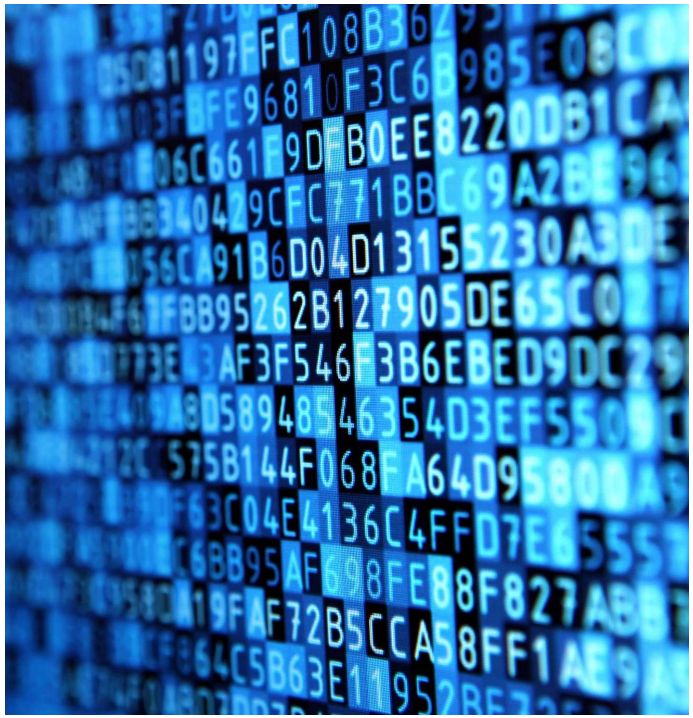
3.12.2 Add String to Resources File

It is important to give your components and variables proper names. Having too many buttons and other variables in your layout without clear names might result in incorrect referencing, which in turn could result in total application failure

All strings of your app will be added to the strings.xml file.

add another string key, "send_button", for the button you'll soon add called "button_send"

```
<string name="send_button"> Send Button </string>
```



3.12.2 Add String to Resources File

After adding strings to the strings.xml file, the content of your final string.xml file should be like the one listed below:

```
<resources>
    <string name="app_name">My First Application</string>
    <string name="insert_message"> Enter Message </string>
    <string name="send_button"> Send Button </string>
    <string name="action_settings">Settings</string>
    <string name="title_activity_display_message">My Message</string>
</resources>
```

3.12.3 Adding Components to the Layout File

- Like the EditText class, Button is a class in the Android SDK.
- Using XML syntax, the Button class is declared below and three of its properties are initialized.

```
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/button_send" />
```

- To put both EditText and Button on the same line, **add the Button class description to the layout file and put it under the EditText declaration**.

3.12.3 Adding Components to the Layout File

To format the EditText input box, add the following attributes to an already declared EditText element in the layout file.

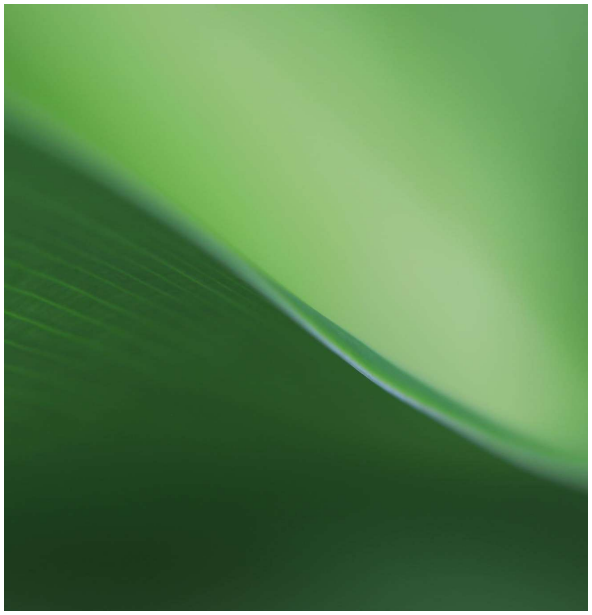
```
android:layout_weight="1"
android:layout_width="0dp"
```

3.12.3 Adding Components to the Layout File

You may think that changing EditText width to 0dp will make it disappear from the screen.

However, this does not happen here because we are setting EditText *layout_weight* to 1 which will stretch the EditText width to the width of the device screen minus the width of other components that are on the same horizontal line.

Now, your end-result activity_main.xml layout file from the res/layout/ directory should be as shown



3.12.3 Adding Components to the Layout File

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
android:background="@color/lightGreen">

<EditText
      android:id="@+id/edit_message"
       android:layout_weight="1"
       android:layout_width="0dp"
       android:layout_height="wrap_content"
       android:hint="@string/insert_message"/>
   <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/send_button"
       android:onClick="sendMessage"/>
</LinearLayout>
```

3.13 Invoke Message on Activity

Add **android:onClick="sendMessage"** to the Button definition.

By doing so, you are saying that when button_send is pressed, a method called sendMessage() inside the MaintActivity.java class should be executed.

The new definition for the Button would be as follows:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

3.13 Invoke Message on Activity

Open the MainActivity.java class inside your project and add sendMessage() definition to the class.

sendMessage is a regular Java class method, and you can add it anywhere in the MainActivity class where the method declaration is acceptable.

The signature for the methods declared inside the layout file **must** abide by certain constraints.

- The **return type** of the method must be **void**,
- the method should have only **one parameter**, and the type of the parameter is **View**.
- An example of the sendMessage() method implementation is given below.

```
/* This method is called when the user clicks the send button */
public void sendMessage(View view) {
  // write your code in response to the button click
}
```

3.14 Intent Class

The MainActivity code for the MyFirstApplication example uses the Intent object when starting a second Activity

The Android SDK has a class called Intent, and objects of the Intent class can be passed between Activities

Intent objects are message objects, or links between Activities

3.14 Intent Class

- You can think of Intent objects as something you would like to do, i.e., your intention of an operation to be carried out as you request the Android system to do something for you

- For example, if you use Intent with the *StartActivity* method (more on StartActivity later) it will launch another Activity

- Intent can also be used to:
- initiate downloading a file
- Broadcast a message to other components of your app or
- Broadcast a message to other apps installed on your device

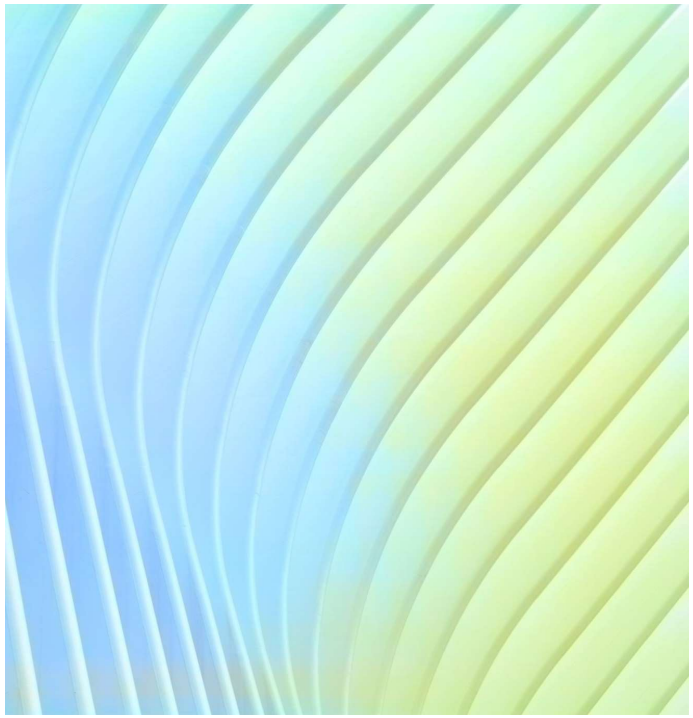
3.14. Explicit Intent

- There are two types of intent, **explicit intent, and implicit intent**

- Explicit Intents are used to activate other components such as activities explicitly

- You use explicit Intent when you know what app components you would like to trigger

- Using explicit Intent enables you to pass objects between activities as well.



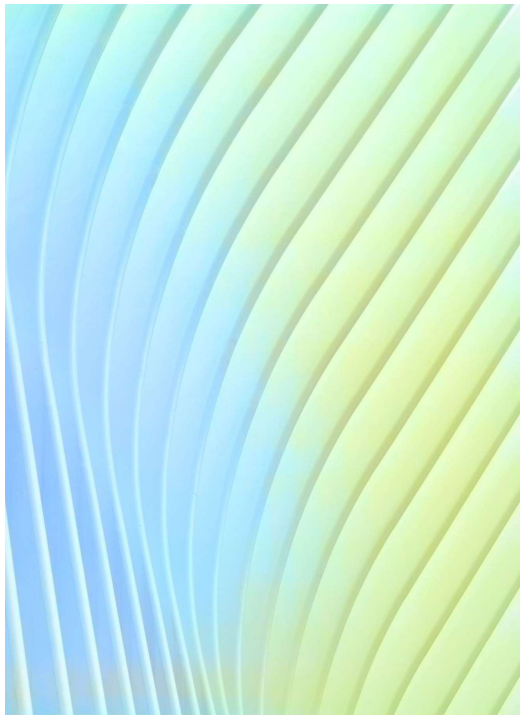
3.14. Explicit Intent

- The code snippet below shows how an explicit Intent object is created

- Intent intent = new Intent (this, **DisplayMessageActivity.class**);

  intent.putExtra("key", "value");

  startActivity(intent);

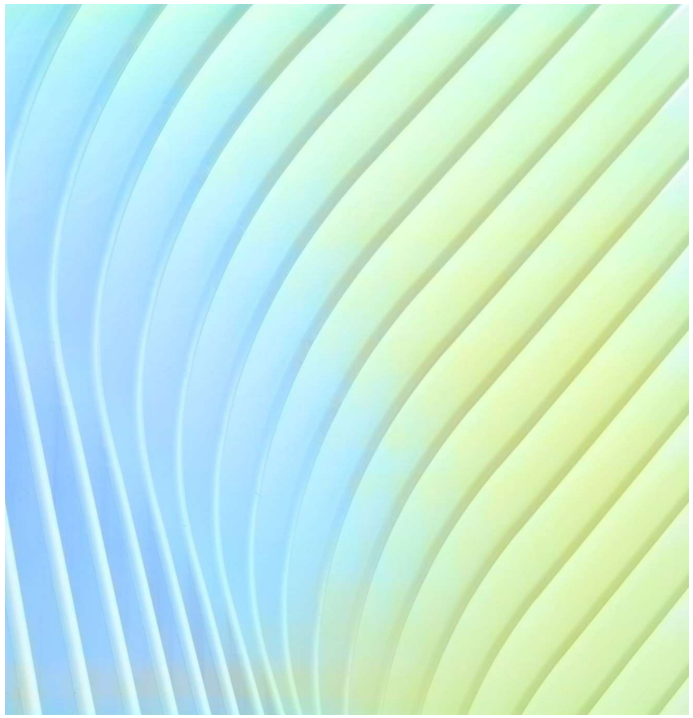


3.14. Explicit Intent

- It is an explicit Intent because you pass explicitly the name of the second Activity, i.e., the MessageDisplayActivity.class, to the intent constructor

- you can also pass data to the next screen using the putExtra (key, value) method from the Intent class

- The value of data can be a String, other Java primitive types, or **objects**



3.14. Explicit Intent

```
public class MainActivity extends AppCompatActivity {

 public final static String EXTRA_MESSAGE =
          "code.android.abdulrahman.com.myfirstapplication.MESSAGE";
   private static final String TAG = "MyActivity";
    @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
   }
   /** Called when the user clicks the Send button */
   public void sendMessage(View view) {
      // Do something in response to button
      Intent intent = new Intent (this, DisplayMessageActivity.class);
      EditText editText = findViewById(R.id.edit_message);
      String message = editText.getText().toString();
      intent.putExtra(EXTRA_MESSAGE, message);
      Log.d(TAG, "Intent fired ");
      startActivity(intent);
   }
}
```

```
public class User implements Serializable {
    private static final long servialVersoin = 987456321741852L;
    private String fName;
    private String lName;
    private int idNumber;

    public User(String fname, String lname, int anId) {
        this.fName = fname;
        this.lName = lname;
        this.idNumber = anId;
    }

    public String getFirstName() {
        return fName;
    }

    public String getLasttName() {
        return lName;
    }

    public int getID() {
        return idNumber;
    }

}
```

Passing object to second activity

3.14. Explicit Intent

```
User user = new User("AR", "Yunis", 123) ;
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    intent.putExtra("user", user);
    Log.d(TAG, "Intent fired ");
    startActivity(intent); ;
```

3.14. Explicit Intent

```
User usr = (User)getIntent().getSerializableExtra("user") ;

TextView objectView = findViewById(R.id.message);

objectView.setText("first name is " + usr.getFirstName() + ", " +

    "Last name is " + usr.getLasttName() +
     " and id is " + usr.getID());
```

3.14. Implicit Intent

There is another type of Intent, implicit Intent

- When implicit Intent is used, the Activity that handles the Intent request is not specified

- The Android system finds one or more Activities that can handle the request

- To create an implicit Intent, you need two pieces of information:
  - The ***action*** that needs to be carried out and
  - The ***data/information*** for the action

3.14. Implicit Intent

- The Intent class has multiple actions, or constants, that can be called. For example,
- *Action_View,*
- *Action_Daial,*
- *Action_Answer,*
- *Action_Call,*
- *Action_Pick,*
- *Action_search,*
- *Action_ send,* and many more.

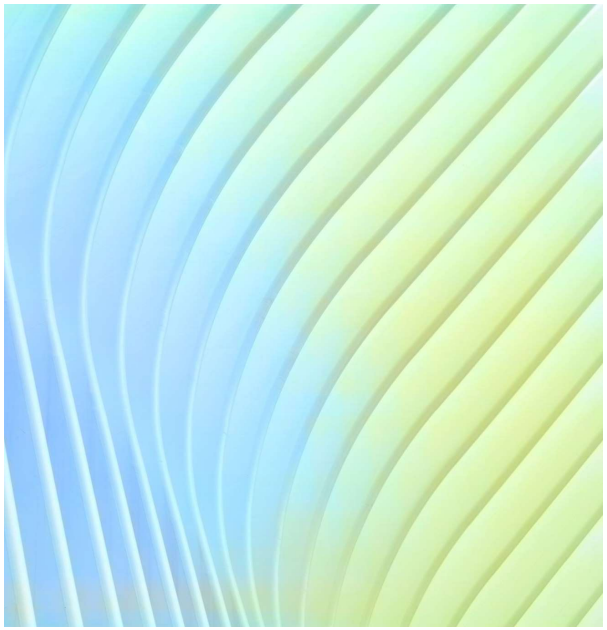
- https://developer.android.com/reference/android/content/Intent#standard-activity-actions

3.14. Implicit Intent

The code below shows how you can use **Action_view** to view a webpage and **Action_Dial** to dial a number using implicit Intent.

```
Uri uri = Uri.parse("http://www.wlu.ca");
Intent anIntent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(anIntent );

Uri uri = Uri.parse("tel:8005551234");
Intent telIntent = new Intent(Intent.ACTION_DIAL, uri);
startActivity(telIntent );
```



3.14. Implicit Intent

When using implicit Intent objects, the Uri object is used in Intent constructor

Uri is a Java class from the net package.

Uri | Android Developers

It has several methods including the *parse*,
fromfile, and
fromParts method that return Uri object

Both previous examples use the parse method to form Uri objects

3.14. Implicit Intent

You need to check the Android reference page to find the proper string format for the Uri parse method and select the proper action for the Uri

The **Intent** has other useful methods. Examples of such methods include PutExtra, putExtras, getIntExtra, etc.

Intent is an important class with many constants, methods, and constructors, and is an essential component for creating apps.

https://guides.codepath.com/android/Using-Intents-to-Create-Flows

3.15 Using StartActivity

Every Activity class inherits the startActivity methods from the Context class

The Activity class, which is a subclass of Context class, has multiple versions of the startActivity method (https://developer.android.com/reference/android/app/Activity)

To start another Activity from your current Activity, you need to call the startActivity method and pass an Intent object to the method

The Intent object should include the name of the second Activity you would like to launch

After adding the startActivity statement to the sendMessage method, the complete code for the sendMessage method that can be invoked by the Send button looks as shown below

3.15 Using StartActivity

- ```
  /** Called when the user clicks the Send button */
  public void sendMessage(View view) {
      // Do something in response to button
      Intent intent = new Intent(this, DisplayMessageActivity.class);
      EditText editText = (EditText) findViewById(R.id.edit_message);
      String message = editText.getText().toString();
      intent.putExtra(EXTRA_MESSAGE, message);
      Log.d(TAG, "Intent fired ");
      startActivity(intent);
  }
  ```
  ```

3.16 Create Second Activity

- To create a new activity using Android Studio do this: Right-click on the app or java folder, find New -> Activity -> Empty Activity, and click to open the create Activity dialog window

3.16 R.String and strings.xml file

Looking at the **onCreate** method for the message display activity, you may notice that there is a call to the **setTtile** method

setTitle is a public method in the activity class that you can use whenever you want to set a title for your activity page

The title string is declared in the strings.xml file and it gets retrieved using **R.string.title**.

*// **public static abstract int message***

This is possible because when compiling an application, the strings.xml file is parsed to the strings.java class which will have a static final data field for each entry in the XML file

You should be able to see it here: app\build\intermediates\runtime_symbol_list\debug directory

3.17 Project Manifest Update

If you open the project's manifest file, you will see that an XML entry for the new activity, DisplayMessageActivity, has been added to the manifest file

The **intent-filter** is part of the MainActivity definition, and both **action** and **category** names are used to specify that MainActivity is the starting, or the launcher, activity

If your app is made of multiple Activities, all Activities must be defined inside the **application element**, then they become the application's child element

Application attributes are used to define properties that can be applied to all components of the app

For example, you set **supportRtl** to true when you want your application to support right-to-left layouts

3.17 Project Manifest Update

- In the code snippet below, the intent-filter is part of the MainActivity definition, and both action and category names are used to specify that MainActivity is the starting, or the launcher, activity.
- If you decide to make, for example, MessageDisplayActivity the starting screen of your app, you need to put the intent-filer element inside the MessageDisplayActivity element

```
<activity android:name=".MainActivity">

    <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category  android:name="android.intent.category.LAUNCHER" />

        </intent-filter>

</activity>
```

3.17.2 Application Attributes

Application attributes are used to define properties that can be applied to all components of the app.

Some of these attributes, such as **icon**, **label**, and **theme**, have default values that can be changed by individual components.

Others, such as the **debuggable** attribute, are applied to the whole application, and once set, it cannot be changed by an individual application component.

Some of the attributes that you can set are listed on the next slide

You don't need to learn about all these attributes from the get-go. Instead, based on the needs of the applications you are building, you will study one or more of these attributes

3.17.2 Application Attributes

```
<application android:allowTaskReparenting="“

      android:allowBackup=""
      android:allowClearUserData=""
      android:allowNativeHeapPointerTagging=""
      android:backupAgent="string"
      android:backupInForeground=""
      android:banner="drawable resource"
      android:debuggable=""

      android:isGame=""
      android:killAfterRestore=""
      android:largeHeap=""
      android:label="string resource"
      android:logo="drawable resource"
      android:manageSpaceActivity="string"
      android:name="string"
      android:vmSafeMode=""["true" | "false"] >
   . . .
</application>
```

https://developer.android.com/guide/topics/manifest/application-element

3.18 Running the App

Once you run the demo app and press the send message button on the main Activity, MessageDisplayActivity starts but is empty

This is because the second Activity has an empty layout

In the following, we will add some elements to the second activity layout to make it more interesting

3.19 Receiving Messages/Data from an Activity

In the DisplayMessageActivity class and inside the onCreate method, you can use the **getIntent** method to receive the intent object passed by the main activity

The getIntent method is a public method in the Activity class and can be used like this: **Intent intent = getIntent()**

The message, or the data delivered by MainActivity, can be extracted as well

The code snippet on next slide shows extracting data from the Intent object received from the caller Activity

The key is declared in the strings.xml file as the message and that is why we are using R.stirng.message as a key to retrieve its value

3.19 Receiving Messages/Data from an Activity

```
Intent intent = getIntent();
        if (intent != null) {
          if (intent.hasExtra(getString(R.string.message))) {
              intentMessage = intent.getStringExtra(
                    getString(R.string.message));
}
```

Few notes:

- We used getStringExtra() because we know that the type of message passed by MainActivity is a string. If it was something else, for example, int, then we use getIntExtra().
- To retrieve the string message, i.e., the string value, you also need to know the message key.
- The key is declared in the strings.xml file as the *message* and that is why we are using ***R.stirng.message*** as a key to retrieve its value.
- https://developer.android.com/reference/android/content/Intent

3.20 Responding to the Messages from an Activity

To display the message on the screen of the second Activity, i.e., the MessageDisplayActivity view, you need to add a TextView element to the display layout of the Activity

3.20 Responding to the Messages from an Activity

To add a TextView, add the following element to the activity display_message.xml file.

```
<TextView
    android:id="@+id/message"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:text="TextView"
    android:textAlignment="center"
    android:textAppearance="@style/TextAppearance.AppCompat.Body1"
    android:textSize="24sp"
    android:typeface="normal"
    android:visibility="visible“
/>
```

https://m2.material.io/design/typography/the-type-system.html#type-scale

https://m2.material.io/design/typography/the-type-system.html#type-scale

3.20 Responding to the Messages from an Activity

Inside the onCreate method of DisplayMessageActivity, you can access the TextView using **findViewByID** and passing Resource id as a parameter

TextView messageTextView = findViewById(R.id.*message*);

messageTextView.setText(intentMessage);

The parameter “R.id.message” is referencing the id we have included inside the Textview element definition: **android:id="@+id/message".**

The **setText** method from the TextView class is used to put the delivered message on the screen

3.20 Responding to the Messages from an Activity

To make the app a bit more interesting, added a button on the second activity. When it is pressed, the Display message Activity gets closed and return to the main activity

Here is the button description inside activity_display_message.xml.

```
<Button
   android:layout_width="match_parent"
   android:layout_height="60dp"
   android:layout_weight=".5"
   android:background="@android:color/holo_green_dark"
   android:text="Back to Main"
   android:visibility="visible"
   android:onClick="goToMain"/>
```

3.20 Responding to the Messages from an Activity

The method implementation inside the Activity must be public with a void return type, and it should have one parameter of type View

Note, inside the button description there is an entry called ***android:onClick*** which takes the name of the method that will be executed when the button is pressed.

You name your method inside MessageDisplayActivity, in this example, as **'goToMain'.**

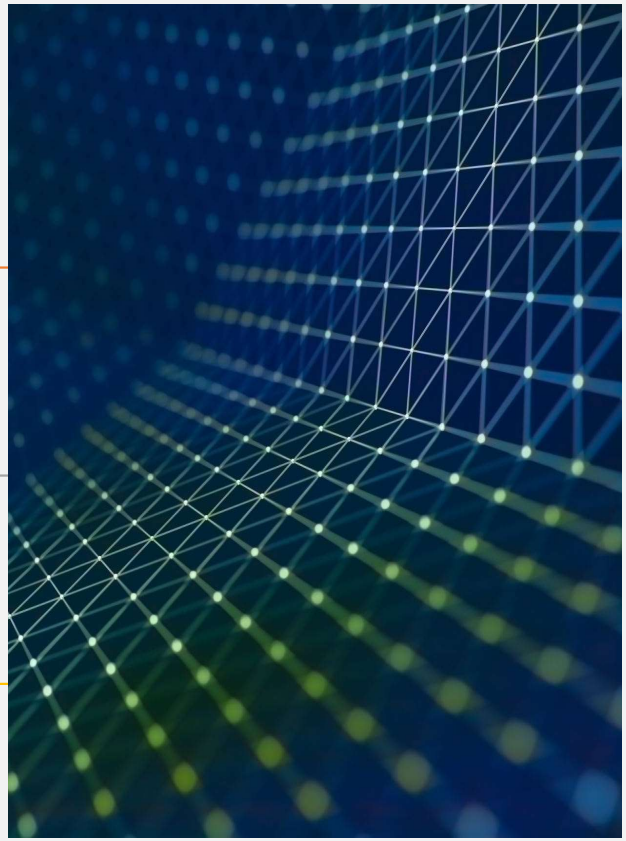
```
public void goToMain(View view) {
        Intent intent = new Intent (this, MainActivity.class);
        startActivity(intent);
}
```

Part 3: Debugging Information

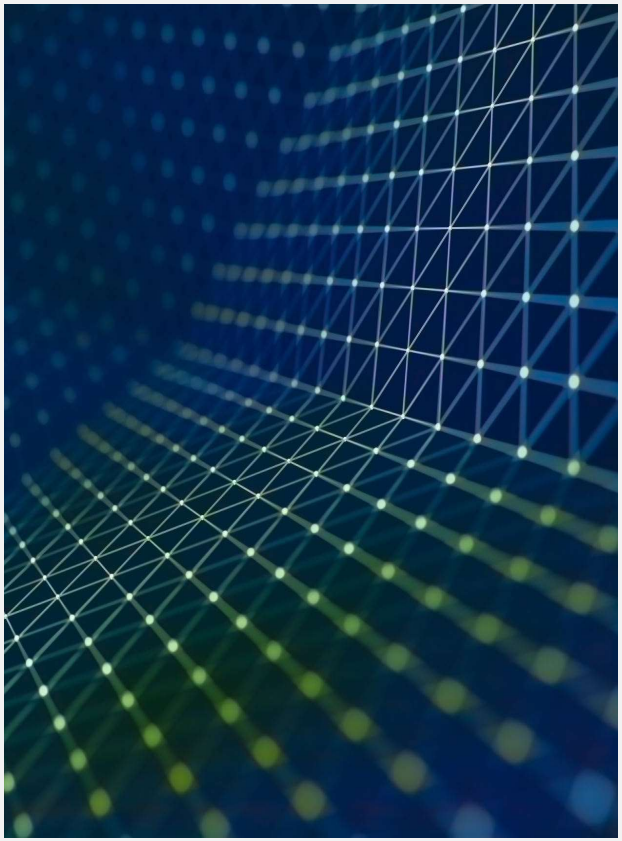
The Android SDK has a **Log class** in the util package to log information and print debugging information to the Logcat window

The Android Log class can be utilized like System.out.println in Java for debugging

The Log class has five methods for displaying debugging information



Part 3: Debugging Information



The methods are:

1. **Log.i()** for writing information
2. **Log.w()** for writing warnings
3. **Log.e()** for writing errors
4. **Log.v()** for writing verbose or detailed messages
5. L**og.d()** for writing debugging statements

In this part, we will describe how to use the Log class methods in your app.

3.21 Debugging Using Log.d

You use log statements in your code for various purposes

For example, you might want to confirm that an Intent fired by one activity is received by another.

This confirmation can be done with Log.d()

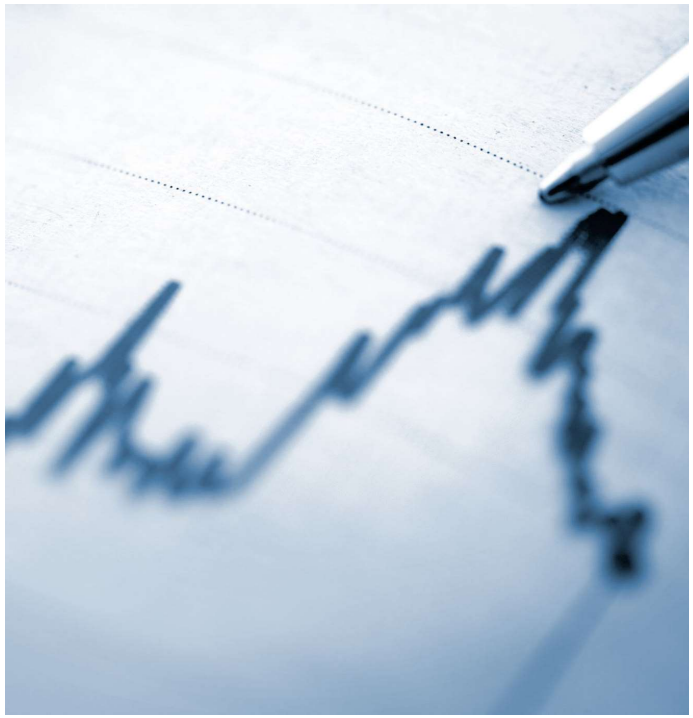
To use Log.d(), you first need to create a TAG in your code and then call Log.d() with the tag

The TAG variable is usually declared as constant in the class, and the value of the TAG variable is the activity name of your app

private static final String TAG = "MainActivity";

The syntax of the log methods is like this:

Log.type (TAG, "message");



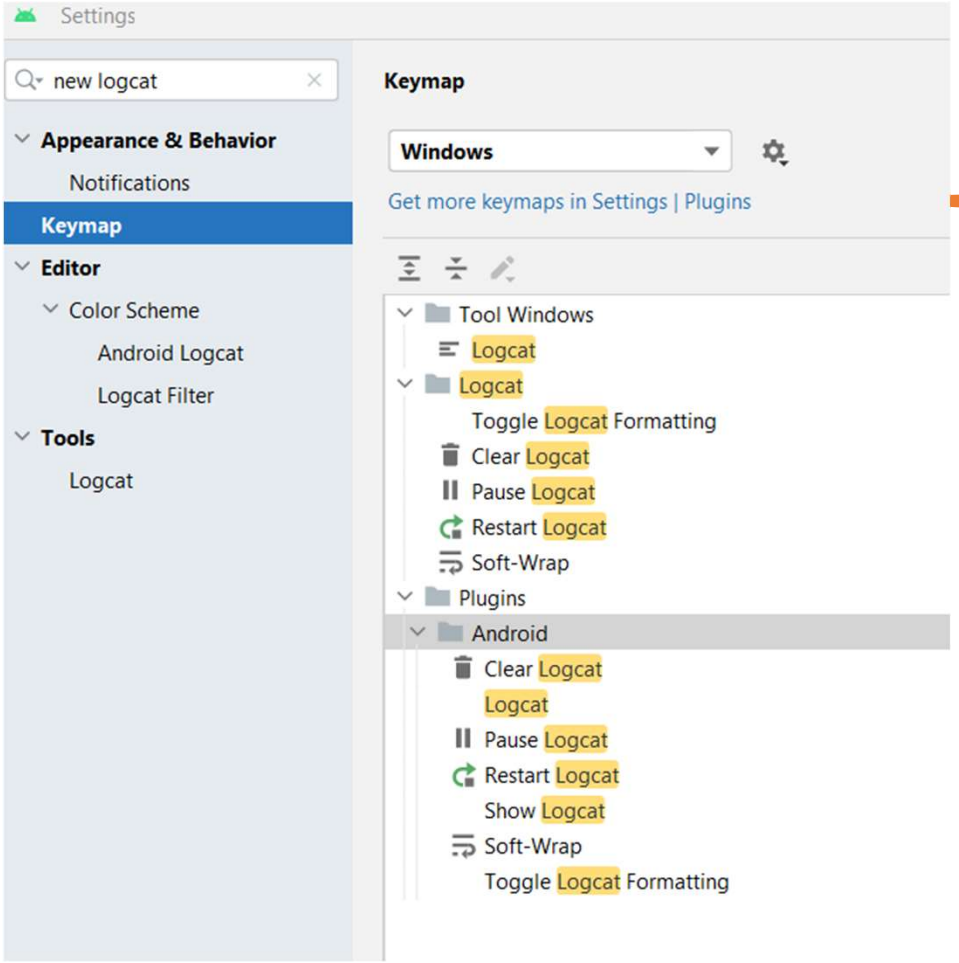
3.22 Using LogCat to View Log Messages

LogCat is a command-line tool integrated into Android Studio

You'll find LogCat at the bottom of Android Studio or by clicking the View tab from the Android menubar followed by the Tool Window and LogCat menu item

What is described here is just a short introduction to Log class and viewing logs in the LogCat window in case you need it while you are creating and running your first app

3.21 Debugging Using Log.d



3.23 Do It Yourself

For this part, we have defined TextView and Button inside the Activities' Layout XML files

These objects can be defined as Java code inside your MainActivity and DisplayMessageyActivity classes

For example, you can create TextView as follows:

```
TextView = new TextView(this);
textView.setTextSize(40);
textView.setText(message);
```

Re-write the MainActivity and the DisplayMessageActivity classes using Java code only.

3.23 Do It Yourself

Each new version of the Android API adds new features to the prior version

We listed some of the Android APIs and their unique features in this chapter.

Search the Android developer page, to find out what is unique to other APIs that we have not listed in the chapter.

3.23 Do It Yourself

Familiarize yourself with Android shortcuts.

Put the link below in your browser or click to see the list of Android shortcuts. (https://developer.android.com/studio/intro/keyboard-shortcuts)

Part 4: Localize Your App and Resources

Android runs on a large number of devices and in all countries and regions of the world

For your app to be used by the largest possible number of users, it should at least handle text, numbers, currency, and graphics in ways that is suitable to users in many places

This part describes the best practices for localizing Android apps

3.24 Create a Resource File for Second Language

To support texts in other languages in your app, you need to create a strings.xml file like the default strings.xml file for the English language for every language you would like to support

3.24 Create a Resource File for Second Language

Right-click on the "*res*" folder of your app and select "New" followed by *Android Resource File*

A window will open. Select "Locale" from the list and click the ">>" button to see the list of languages that you can choose from.

select any language, other than English, that you would like to support.

In the file name field type *strings* and press *OK*.

This should create a second "strings.xml" file in the *values* folder.

3.24 Create a Resource File for Second Langauge

The pattern for the newly created filename would be *string(XX-rYY)*

the string is the language like French or Arabic,

the first XX is the language qualifier like fr or ab,

and the second rYY is the specific region you selected.

You have a choice to not select a specific region. To do so, you leave the Specific Region option to *Any Region*.

For example, when French language is selected and is not specified for a specific region, the file name would be:

french.xml(fr)

3.25 Create Resource Entries for Languages Supported

To add content to the newly created strings.xml file for the second language, copy the xml elements between the <resources> tags of the original “values/strings.xml” file and paste them between the <resources> tags of the new strings .xml file

In the strings.xml file for the second language, modify the string values to the new language

For example, the app name element in English and French string files would be as follows respectively

<string name="app_name">My First Application</string>

<string name="app_name">Ma première application</string>

3.26 Set Device Language

To be able to read the content of the second strings.xml you just created, you need to change the language of your device or the emulator if you are running your app on the emulator

Set the language of the phone to whatever you choose as your second language for your app

Your app will automatically read the strings from the other strings.xml file you created

3.5 Summary

In this part, we described how to create your first Android app

We defined Activity as a Java class or a component in your app representing a window that fills the screen

We also described both explicit and implicit Intent classes as a message object and a description of the operation that needs to be carried out

3.5 Summary

We studied:

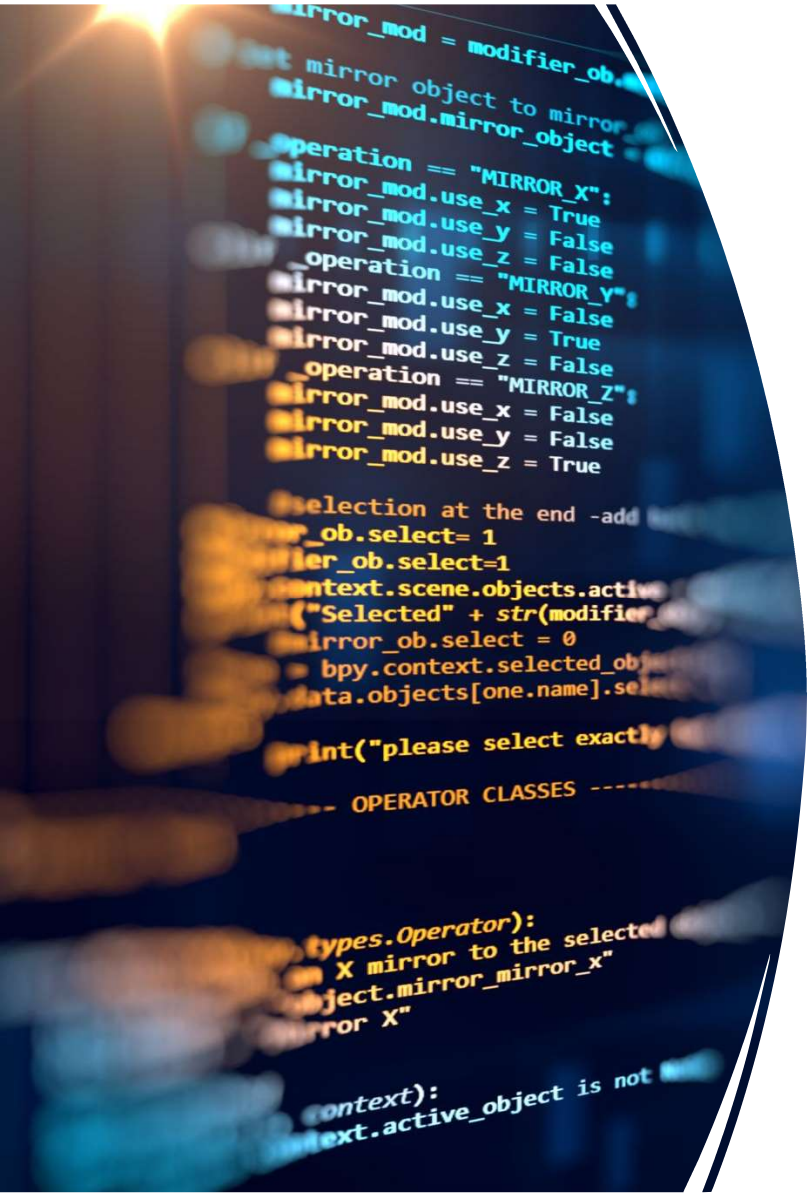
- how to start a second Activity with an Intent object using the startActivity method

- passing data between Activities with extras, and going back and forth between Activities

- defining a layout in XML, defining an Activity by extending AppCompatActivity

- connecting Activity to the XML layout file using the setContentView method in the onCreate method,

- declaring Activity in the Android manifest file



Check Your Knowledge

Below are some of the fundamental concepts and vocabularies that have been covered in this chapter.

To test your knowledge and your understating of this chapter you should be able to describe each of the below concepts in one or two sentences.

Activity

Android API

Android SDK

Clean project

EditText

findViewByID

hasExtra

Check Your Knowledge

- Implicit Intent
- Intent
- Local
- Log class
- LogCat
- OnClick
- OnCreate
- Open the Layout editor
- getApplicationContext
- getContext

Check Your Knowledge

- getIntent
- getStringExtra
- intent.putExtra
- Project Sync
- R.java
- setText
- startActivity
- TextView
- Uri

Further Reading

For more information about the topics covered in this lesson, we suggest that you refer to the online resources listed in book chapter

These links provide additional insight into the topics covered

The links are mostly maintained by Google and are a part of the Android API specification

The resource titles convey which section/subsection of the chapter the resource is related to