

Course Manual for Android App Development

MODULE 1: Revisiting JAVA & OOP

Lecture-1

CHAPTER OVERVIEW:

Android is an operating system based on the Linux kernel. Android is developed in the Android Open Source Project (AOSP). This project is lead by Google.

1.1-Introduction to Android

1.2-Installing the Android Development

OBJECTIVS:

- Android operating system and different versions.
- Android application single installable unit which can be started and used independently. An Android application consists of configuration files.

LEARNING OUTCOME:

- Android versions
- Android application
- Technology stack needed for android apps development

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- API level
- Version
- Environment Setup

CORE DISCUSSION:

- **Historical approach :**

The founding of Android-In October 2003, well before the term “smartphone” was used by most of the public, and several years before Apple announced its first iPhone and its iOS, the company Android Inc was founded in Palo Alto, California. Its four founders were Rich Miner, Nick Sears, Chris White, and Andy Rubin. At the time of its public founding, Rubin was quoted as saying that Android Inc was going to develop “smarter mobile devices that are more aware of its owner’s location and preferences.”

Course Manual for Android App Development

- **Fundamental technology**

Android Studio: Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

Eclipse IDE: Eclipse IDE (integrated development environment). Included in the Android SDK download, the Eclipse IDE provides the “hands-on” controls you need for writing your app using Java, the Android SDK and the Android ADT. Android ADT (Android Development). Apr 25, 2013

- **Alternative technology**

Android development by unity :

The Android environment setup topic of the Unity Manual contains a basic outline of the tasks that you must complete before you are able to run code on your Android device, or in the Android emulator. For more in-depth information on setting your Android development environment, see the step-by-step instructions on the Android developer portal.

- **Scope of implementation & case study**

1. Oriented Programming in Java
2. Problem solving
3. Simple input output program

1.1-Introduction to Android:

The Android operating system can be divided into the four areas as depicted in the following graphic. An Android application developer typically works with the two layers on top to create new Android applications.

The levels can be described as:

- Applications - Contains the applications, like the Browser, Camera, Gallery, Music and Phone
- Application framework - An API which allows high-level interactions with the Android system
- Libraries and runtime - The libraries for many common framework functions, like graphic rendering, data storage, web browsing. Also contains the Android runtime, as well as the core Java libraries for running Android applications.
- Linux kernel - Communication layer for the underlying hardware.

Course Manual for Android App Development

Android versions

The Android operating system is published in different versions. The following table gives an overview of the available versions.

<i>Table 1. Android versions</i>		
Code name	Version	API level
Oreo	8.0	26
Nougat	7.0 – 7.1.1	24 -25
Marshmallow	6.0	23
Lollipop	5.1	22
Lollipop	5.0	21
KitKat	4.4 - 4.4.4	19
Jelly Bean	4.1.x- 4.3.x	16 - 18
Ice Cream Sandwich	4.0. - 4.0.4	14 - 15
Honeycomb	3.2.x	13
Honeycomb	3.0 - 3.1	11 - 12
Gingerbread	2.3 - 2.3.7	9-10
Froyo	2.2.x	8
Eclair	2.1	7
Eclair	2.0 - 2.0.1	5 -6
Donut	1.6	4

Course Manual for Android App Development

Table 1. Android versions

Code name	Version	API level
Cupcake	1.5	3
(no code name)	1.1	2
(no code name)	1.0	1

Android application

An Android application (app) is a single installable unit which can be started and used independently. An Android application consists of configuration files, Java source and resource files. You can define the following components in your configuration files:

- Application
- Activities
- Services
- Content Provider
- Broadcast Receiver

Technology stack needed for android apps development

To develop android apps you need the following tools:

1. JDK
2. Android SDK
3. Android Studio

Course Manual for Android App Development

1.2-Installing the Android Development

Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advanced tab > **Environment Variables** and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk

Configure your computer for setting up android application development environment (practical)

Setup Android Apps Development Environment in your computer (practical)

PRACTICE:

Android versions-> JDK-> Android SDK-> Android Studio ->Android application

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

<http://www.wrox.com/WileyCDA/WroxTitle/Professional-Android-2-Application-Development.productCd-0470565527.html>

REFARANCE LINK:

<https://developer.android.com/guide/>

<https://developer.android.com/training/basics/firstapp/>

MODULE 2: Revisiting JAVA & OOP

Lecture-2

CHAPTER OVERVIEW:

Programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as *attributes*; and code, in the form of procedures, often known as *methods*. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another

2.1-OOP Concepts

2.2-Java basic

OBJECTIVS:

- Core OOPS concepts
- Data type specifies the size and type of values
- Java Control Flow statements control the order of execution in a java program, based on data values and conditional logic.

LEARNING OUTCOME:

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance
- Association
- Aggregation
- Composition
- Data types in Java
- Work with java variables: integer, string, double
- Understand conditional statement, looping
- If-statement

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Data types in Java
- Methods in java

CORE DISCUSSION:

- **Historical approach :**

OOP languages simula (1967) is generally accepted as being the first language with the primary features of an object-oriented language. It was created for making simulation programs, in which what came to be called objects were the most important information representation.

- **Fundamental technology**

Java view technologies and frameworks- are web-based software libraries that provide the user interface, or "view-layer", of Javaweb applications. Such application frameworks are used for defining web pages and handling the HTTP requests (clicks) generated by those web pages. As a sub-category of web frameworks, view-layer frameworks often overlap to varying degrees with web frameworks that provide other functionality for Java web applications

1. Java Servlets.
2. Core Java
3. JavaServer Pages (JSP)
4. Enterprise JavaBeans (EJB)
5. Java Database Connectivity (JDBC)

- **Scope of implementation & case study**

1. Oriented Programming in Java
2. Core Java
3. Problem solving
4. Simple input output program

2.1-OOP Concepts:Core OOPS concepts are:

1. Abstraction

Course Manual for Android App Development

2. Encapsulation
3. Polymorphism
4. Inheritance
5. Association
6. Aggregation
7. Composition

Abstraction: Abstraction is the concept of hiding the internal details and describing things in simple terms. For example, a method that adds two integers. The method internal processing is hidden from outer world. There are many ways to achieve abstraction in object oriented programming, such as encapsulation and inheritance.

A java program is also a great example of abstraction. Here java takes care of converting simple statements to machine language and hides the inner implementation details from outer world.

Encapsulation: Encapsulation is the technique used to implement abstraction in object oriented programming. Encapsulation is used for access restriction to a class members and methods.

Access modifier keywords are used for encapsulation in object oriented programming. For example, encapsulation in java is achieved using private, protected and public keywords.

Polymorphism: Polymorphism is the concept where an object behaves differently in different situations. There are two types of polymorphism – compile time polymorphism and runtime polymorphism.

Compile time polymorphism is achieved by method overloading. For example, we can have a class as below.

```
public class Circle {  
    public void draw() {  
        System.out.println("Drawing circle with default color Black and diameter 1 cm.");  
    }  
  
    public void draw(int diameter) {  
        System.out.println("Drawing circle with default color Black and  
diameter"+diameter+" cm.");  
    }  
  
    public void draw(int diameter, String color) {  
        System.out.println("Drawing circle with color"+color+" and diameter"+diameter+"  
cm.");  
    }  
}
```

Here we have multiple draw methods but they have different behavior. This is a case of method overloading because all the methods name is same and arguments are different. Here compiler will be able to identify the method to invoke at compile time, hence it's called compile time polymorphism.

Course Manual for Android App Development

Runtime polymorphism is implemented when we have “IS-A” relationship between objects. This is also called as method overriding because subclass has to override the superclass method for runtime polymorphism. If we are working in terms of superclass, the actual implementation class is decided at runtime. Compiler is not able to decide which class method will be invoked. This decision is done at runtime, hence the name as runtime polymorphism or dynamic method dispatch.

```
package com.journaldev.test;

public interface Shape {

    public void draw();
}

package com.journaldev.test;

public class Circle implements Shape {

    @Override
    public void draw() {
        System.out.println("Drawing circle");
    }

}

package com.journaldev.test;

public class Square implements Shape {

    @Override
    public void draw() {
        System.out.println("Drawing Square");
    }

}
```

Shape is the superclass and there are two subclasses Circle and Square. Below is an example of runtime polymorphism.

```
Shape sh = new Circle();
sh.draw();
```

```
Shape sh1 = getShape();//some third party logic to determine shape
sh1.draw();
```

In above examples, java compiler don't know the actual implementation class of Shape that will be used at runtime, hence runtime polymorphism.

Course Manual for Android App Development

Inheritance: Inheritance is the object oriented programming concept where an object is based on another object. Inheritance is the mechanism of code reuse. The object that is getting inherited is called superclass and the object that inherits the superclass is called subclass.

We use extends keyword in java to implement inheritance. Below is a simple example of inheritance in java.

```
package com.journaldev.java.examples1;

class SuperClassA{

    public void foo(){
        System.out.println("SuperClassA");
    }

}

class SubClassB extends SuperClassA{

    public void bar(){
        System.out.println("SubClassB");
    }

}

public class Test{
    public static void main(String args[]){
        SubClassB a = new SubClassB();

        a.foo();
        a.bar();
    }
}
```

Association: Association is the OOPS concept to define the relationship between objects. Association defines the multiplicity between objects. For example Teacher and Student objects. There is one to many relationship between a teacher and students. Similarly a student can have one to many relationship with teacher objects. However both student and teacher objects are independent of each other.

Aggregation: Aggregation is a special type of association. In aggregation, objects have their own life cycle but there is an ownership. Whenever we have “HAS-A” relationship between objects and ownership then it’s a case of aggregation.

Composition: Composition is a special case of aggregation. Composition is a more restrictive form of aggregation. When the contained object in “HAS-A” relationship can’t exist on its own, then it’s a case of composition. For example, House has-a Room. Here room can’t exist without house.

2.2-Java basic:

Course Manual for Android App Development

Data types in Java:

Data type specifies the size and type of values that can be stored in an identifier. The Java language is rich in its data types. Different data types allow you to select the type appropriate to the needs of the application.

Data types in Java are classified into two types:

Primitive— Which include Integer, Character, Boolean, and Floating Point.

Non-primitive— Which include Classes, Interfaces, and Arrays.

Work with java variables: integer, string, double

Integer: An integer is a number that can be written without a fractional component. For example, 21, 4, 0, and -2048 are integers, while 9.75, $5\frac{1}{2}$, and $\sqrt{2}$ are not.

String: String is basically an object that represents sequence of char values.

Double: The double data type is a double-precision 64-bit IEEE 754 floating point. It is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.

Understand conditional statement, looping:

There are three main categories of control flow statements:

Selection statements: if, if-else and switch.

Loop statements: while, do-while and for.

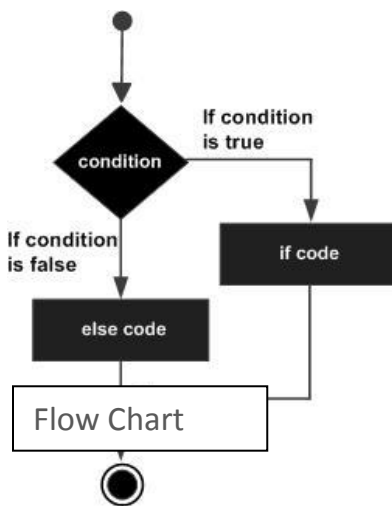
Transfer statements: break, continue, return, try-catch-finally and assert.

We use control statements when we want to change the default sequential order of execution

If-statement

The “if” statement in Java works exactly like in most programming languages. With the help of “if” you can choose to execute a specific block of code when a predefined condition is met. The structure of the “if” statement in Java looks like this:

for Android App Development



Expression is true.

```
int test = 5;

if (test < 10)
{
    // codes
}
```

// codes aft

Expression is false.

```
int test = 5;

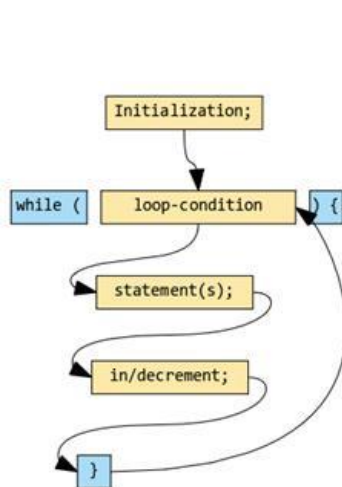
if (test > 10)
{
    // codes
}
```

// codes after if

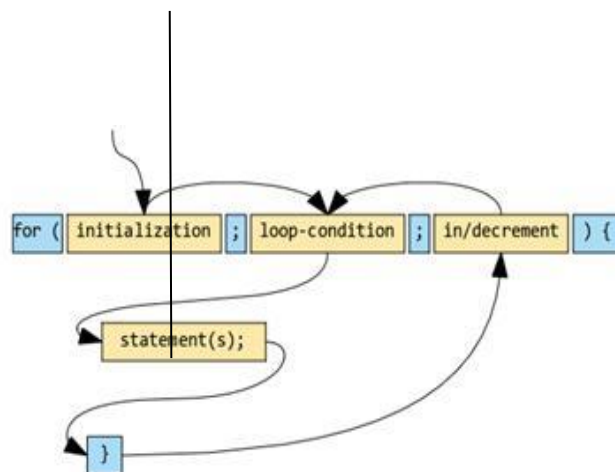
Code Example

Loops

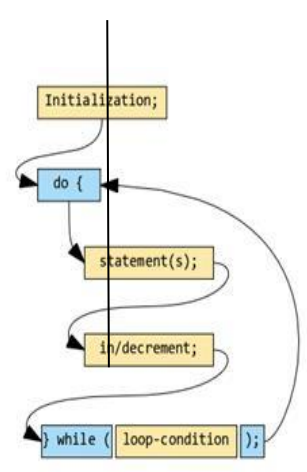
Java provides three repetition statements/loop statements that enable programmers to control the flow of execution by repetitively performing a set of statements as long as the continuation condition remains true. These three looping statements are called [*for*, *while*, and *do...while*] statements. The '*for*' and '*while*' statements perform the repetition declared in their body **zero or more times**. If the loop continuation condition is false, it stops execution. The '*do...while*' loop is slightly different in the sense that it executes the statements within its body for **one or more times**.



While Loop



For Loop



Do - While Loop

Switch

A 'switch' statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Break

The 'break' statement in Java programming language has the following two usages –

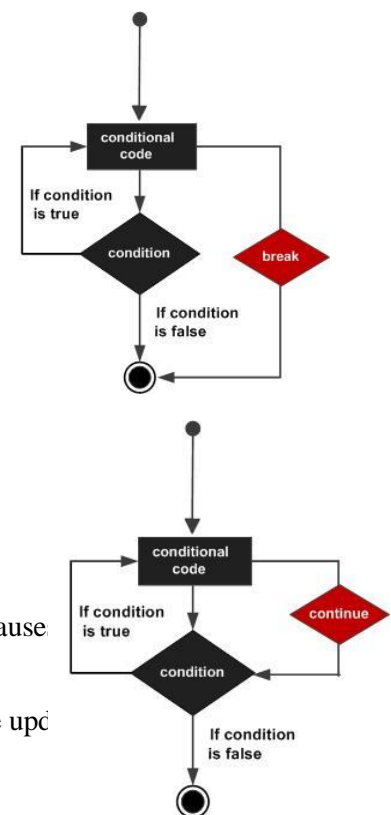
When the 'break' statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next

Continue:

It can be used to terminate a case in the 'switch' statement (covered in the next chapter).

The 'continue' keyword can be used in any of the loop control structures. It causes immediately jump to the next iteration of the loop.

In a for loop, the continue keyword causes control to immediately jump to the upc loop or do/while loop, control immediately jumps to the Boolean expression.



Methods in java

A Java method is a collection of statements that are grouped together to perform an operation.

User defined type

In addition to the basic data types already described, Java supports user-defined data types, which are aggregations of these basic types. These complex data types include arrays, sequences, enumerations, and constructed data types you define yourself using structs and unions.

Course Manual for Android App Development

A complex data type is used in IDL by first giving it a type name, then using the type name wherever you would use a basic data- or interface-type name (e.g., declaring attributes, method arguments). There are a few ways a name is assigned to a complex data type:

- With structures, unions, and enumerations, the name is included in the declaration of the data type.
- A **typedef** can be used to assign a name to a specific type (basic or complex).

Before we go on to see how complex data types are declared in IDL, let's take a look at how a **typedef** assigns a type name to a complex data type.

PRACTICE:

OOP Concepts ->Java Basic->conditional statement ->Loops

SUGGESTED READING LIST: (Books & Links)

Head First Java - Book by Bert Bates and Kathy Sierra

Java: A Beginner's Guide - Book by Herbert Schildt

Beginning Programming with Java For Dummies - Book by Barry A. Burd

<https://www.pdfdrive.net/head-first-javapdf-e18943750.html>

http://www.academia.edu/32243463/Java_A_Beginners_Guide_6th_Edition_PDF

REFARANCE LINK:

<https://docs.oracle.com/javase/tutorial/java/concepts/>

<http://www.exforsys.com/tutorials/oops-concepts/the-history-of-object-oriented-programming.html>

MODULE 3: Android Development Environments

Lecture-3

CHAPTER OVERVIEW:

You can set environment variables for Android Studio and the command-line tools that specify things like where the SDK is installed and where user-specific data is stored. This page describes the most commonly used environment variables.

3.1-Creating a New Android Studio Project

3.2-Designing the User Interface

3.3-Create New activity

3.4- The Android Manifest file

3.5- Testing in the Emulator

3.6- Opening a Saved App in Android Studio

OBJECTIVS:

- Creating a New Android Studio Project
- View objects are the basic building blocks of User Interface(UI) elements in Android.
- Create a new AVD Manager.

LEARNING OUTCOME:

- Understand View and Layout
- Adding a Form Widgets

NEW TERAMS INTRUDUCE IN THE CHAPTER:

Course Manual for Android App Development

- Select form factors and API level
- Testing in the Emulator

CORE DISCUSSION:

- **Scope of implementation & case study**
 1. Testing in the App by Emulator
 2. Create New App

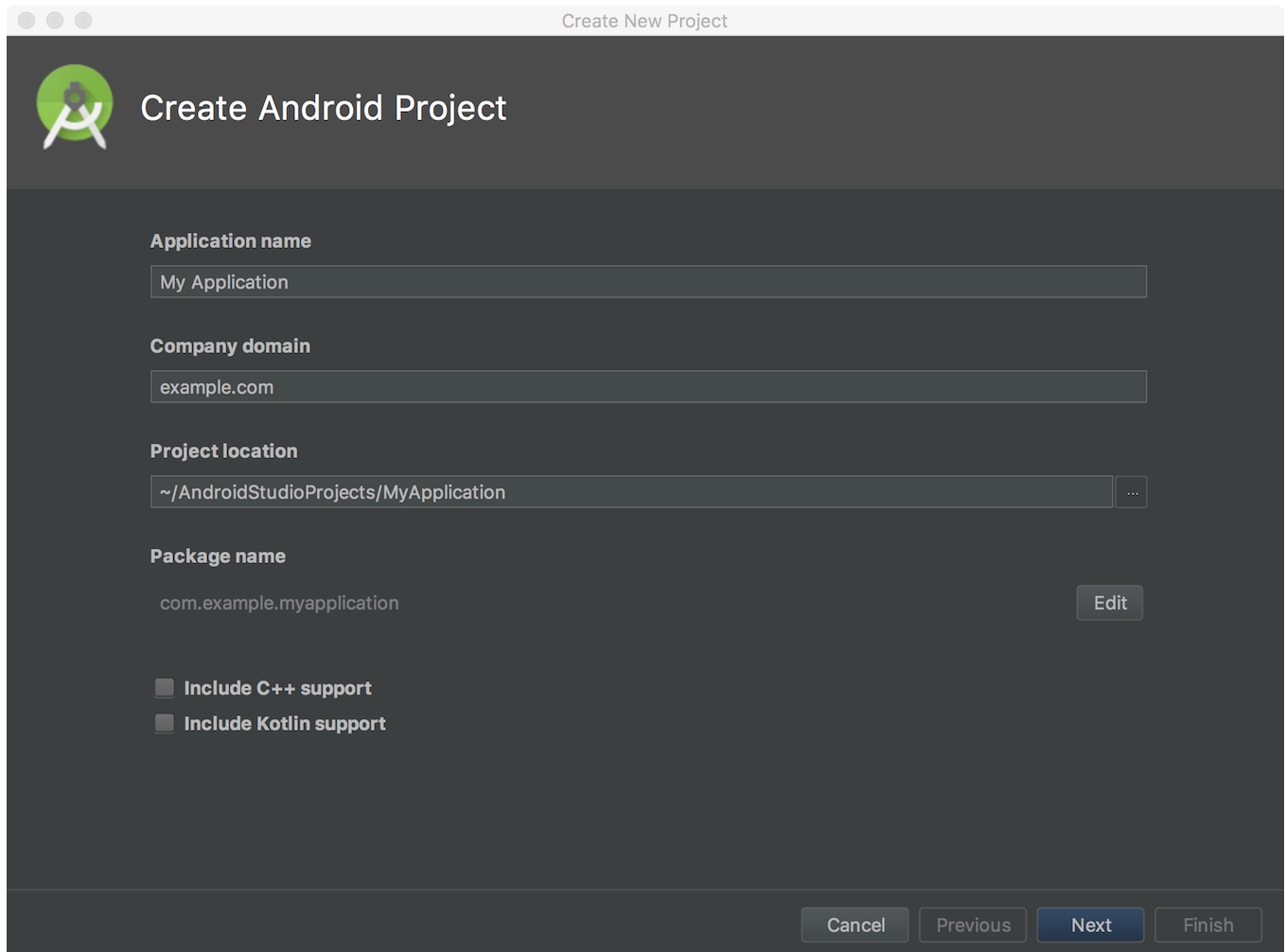
3.1-Creating a New Android Studio Project:

The **New Project** wizard lets you choose the form factors for your app and populates the project structure with everything you need to get started.

Step 1: Start and configure the project

If you don't have a project opened, Android Studio shows the Welcome screen, where you can click **Start a new Android Studio project**.

If you do have a project opened, click **File>New>New Project**.



Create New Project

Create Android Project

Application name
My Application

Company domain
example.com

Project location
~/AndroidStudioProjects/MyApplication

Package name
com.example.myapplication Edit

☐ Include C++ support
☐ Include Kotlin support

Cancel Previous Next Finish

Figure 1. The **Configure your new project** screen.

Enter the values for your project then click **Next**.

Step 2: Select form factors and API level

The next window lets you select the device form factors you want to build for, and the minimum version you want to support for each. For each device you select, the wizard adds a corresponding module to your project.

Each module contains all the code and resources that will be built into an Android app package (APK) for the corresponding device. If you later decide to add support for a new device, you can add a module at that time. And you can share code and resources between modules using an Android library.

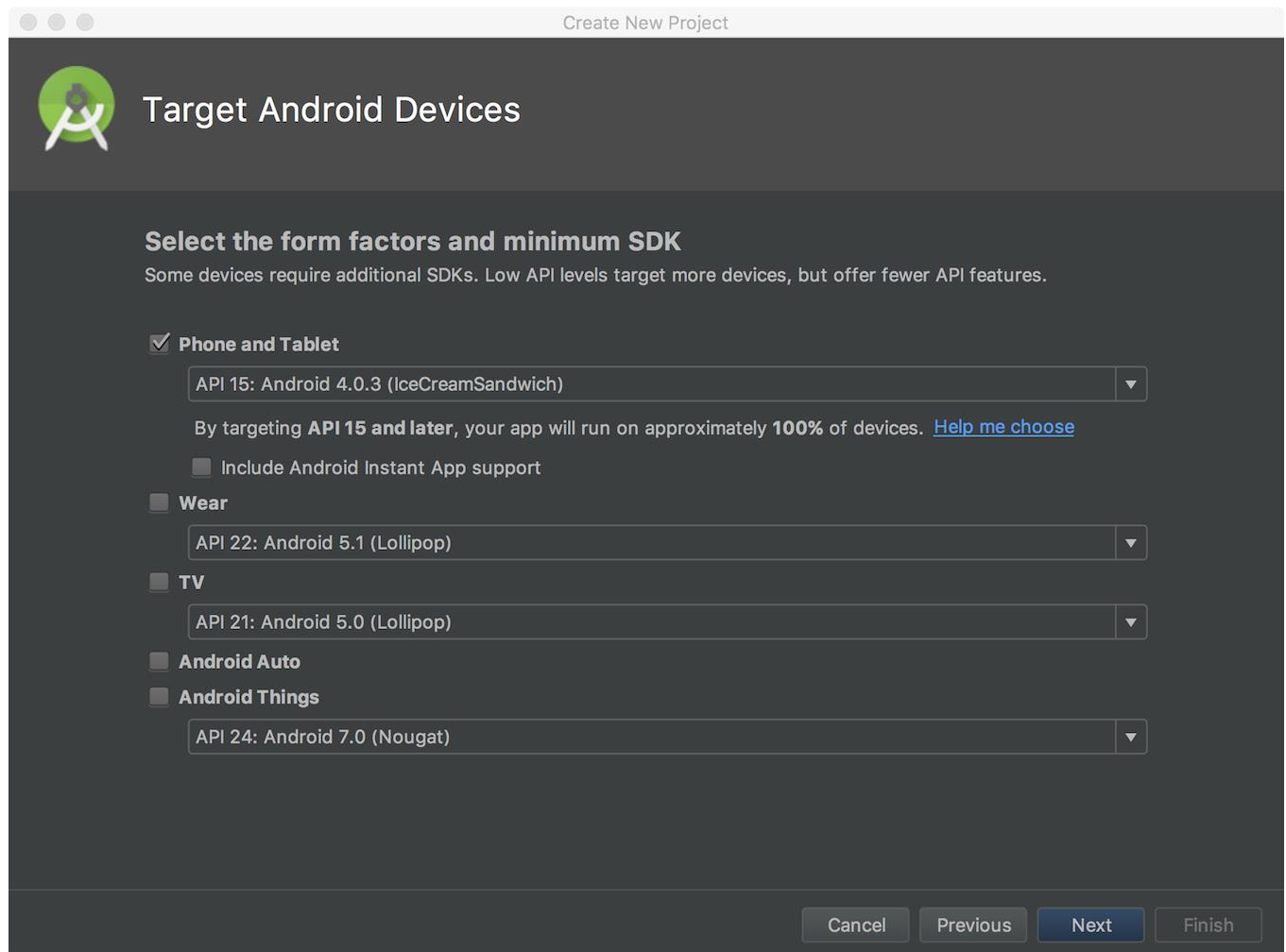


Figure 2. The **Target Android Devices** screen.

To see more information about the different Android versions, click **Help me choose**. This opens a new window that shows the distribution of devices running each version of Android. Click on an API level to see a summary of top features introduced in that version. To return to the wizard, click **OK**.

Once you've selected your form factors and API versions, click **Next**.

Step 3: Add an activity

The next screen lets you select an activity type to add to your app, as shown in figure 3. This screen displays a different set of activities for each of the form factors you selected earlier.

Course Manual for Android App Development

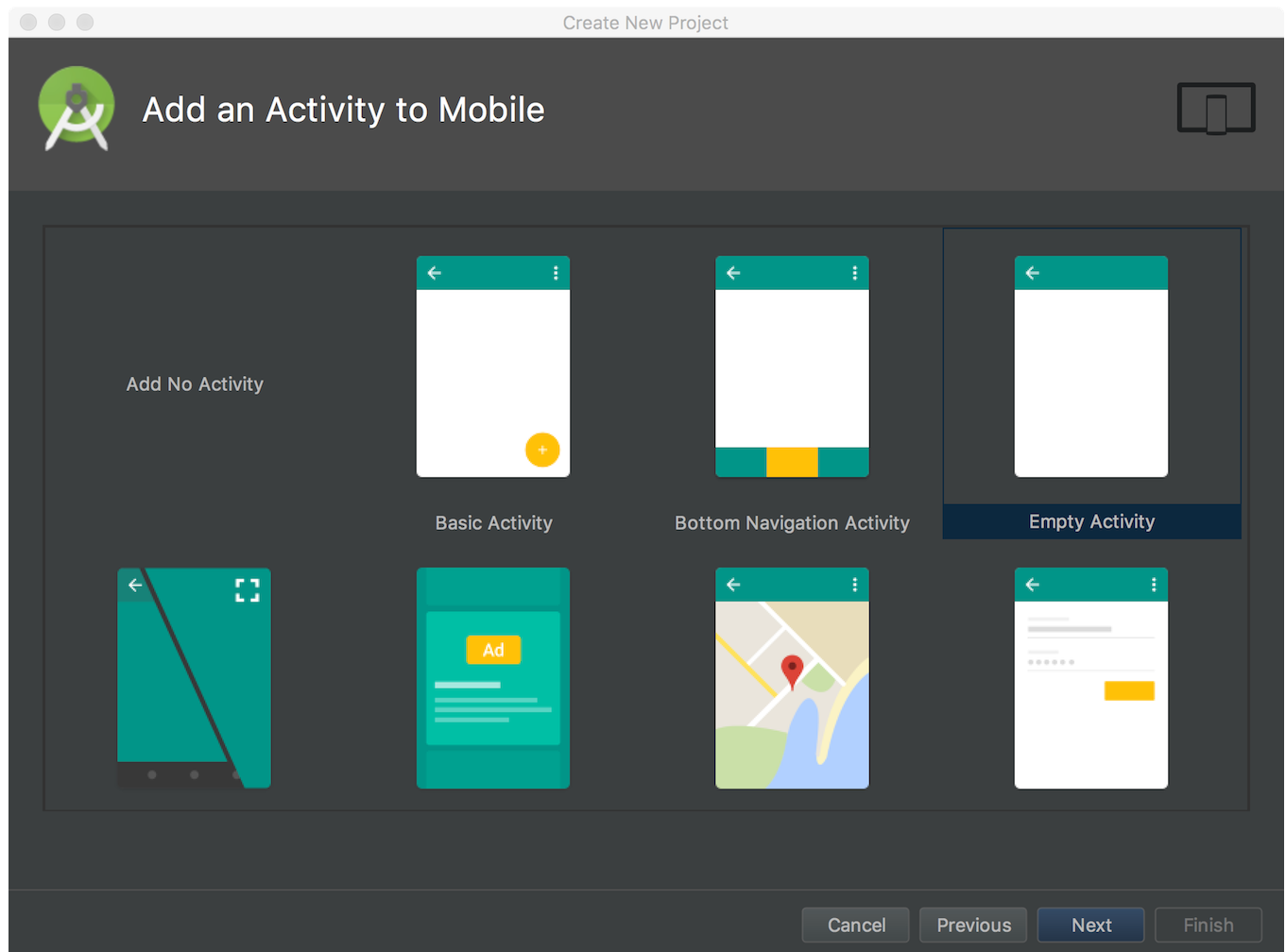


Figure 3. The **Add an Activity** screen for a mobile form factor.

Choose an activity type then click **Next**.

Note: If you choose "Add No Activity," click **Finish** to create the project.

Step 4: Configure your activity

The next screen lets you configure the activity to add to your app, as shown in figure 4.

Course Manual for Android App Development

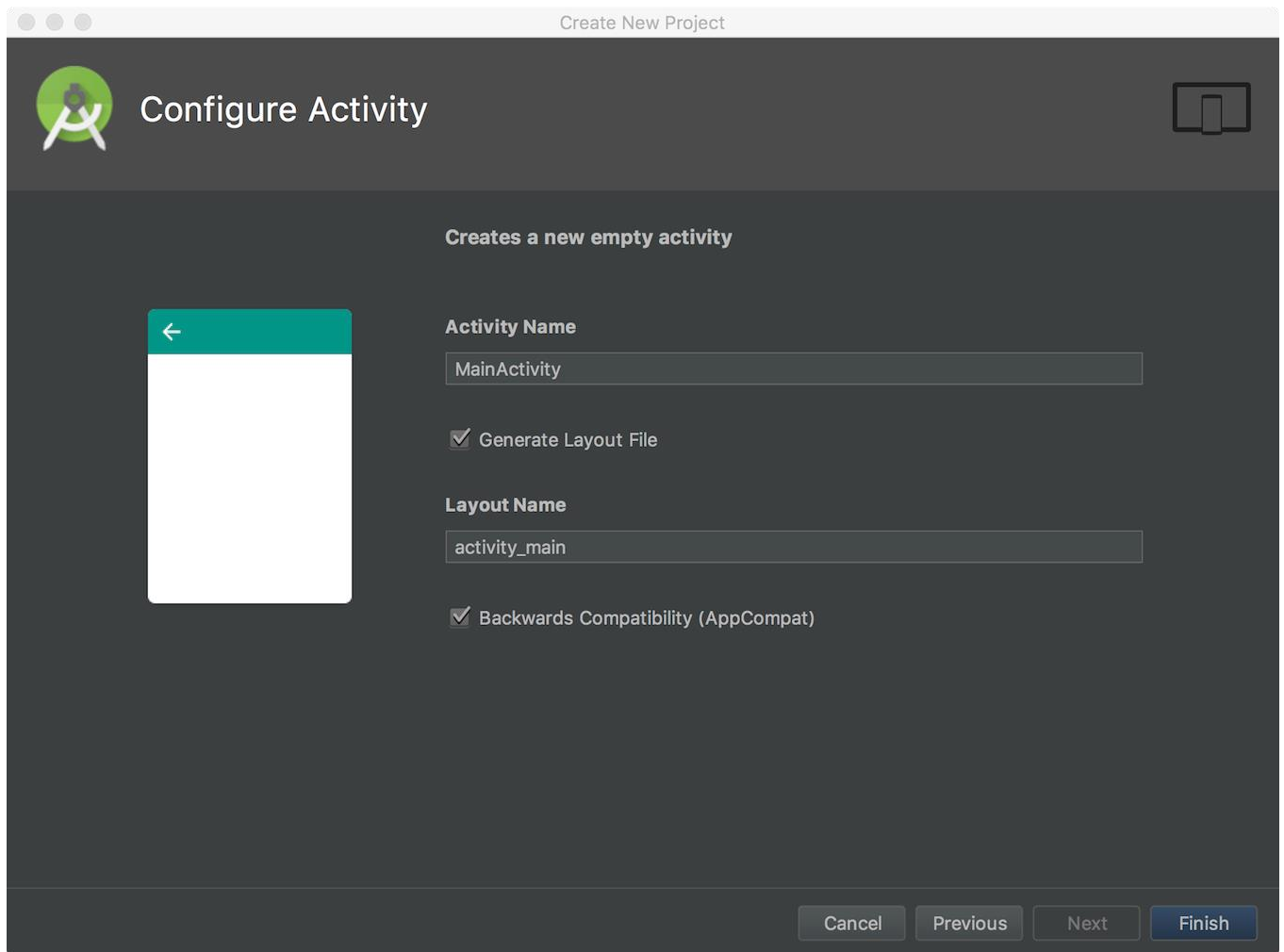


Figure 4. The **Customize the Activity** screen.

Enter the activity name, the layout name, and the activity title. Then click **Finish**. Android Studio now sets up your project and opens the IDE.

Now you're ready to develop your app.

3.2-Designing the User Interface:

Understand View and Layout

Course Manual for Android App Development

View:

View is a simple rectangle box which responds to the user's actions. View refers to the android.view.View class, which is the base class of all UI classes. Examples are EditText, Button, CheckBox etc..

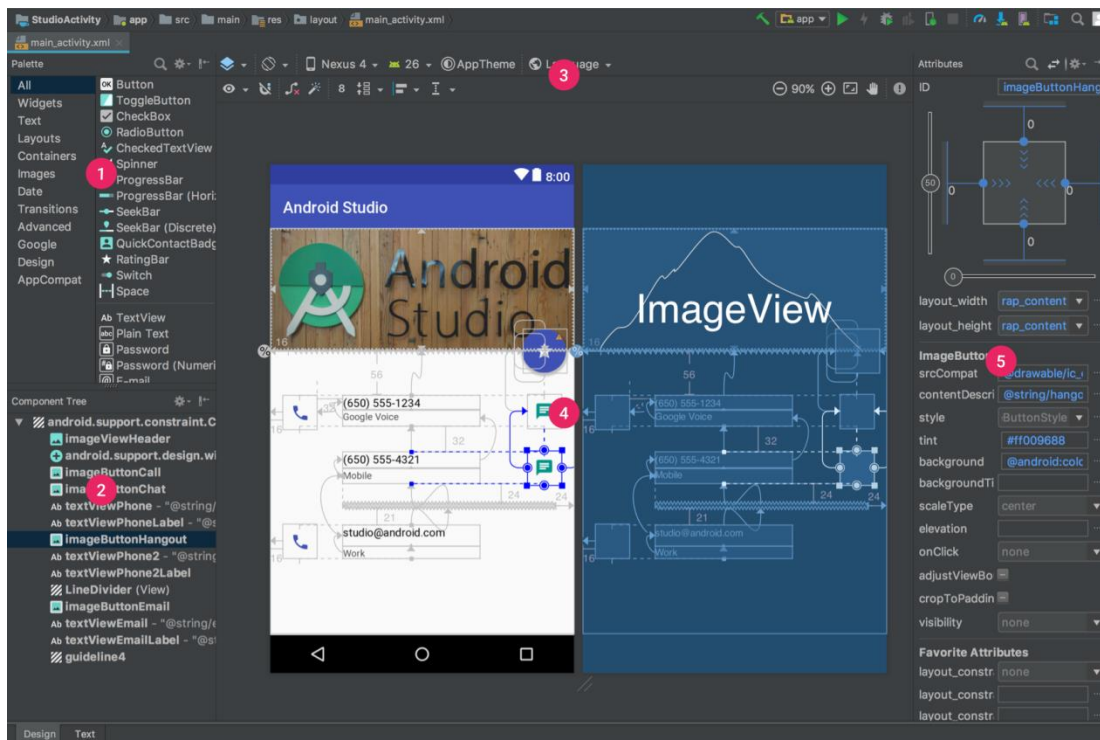
Layout:

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- **Declare UI elements in XML.** Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime.** Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

➤ Adding a Form Widgets

Widgets in Android Studio are a layout builder that enables users to make UI Element easily.



Design a simple activity layout for some basic user operation (practical)

Lecture-4

3.3-Create New activity:

In the **Project** window, right-click the **app** folder and select **New > Activity > Empty Activity**.

In the **Configure Activity** window, enter "DisplayMessageActivity" for **Activity Name** and click **Finish** (leave all other properties set to the defaults).

3.4- The Android Manifest file:

Every app project must have an AndroidManifest.xml file (with precisely that name) at the root of the project source set. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Example manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="riseuplabs.com.videotest">

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:largeHeap="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".priview"></activity>
    </application>

</manifest>
```

Course Manual for Android App Development

3.5- Testing in the Emulator:

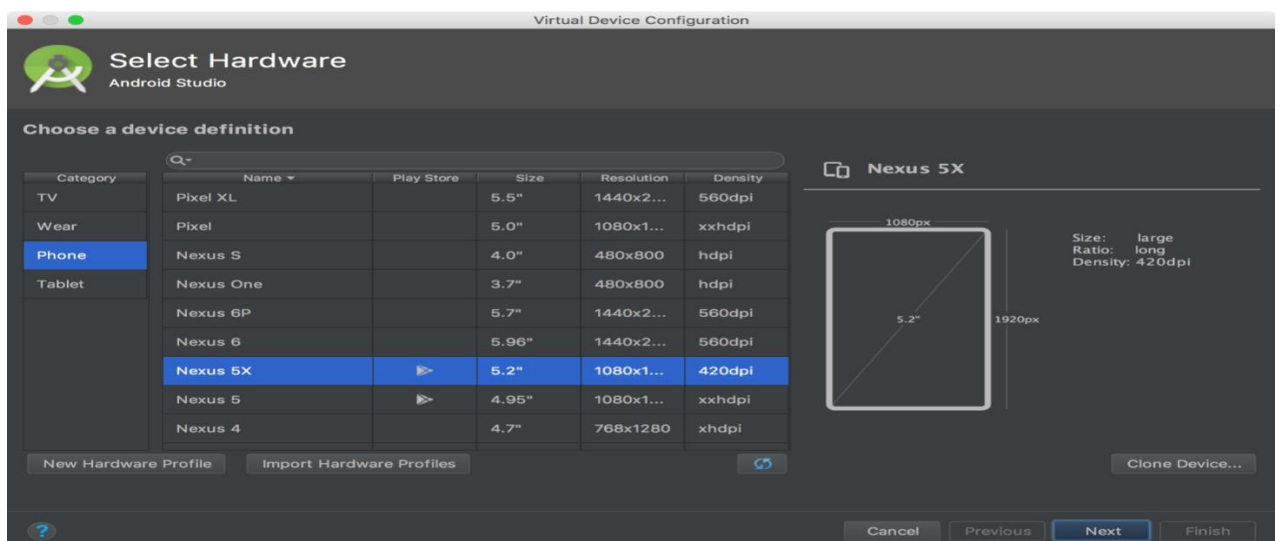
To create a new AVD:

1. Open the AVD Manager by clicking **Tools > AVD Manager**.



2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog.

The **Select Hardware** page appears.

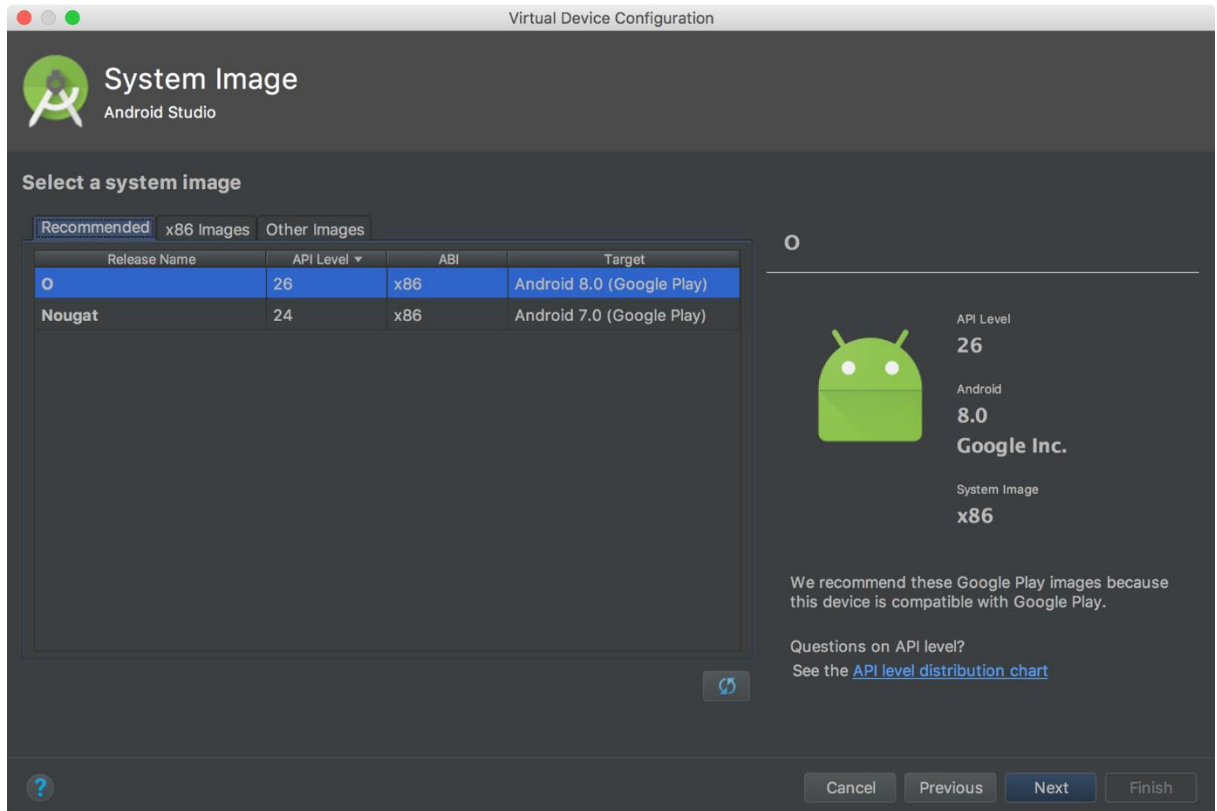


Course Manual for Android App Development

3. Select a hardware profile, and then click **Next**.

If you don't see the hardware profile you want, you can create or import a hardware profile.

The **System Image** page appears.



4. Select the system image for a particular API level, and then click **Next**.

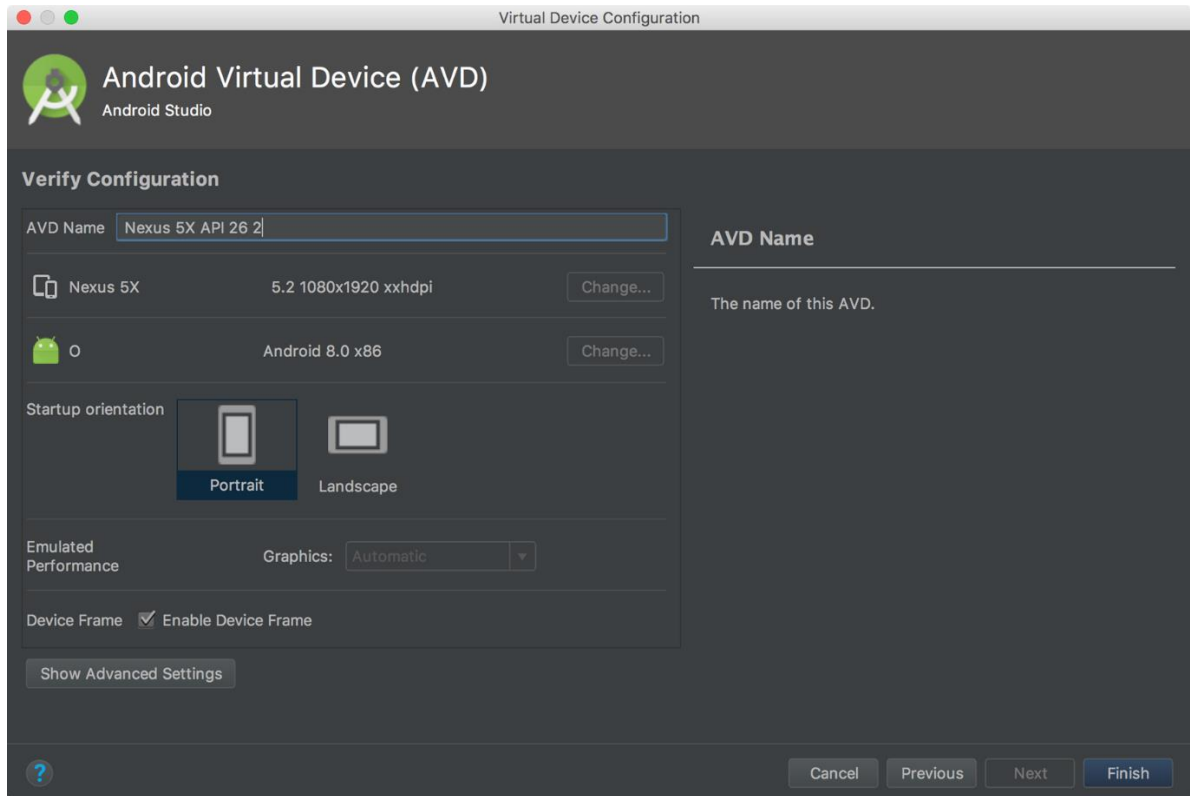
The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

Course Manual for Android App Development

The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as

The **Verify Configuration** page appears.



5. Change AVD properties as needed, and then click **Finish**.

Click **Show Advanced Settings** to show more settings, such as the skin.

The new AVD appears in the **Your Virtual Devices** page or the **Select Deployment Target** dialog.

Course Manual for Android App Development

3.6- Opening a Saved App in Android Studio:

To import an existing project into Android Studio, proceed as follows:

1. Click **File>New>Import Project**.
2. In the **Select Eclipse or Gradle Project to Import** window that appears, navigate to the root directory of the project you want to import.
3. Click **OK**.

Android Studio then opens the project in a new IDE window.

PRACTICE:

Create New Project:

File >New > Activity > Empty Activity.

Import New Project:

1. Click **File>New>Import Project**.
2. In the **Select Eclipse or Gradle Project to Import** window that appears, navigate to the root directory of the project you want to import.
3. Click **OK**.

SUGGESTED READING LIST: (Books & Links)

Head First Java - Book by Bert Bates and Kathy Sierra

Java: A Beginner's Guide - Book by Herbert Schildt

Beginning Programming with Java For Dummies - Book by Barry A. Burd

<https://www.pdfdrive.net/head-first-javapdf-e18943750.html>

http://www.academia.edu/32243463/Java_A_Beginners_Guide_6th_Edition_PDF

REFARANCE LINK:

<https://developer.android.com/studio/projects/create-project.html>

<https://developer.android.com/guide/topics/appwidgets/index.html><https://developer.android.com/guide/in-dex.html>

MODULE 4: Android User Input, Variables, and Operations

Lecture-5

CHAPTER OVERVIEW:

The following pages cover everything about user input, from basic touch input and gestures, to keyboards and game controllers. You can add convenient features such as copy/paste and spell checking to your app, and develop your own text services to offer custom keyboards (Input Method Editors), dictionaries, and spelling checkers that you can distribute to users as applications.

OBJECTIVS:

- 4.1-Android Themes
- 4.2- Simplifying User Input
- 4.3-Declaring Variables
- 4.4- Working With Mathematical Operations
- 4.5- Displaying Android Output
- 4.6- Simple app using the module taught e.g: Calculator

LEARNING OUTCOME:

- Create and apply a style
- Specify the Keyboard Type
- Enable Spelling Suggestions and Other Behaviors
- Specify the Input Method Action
- Provide Auto-complete Suggestions
- Syntax for variable declaration
- Addition/Sum
- Subtraction
- Division
- Multiplication

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Android Themes
- Specify the Keyboard Type

CORE DISCUSSION:

- **Fundamental technology:**

An input method editor (IME) is a user control that enables users to enter text. Android provides an extensible input-method framework that allows applications to provide users alternative input methods, such as on-screen keyboards or even speech input. After installing the desired IMEs, a user can select which one to use from the system settings, and use it across the entire system; only one IME may be enabled at a time.

To add an IME to the Android system, you create an Android application containing a class that extends `InputMethodService`. In addition, you usually create a "settings" activity that passes options to the IME service. You can also define a settings UI that's displayed as part of the system settings.

- **Scope of implementation & case study**
 1. Specify the Keyboard Type
 2. Specify the Input Method Action

4.1-Android Themes :

Styles and themes on Android allow you to separate the details of your app design from the UI structure and behavior, similar to stylesheets in web design.

A *style* is a collection of attributes that specify the appearance for a single View. A style can specify attributes such as font color, font size, background color, and much more.

A *theme* is a type of style that's applied to an entire app, activity, or view hierarchy, not just an individual view. When you apply your style as a theme, every view in the app or activity applies each style attribute that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Styles and themes are declared in a style resource file in `res/values/`, usually named `styles.xml`.

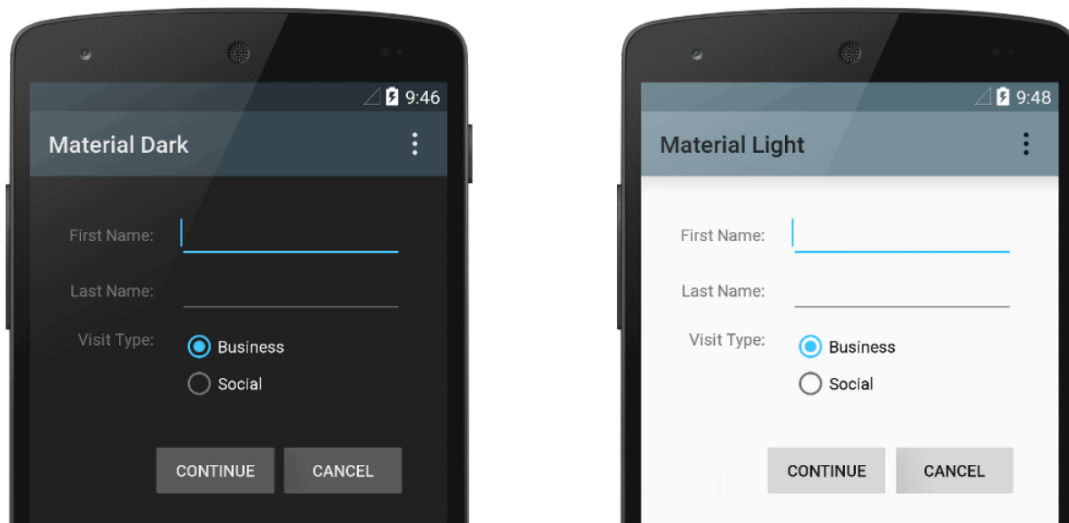


Figure 1. Two themes applied to the same activity: `Theme.AppCompat` (left) and `Theme.AppCompat.Light` (right)

Create and apply a style

To create a new style or theme, open your project's `res/values/styles.xml` file. For each style you want to create, follow these steps:

1. Add a `<style>` element with a name that uniquely identifies the style.
2. Add an `<item>` element for each style attribute you want to define.

The name in each item specifies an attribute you would otherwise use as an XML attribute in your layout. The value in the `<item>` element is the value for that attribute.

For example, if you define the following style:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="GreenText" parent="TextAppearance.AppCompat">
    <item name="android:textColor">#00FF00</item>
  </style>
</resources>
```

You can apply the style to a view as follows:

```
<TextView
  style="@style/GreenText"
  ... />
```

Each attribute specified in the style is applied to that view if the view accepts it. The view simply ignores any attributes that it does not accept.

Note: Only the element to which you add the style attribute receives those style attributes—any child views do not apply the styles. If you want child views to inherit styles, instead apply the style with the `android:theme` attribute.

4.2- Simplifying User Input:

Every text field expects a certain type of text input, such as an email address, phone number, or just plain text. So it's important that you specify the input type for each text field in your app so the system displays the appropriate soft input method (such as an on-screen keyboard).

Beyond the type of buttons available with an input method, you should specify behaviors such as whether the input method provides spelling suggestions, capitalizes new sentences, and replaces the carriage return button with an action button such as a **Done** or **Next**. This lesson shows how to specify these characteristics

Specify the Keyboard Type

You should always declare the input method for your text fields by adding the `android:inputType` attribute to the `<EditText>` element.



Figure 1. The phone input type.

For example, if you'd like an input method for entering a phone number, use the "phone" value:

```
<EditText
  android:id="@+id/phone"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:hint="@string/phone_hint"
  android:inputType="phone"/>
```

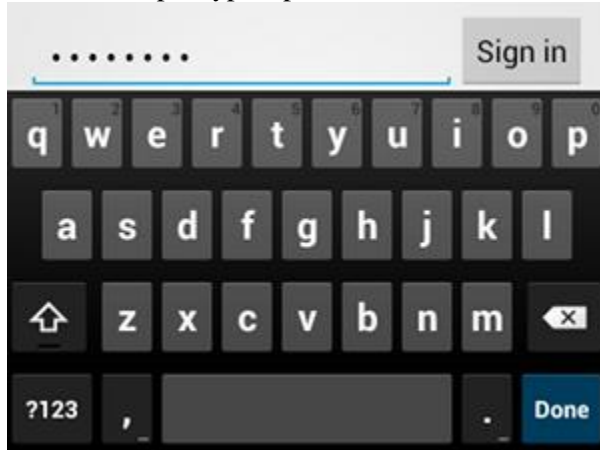


Figure 2. The textPassword input type.

Or if the text field is for a password, use the "textPassword" value so the text field conceals the user's input:

```
<EditText
  android:id="@+id/password"
  android:hint="@string/password_hint"
  android:inputType="textPassword"
  ... />
```

There are several possible values documented with the android:inputType attribute and some of the values can be combined to specify the input method appearance and additional behaviors.

Enable Spelling Suggestions and Other Behaviors

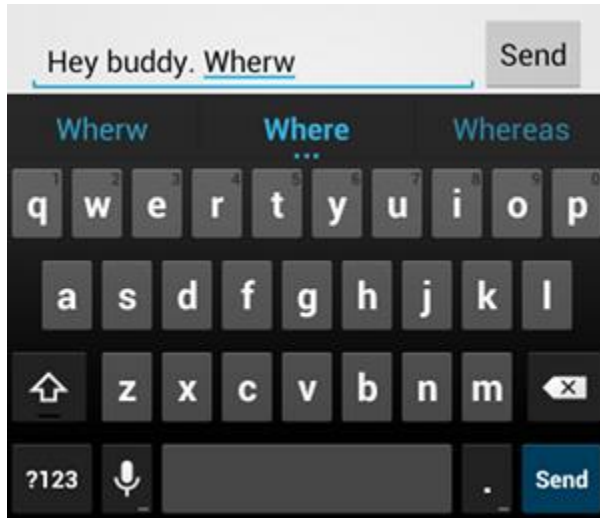


Figure 3. Adding `textAutoCorrect` provides auto-correction for misspellings.

The `android:inputType` attribute allows you to specify various behaviors for the input method. Most importantly, if your text field is intended for basic text input (such as for a text message), you should enable auto spelling correction with the "textAutoCorrect" value.

You can combine different behaviors and input method styles with the `android:inputType` attribute. For example, here's how to create a text field that capitalizes the first word of a sentence and also auto-corrects misspellings:

```
<EditText
    android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType=
        "textCapSentences|textAutoCorrect"
    ... />
```

Specify the Input Method Action

Most soft input methods provide a user action button in the bottom corner that's appropriate for the current text field. By default, the system uses this button for either a **Next** or **Done** action unless your text field allows multi-line text (such as with `android:inputType="textMultiLine"`), in which case the action button is a carriage return. However, you can specify additional actions that might be more appropriate for your text field, such as **Send** or **Go**.

Course Manual for Android App Development

To specify the keyboard action button, use the `android:imeOptions` attribute with an action value such as "actionSend" or "actionSearch". For example:



Figure 4. The Send button appears when you declare `android:imeOptions="actionSend"`.

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend"/>
```

You can then listen for presses on the action button by defining a `TextView.OnEditorActionListener` for the `EditText` element. In your listener, respond to the appropriate IME action ID defined in the `EditorInfo` class, such as `IME_ACTION_SEND`. For example:

```
EditText editText =(EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener(){
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event){
        boolean handled = false;
        if(actionId == EditorInfo.IME_ACTION_SEND){
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

Provide Auto-complete Suggestions

If you want to provide suggestions to users as they type, you can use a subclass of `EditText` called `AutoCompleteTextView`. To implement auto-complete, you must specify an `Adapter` that provides the text suggestions. There are several kinds of adapters available, depending on where the data is coming from, such as from a database or an array.

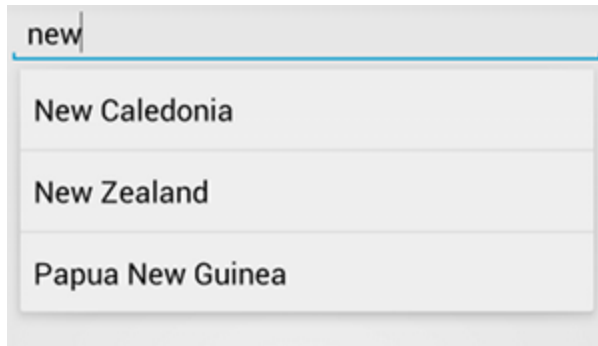


Figure 5. Example of `AutoCompleteTextView` with text suggestions.

The following procedure describes how to set up an `AutoCompleteTextView` that provides suggestions from an array, using `ArrayAdapter`:

1. Add the `AutoCompleteTextView` to your layout. Here's a layout with only the text field:

```
<?xml version="1.0" encoding="utf-8"?>
<AutoCompleteTextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

2. Define the array that contains all text suggestions. For example, here's an array of country names that's defined in an XML resource file (res/values/strings.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Afghanistan</item>
        <item>Albania</item>
        <item>Algeria</item>
        <item>American Samoa</item>
        <item>Andorra</item>
        <item>Angola</item>
        <item>Anguilla</item>
        <item>Antarctica</item>
        ...
    </string-array>
</resources>
```

Every text field expects a certain type of text input, such as an email address, phone number, or just plain text. So it's important that you specify the input type for each text field in your app so the system displays the appropriate soft input method (such as an on-screen keyboard).

Beyond the type of buttons available with an input method, you should specify behaviors such as whether the input method provides spelling suggestions, capitalizes new sentences, and replaces the carriage return button with an action button such as a **Done** or **Next**. This lesson shows how to specify these characteristics

Lecture-6

4.3-Declaring Variables:

Identifiers are the names of variables. They must be composed of only letters, numbers, the underscore, and the dollar sign (\$). They cannot contain white spaces. Identifiers may only begin with a letter, the underscore, or the dollar sign. A variable cannot begin with a number. All variable names are case sensitive.

Syntax for variable declaration

```
datatype1 variable1, datatype2 variable2, ... datatypen variablen;
```

For example:

```
int a, char ch, String string, Boolean boolean;
```

Initialisation

Variables can be assigned values in the following way: `Variablename = value;`

For example;

```
ch='a';  
a=0;  
string="abcd"  
Boolean=true;  
Boolean=0;
```

4.4Working With Mathematical Operations:

CREATE NEW ANDROID PROJECT:

Step 1: Write code into **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"
```

Course Manual for Android App Development

```
android:orientation="vertical" >
```

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:text="Maths Function" />
```

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number" >
```

```
    <requestFocus />
</EditText>
```

```
<EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number" />
```

```
<Button
    android:id="@+id/buttonsum"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Sum/Addition" />
```

```
<Button
    android:id="@+id/buttonsub"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Subtraction" />
```

```
<Button
    android:id="@+id/buttondiv"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Division" />
```

```
<Button
    android:id="@+id/buttonmul"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="multiplication" />
```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

Course Manual for Android App Development

```
android:text="Large Text"
android:textAppearance="?android:attr/textAppearanceLarge" />
```

```
</LinearLayout>
```

Step 2: Write code into **MainActivity.java**

```
package dev.androidapplink.mathsfunapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Create object
        Button btnsum = (Button) findViewById(R.id.buttonsum);
        Button btnsub = (Button) findViewById(R.id.buttonsub);
        Button btndiv = (Button) findViewById(R.id.buttondiv);
        Button btnmul = (Button) findViewById(R.id.buttonmul);
        final EditText etv = (EditText) findViewById(R.id.editText1);
        final EditText etv2 = (EditText) findViewById(R.id.editText2);
        final TextView result = (TextView) findViewById(R.id.textView1);
        // Create button click event
        btnsum.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                int x = new Integer(etv.getText().toString());
                int y = new Integer(etv2.getText().toString());
                int sum = x + y; //Perform Maths operation
                result.setText("The ANS of " + x + " + " + y + " = " + sum); //print answer
            }
        });

        btnsub.setOnClickListener(new OnClickListener() {
```

Course Manual for Android App Development

```
@Override
public void onClick(View v) {
    int x = new Integer(etv.getText().toString());
    int y = new Integer(etv2.getText().toString());
    int sub = x - y; //Perform Maths operation
    result.setText("The ANS of " + x + " - " + y + " = " + sub); //print answer
}
});
```

```
btndiv.setOnClickListener(new OnClickListener() {
```

```
@Override
public void onClick(View v) {
    int x = new Integer(etv.getText().toString());
    int y = new Integer(etv2.getText().toString());
    int div = x / y; //Perform Maths operation
    result.setText("The ANS of " + x + " / " + y + " = " + div); //print answer
}
});
```

```
btnmul.setOnClickListener(new OnClickListener() {
```

```
@Override
public void onClick(View v) {
    int x = new Integer(etv.getText().toString());
    int y = new Integer(etv2.getText().toString());
    int mul = x * y; //Perform Maths operation
    result.setText("The ANS of " + x + " * " + y + " = " + mul); //Print answer
}
});
}
}
```

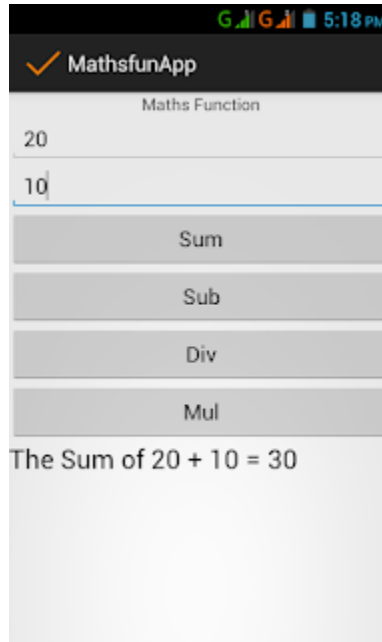
NOTE: No need any special permission in **AndroidManifest.xml**

Lecture-7

4.5- Displaying Android Output:

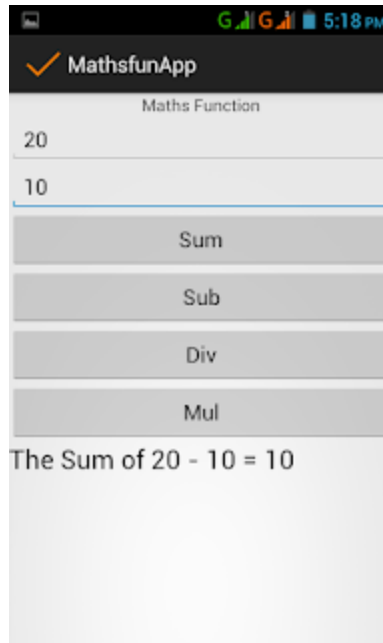
- Step 3: Now Run Your Project:

Addition/Sum:

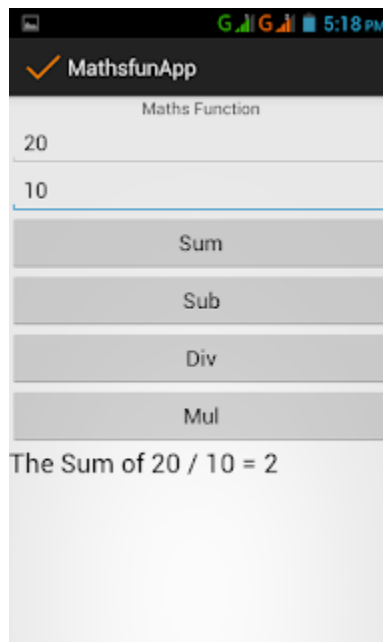


Subtraction:

Course Manual for Android App Development

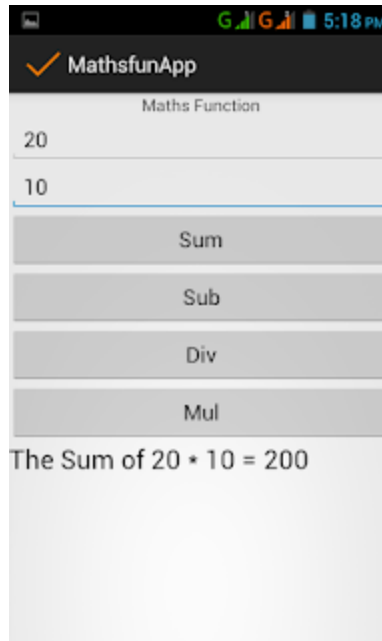


Division:



Multiplication:

Course Manual for Android App Development



Lecture-7,8

4.6- Simple app using the module taught e.g: Calculator:

Make a simple calculator as a class work

PRACTICE:

User Input ->Keyboard Type ->Input Method Action ->variable declaration

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

<https://developer.android.com/guide/topics/text/creating-input-method>

REFARANCE LINK:

https://www.tutorialspoint.com/android/android_resources.htm

MODULE 5: Icon and Decision-Making Controls

CHAPTER OVERVIEW:

- 5.1- The Launcher Icon
- 5.2- RadioButton and RadioGroup Controls
- 5.3- Making Decisions with Conditional Statements
- 5.4- Different Types of Sensors

OBJECTIVES:

- Set Your App Icon
- RadioButton and RadioGroup Controls:
- Making Decisions with Conditional Statements

LEARNING OUTCOME:

- Set the string values
- Code the Main Activity
- Run the application
- The if-then-else Statement
- Switch Case

CORE DISCUSSION:

- **Historical approach:**

Detecting and recognizing brake lights is essential to avoid collisions and accidents caused by a vehicle driving behind another vehicle. This refers to traffic risk, among other things, to man-controlled vehicles as well as autonomous cars. There are many systems supporting drivers, which are also applied in autonomous cars. This paper presents a mobile decision-making system warning against traffic risks related to breaking a preceding vehicle. The new approach is two-stage and real-time one. Unlike other approaches, the presented solution, thanks to proper limitation of a searched area, works effectively on a mobile platform with restricted system resources. The results of experiments conducted on real video sequences on roads in various weather conditions showed high effectiveness of the proposed approach.

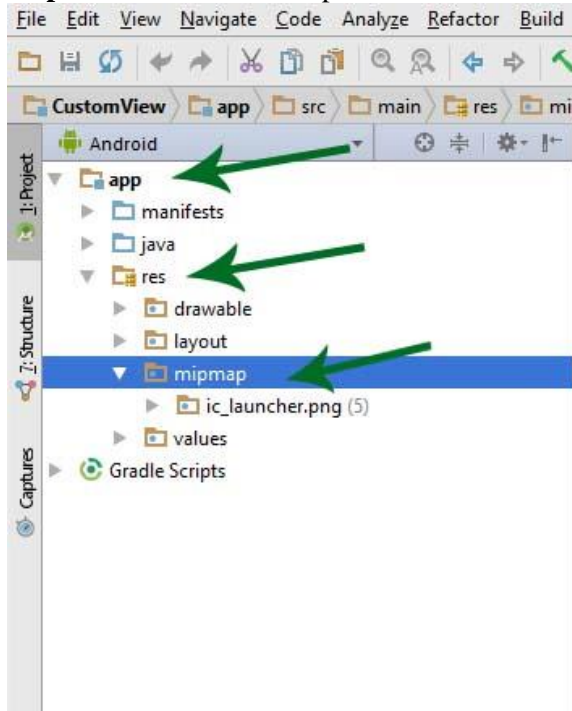
Course Manual for Android App Development

- **Scope of implementation & case study**
 1. Scope of implementation & case study
 2. Making Decision control and Sensors app

5.1- The Launcher Icon:

Step 1- Open your application in Android Studio.

Step 2- Further follow the path to reach the desired folder to add icon (app -> res-> mipmap).



Step 3- Here add you app icon. You can just simply copy and paste the image in mipmap folder.

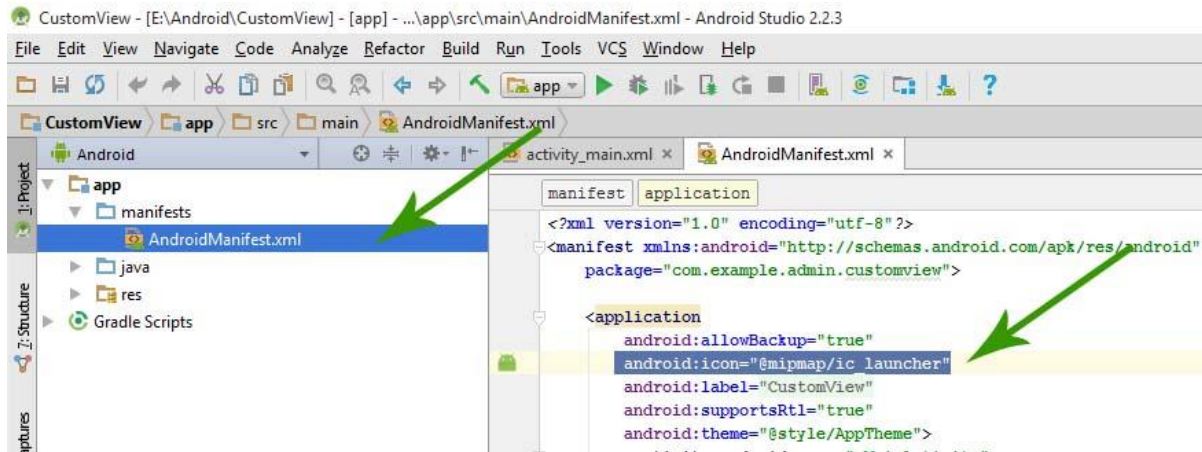
Step 4- After placing the image in the mipmap folder. You need to rename the default icon name to your icon image name.

Step 5- Go to (app -> manifests) open AndroidManifest.xml file. Here find the following code.

`android:icon="@mipmap/ic_launcher"`

Here ic_launcher is the default image name, rename it.

Course Manual for Android App Development



Important Note: You can make any image as your app icon just define a correct path of the image in *AndroidManifest.xml* file.

5.2- RadioButton and RadioGroup Controls:

Open `res/layout/activity_main.xml`, go to the respective tab and paste the following code.
activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
```

```
    android:paddingTop="@dimen/activity_vertical_margin"
```

```
    tools:context=".MainActivity" >
```

```
<TextView
```

```
    android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
android:layout_marginBottom="10dp"
```

```
android:id="@+id/text"
```

```
android:text="@string/ChoiceText" />
```

```
<RadioGroup
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_below="@+id/text"
```

```
    android:id="@+id/myRadioGroup"
```

```
    android:background="#abf234"
```

```
    android:checkedButton="@+id/sound" >
```

```
<RadioButton
```

```
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
```

```
        android:id="@+id/sound"
```

```
        android:text="@string/Sound" />
```

```
<RadioButton
```

```
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
```

Course Manual for Android App Development

```
android:id="@+id/vibration"
```

```
android:text="@string/Vibration" />
```

```
<RadioButton
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:id="@+id/silent"
```

```
    android:text="@string/Silent" />
```

```
</RadioGroup>
```

```
<Button
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_below="@+id/myRadioGroup"
```

```
    android:layout_marginTop="10dp"
```

```
    android:id="@+id/chooseBtn"
```

```
    android:text="@string/Choose" />
```

```
</RelativeLayout>
```

Course Manual for Android App Development

The basic attribute of `RadioGroup` is `android:checkedButton`, that specifies which radio button should be checked by default. The other components are inherited from the `View` class. As you can notice from the code above, the set of the radio buttons are embodied by a `RadioGroup`, so every configuration of its component affects the radio buttons too.

3. Set the string values

At this step we just going to declare the string references from the `activity_main.xml` to the appropriate resource.

So open `res/values/strings.xml`, go to the respective tab and paste the following code.

strings.xml

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

<resources>

    <string name="app_name">RadioGroupExample</string>

    <string name="action_settings">Settings</string>

    <string name="ChoiceText">Choose one of the radio buttons below</string>

    <string name="Sound">Sound</string>

    <string name="Vibration">Vibration</string>

    <string name="Silent">Silent</string>

    <string name="Choose">Choose</string>

</resources>
```

4. Code the Main Activity

At this point we will show how we can handle the change of a radio button, that belongs to a `RadioGroup`.

Open `src/com.javacodegeeks.android.radiogrouppexample/MainActivity.java` and paste the following code.

MainActivity.java:

```
package com.javacodegeeks.android.radiogrouppexample;

import android.os.Bundle;

import android.app.Activity;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.RadioButton;

import android.widget.RadioGroup;

import android.widget.RadioGroup.OnCheckedChangeListener;

import android.widget.TextView;

import android.widget.Toast;

public class MainActivity extends Activity {

    private RadioGroup radioGroup;

    private RadioButton sound, vibration, silent;

    private Button button;
```


Course Manual for Android App Development

```
private TextView textView;

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    radioGroup = (RadioGroup) findViewById(R.id.myRadioGroup);

    radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener() {

        @Override

        public void onCheckedChanged(RadioGroup group, int checkedId) {

            // find which radio button is selected

            if(checkedId == R.id.silent) {

                Toast.makeText(getApplicationContext(), "choice:
Silent",

                Toast.LENGTH_SHORT).show();

            } else if(checkedId == R.id.sound) {

                Toast.makeText(getApplicationContext(), "choice:
Sound",

                Toast.LENGTH_SHORT).show();
```

Course Manual for Android App Development

```
        } else {  
            Toast.makeText(getApplicationContext(), "choice:  
Vibration",  
            Toast.LENGTH_SHORT).show();  
        }  
    }  
});  
  
    sound = (RadioButton) findViewById(R.id.sound);  
    vibration = (RadioButton) findViewById(R.id.vibration);  
    silent = (RadioButton) findViewById(R.id.silent);  
    textView = (TextView) findViewById(R.id.text);  
  
    button = (Button) findViewById(R.id.chooseBtn);  
    button.setOnClickListener(new OnClickListener() {  
  
        @Override  
        public void onClick(View v) {  
            int selectedId = radioGroup.getCheckedRadioButtonId();  
  
            // find which radioButton is checked by id
```

```
        if(selectedId == sound.getId()) {

            textView.setText("You chose 'Sound' option");

        } else if(selectedId == vibration.getId()) {

            textView.setText("You chose 'Vibration' option");

        } else {

            textView.setText("You chose 'Silent' option");

        }

    }

});

}
```

Now let's have a look at the code above. When a checked radio button is changed in its group, `OnCheckedChangeListener` is invoked in order to handle this situation.

The `onCheckedChanged()` method of this interface, includes the unique id of the radio button that was selected and caused the invoke of the callback.

In this example we will show you another way of selecting the choice information (for example when a button is pressed). This can be done through `getCheckedRadioButtonId()`, which is a public function of `RadioGroup` class. This method returns the unique id of the radio button that is chosen from the group. You can have a look at the code to see how you can handle both situations.

Of course Android system provides us a more dynamic way of changing and handling the attributes of the application views. As a prerequisite is to map every view with the unique id component of the XML. This can be done via `findViewById()` method.

5. Run the application

To run our application, right click on our project → Run as → Android Application. The AVD will appear with the app loaded.

5.3- Making Decisions with Conditional Statements:

There are two types of conditional statements.

- If-else

Let's say part of a program we're writing needs to calculate if the purchaser of a ticket is eligible for a child's discount. Anyone under the age of 16 gets a 10% discount on the ticket price.

We can let our program make this decision by using an **if-then** statement:

```
if (age < 16)
    isChild = true;
```

In our program, an integer variable called **age** holds the age of the ticket purchaser. The condition (i.e., is the ticket purchaser under 16) is placed inside the brackets. If this condition is true, then the statement beneath the if statement is executed -- in this case a **boolean** variable **isChild** is set to **true**.

The syntax follows the same pattern every time. The **if** keyword followed by a condition in brackets, with the statement to execute underneath:

```
if (condition is true)
    execute this statement
```

The key thing to remember is the condition must equate to a **boolean** value (i.e., true or false).

Often, a Java program needs to execute more than one statement if a condition is true. This is achieved by using a block (i.e., enclosing the statements in curly brackets):

```
if (age < 16)
{
    isChild = true;
    discount = 10;
}
```

This form of the **if-then** statement is the most commonly used, and it's recommended to use curly brackets even when there is only one statement to execute.

It improves the readability of the code and leads to fewer programming mistakes. Without the curly brackets, it's easy to overlook the effect of the decision being made or to come back later and add another statement to execute but forget to also add the curly brackets.

The if-then-else Statement

The **if-then** statement can be extended to have statements that are executed when the condition is false. The **if-then-else** statement executes the first set of statements if the condition is true, otherwise, the second set of statements are executed:

Course Manual for Android App Development

```
if (condition)
{
execute statement(s) if condition is true
}
else
{
execute statement(s) if condition is false
}
```

In the ticket program, let's say we need to make sure the discount is equal to 0 if the ticket purchaser is not a child:

```
if (age < 16)
{
isChild = true;
discount = 10;
}
else
{
discount = 0;
}
```

The **if-then-else** statement also allows the nesting of **if-then** statements. This allows decisions to follow a path of conditions. For example, the ticket program might have several discounts. We might first test to see if the ticket purchaser is a child, then if they're a pensioner, then if they're a student and so on:

```
if (age < 16)
{
isChild = true;
discount = 10;
}
else if (age > 65)
{
isPensioner = true; discount = 15;
}
else if (isStudent == true)
{
discount = 5;
}
```

As you can see, the **if-then-else** statement pattern just repeats itself. If at any time the condition is **true**, then the relevant statements are executed and any conditions beneath are not tested to see whether they are **true** or **false**.

For example, if the age of the ticket purchaser is 67, then the highlighted statements are executed and the **(isStudent == true)** condition is never tested and the program just continues on.

There is something worth noting about the **(isStudent == true)** condition. The condition is written to make it clear that we're testing whether **isStudent** has a value of true, but because it is a **boolean** variable, we can actually write:

```
else if (isStudent)
{
    discount = 5;
}
```

If this is confusing, the way to think about it is like this -- we know a condition is tested to be true or false.

For integer variables like **age**, we have to write an expression that can be evaluated to true or false (e.g., **age == 12**, **age > 35**, etc..).

However, boolean variables already evaluate to be true or false. We don't need to write an expression to prove it because **if (isStudent)** is already saying "if isStudent is true..". If you want to test that a boolean variable is false, just use the unary operator!. It inverts a boolean value, therefore **if (!isStudent)** is essentially saying "if isStudent is false."

- Switch Case

The **switch** statement provides an effective way to deal with a section of code that could branch in multiple directions based on a *single* variable. It does not support the conditional operators that the **if-then** statement does, nor can it handle multiple variables. It is, however, a preferable choice when the condition will be met by a single variable, because it can improve performance and is easier to maintain.

Here's an example:

```
switch ( single_variable ) {
    case value:
        //code_here;
        break;
    case value:
        //code_here;
        break;
    default:
        //set a default;
}
```

5.4- Different Types of Sensors:

Statement

Most of the android devices have built-in sensors that measure motion, orientation, and various environmental condition. The android platform supports three broad categories of sensors.

- Motion Sensors
- Environmental sensors
- Position sensors

Some of the sensors are hardware based and some are software based sensors. Whatever the sensor is, android allows us to get the raw data from these sensors and use it in our application. For this android provides us with some classes.

Android provides `SensorManager` and `Sensor` classes to use the sensors in our application. In order to use sensors, first thing you need to do is to instantiate the object of `SensorManager` class. It can be achieved as follows.

```
SensorManager sMgr;  
sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
```

The next thing you need to do is to instantiate the object of `Sensor` class by calling the `getDefaultSensor()` method of the `SensorManager` class. Its syntax is given below –

```
Sensor light;  
light = sMgr.getDefaultSensor(Sensor.TYPE_LIGHT);
```

Once that sensor is declared, you need to register its listener and override two methods which are `onAccuracyChanged` and `onSensorChanged`. Its syntax is as follows –

```
sMgr.registerListener(this, light, SensorManager.SENSOR_DELAY_NORMAL);  
public void onAccuracyChanged(Sensor sensor, int accuracy) {  
}  
  
public void onSensorChanged(SensorEvent event) {  
}
```

Getting list of sensors supported

You can get a list of sensors supported by your device by calling the `getSensorList` method, which will return a list of sensors containing their name and version number and much more information. You can then iterate the list to get the information. Its syntax is given below –

```
sMgr =(SensorManager)this.getSystemService(SENSOR_SERVICE);  
  
List<Sensor> list = sMgr.getSensorList(Sensor.TYPE_ALL);
```

Course Manual for Android App Development

```
for(Sensor sensor: list){  
}
```

Apart from the these methods, there are other methods provided by the `SensorManager` class for managing sensors framework. These methods are listed below –

Sr.No	Method & description
1	<code>getDefaultSensor(int type)</code> This method get the default sensor for a given type.
2	<code>getOrientation(float[] R, float[] values)</code> This method returns a description of the current primary clip on the clipboard but not a copy of its data.
3	<code>getInclination(float[] I)</code> This method computes the geomagnetic inclination angle in radians from the inclination matrix.
4	<code>registerListener(SensorListener listener, int sensors, int rate)</code> This method registers a listener for the sensor
5	<code>unregisterListener(SensorEventListener listener, Sensor sensor)</code> This method unregisters a listener for the sensors with which it is registered.
6	<code>getOrientation(float[] R, float[] values)</code> This method computes the device's orientation based on the rotation matrix.
7	<code>getAltitude(float p0, float p)</code> This method computes the Altitude in meters from the atmospheric pressure and the pressure at sea level.

Course Manual for Android App Development

Example

Here is an example demonstrating the use of `SensorManager` class. It creates a basic application that allows you to view the list of sensors on your device.

To experiment with this example, you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Android studio to create an Android application under a package <code>com.example.sairamkrishna.myapplication</code> .
2	Modify <code>src/MainActivity.java</code> file to add necessary code.
3	Modify the <code>res/layout/activity_main</code> to add respective XML components.
4	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified **MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.hardware.SensorManager;
import android.os.Bundle;

import android.util.Log;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
```

Course Manual for Android App Development

```
import android.widget.TextView;

import java.util.List;

import android.hardware.Sensor;
import android.hardware.SensorManager;

public class MainActivity extends Activity {
    TextView tv1 = null;
    private SensorManager mSensorManager;

    @Override

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        tv1 = (TextView) findViewById(R.id.textView2);
        tv1.setVisibility(View.GONE);

        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        List<Sensor> mList = mSensorManager.getSensorList(Sensor.TYPE_ALL);

        for (int i = 1; i < mList.size(); i++) {
            tv1.setVisibility(View.VISIBLE);

            tv1.append("\n" + mList.get(i).getName() + "\n" + mList.get(i).getVendor() + "\n" +
mList.get(i).getVersion());
        }
    }
}
```

```
@Override  
  
publicboolean onCreateOptionsMenu(Menu menu){  
  
    // Inflate the menu; this adds items to the action bar if it is present.  
  
    getMenuInflater().inflate(R.menu.menu_main, menu);  
  
    returntrue;  
  
}
```

```
@Override  
  
publicboolean onOptionsItemSelected(MenuItem item){  
  
    // Handle action bar item clicks here. The action bar will  
  
    // automatically handle clicks on the Home/Up button, so long  
  
    // as you specify a parent activity in AndroidManifest.xml.  
  
  
    int id = item.getItemId();  
  
  
  
    //noinspection SimplifiableIfStatement  
    if(id == R.id.action_settings){  
  
        returntrue;  
  
    }  
  
    returnsuper.onOptionsItemSelected(item);  
  
}  
  
}
```

Following is the modified content of the xml **activity_main.xml**.

In the below code **abc** indicates about the logo of ictdivision.com

```
<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"  
  
    xmlns:tools="http://schemas.android.com/tools"android:layout_width="match_parent"
```

```
android:layout_height="match_parent"android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity"
android:transitionGroup="true">
```

```
<TextViewandroid:text="Sensor "android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/textview"
android:textSize="35dp"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"/>
```

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Tutorials point"
android:id="@+id/textView"
android:layout_below="@+id/textview"
android:layout_centerHorizontal="true"
android:textColor="#ff7aff24"
android:textSize="35dp"/>
```

```
<ImageView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

Course Manual for Android App Development

```
android:id="@+id/imageView"

android:src="@drawable/abc"

android:layout_below="@+id/textView"

android:layout_centerHorizontal="true"

android:theme="@style/Base.TextAppearance.AppCompat"/>

<TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="New Text"

android:id="@+id/textView2"

android:layout_below="@+id/imageView"

android:layout_alignParentBottom="true"

android:layout_alignParentRight="true"

android:layout_alignParentEnd="true"

android:layout_alignParentLeft="true"

android:layout_alignParentStart="true"/>

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

```
<resources>

<stringname="app_name">My Application</string>

<stringname="hello_world">Hello world!</string>

<stringname="action_settings">Settings</string>
```

Course Manual for Android App Development

```
</resources>
```

Following is the content of **AndroidManifest.xml** file.

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sairamkrishna.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">

            <intent-filter>

                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>

            </intent-filter>

        </activity>

    </application>

</manifest>
```

Let's try to run our application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and



Course Manual for Android App Development

click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window –

Now if you will look at your device screen, you will see the list of sensors supported by your device along with their name and version and other information.

If you would run this application on different devices, the output would be different because the output depends upon the number of sensors supported by your device.

PRACTICE:

Launcher Icon -> RadioGroup Controls -> Conditional Statements-> Different Types of Sensors

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

https://nanopdf.com/download/9781133597209pptch04_pdf

REFARANCE LINK:

<http://abhiandroid.com/androidstudio/change-icon-android-studio.html>

<http://www.vogella.com/tutorials/AndroidSensor/article.html>

Course Manual for Android App Development

MODULE 6: Android List, arrays, Web Browsers

CHAPTER OVERVIEW:

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

6.1-Steps to set up the ListView item listener

6.2- Dialog Box Content

6.3-Implement onItemClick

6.4- Decision Structure – Switch Statement

6.5- Android Intents

6.6- Launching the Browser from an Android Device

6.7- Building layouts including xml , views, ViewGroups

6.8- Adding Multiple Class File

6.5- Simple app using the module taught. E.g.: Simple MediaPlayer

OBJECTIVES:

- Create your own custom ListView, Dialog Box
- layouts including xml , views, ViewGroups

LEARNING OUTCOME:

- ListView Layout
- ListView Item Layout

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Launching the Browser from an Android Device

CORE DISCUSSION:

- Fundamental technology

Android Studio : Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

- Scope of implementation & case study

1. Create A new Browser App

6.1-Steps to set up the ListView item listener :

1. Create your own custom ListView. Your activity should extend `ListActivity`.
2. Override `onListItemClick`.
3. In `onListItemClick` implementation, create a dialog box, set its content view and then show the dialog.
4. The dialog box should contain the illustrator's name and picture.

ListView Layout

If you are new to creating list view, read my article on building [customized ListView](#). Below XML is the layout of ListView.

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"

    android:id="@+id/taskListParent"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    android:orientation="vertical">
```

```
<ListView  
    android:id="@android:id/list"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
</ListView>
```

```
</LinearLayout>
```

ListView Item Layout

List Item will show the cartoon character, some description about the character and the character's name. Below XML describes its layout.

cartoon_list_item_view.xml:

```
<?xmlversion="1.0"encoding="utf-8"?>  
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">  
  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="horizontal">  
  
        <ImageView  
            android:id="@+id/cartoon_pic"  
            android:layout_width="wrap_content"  
            android:layout_height="fill_parent"  
            android:layout_marginRight="6dip"  
            android:layout_weight="0"/>
```

Course Manual for Android App Development

```
<TextView  
    android:id="@+id/about_cartoon"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/name"  
    android:layout_weight="1"/>
```

```
</LinearLayout>
```

```
<TextView  
    android:id="@+id/name"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/name"/>
```

```
</LinearLayout>
```

Here is our ListActivity class. In our activity class, we set the content view to ListView's layout and then attach the model.

CartoonListActivity:

```
packagecom.javarticles.android;  
  
importjava.util.ArrayList;  
importjava.util.List;  
  
importandroid.app.Dialog;  
importandroid.app.ListActivity;  
importandroid.os.Bundle;  
importandroid.view.View;  
importandroid.view.View.OnClickListener;  
importandroid.widget.Button;
```

Course Manual for Android App Development

```
import android.widget.ListView;
import android.widget.TextView;

public class CartoonListActivity extends ListActivity {
    private CartoonListAdapter cartoonListAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.cartoon_list_view);
        List<Cartoon> cartoonList = new ArrayList<>();
        cartoonList
            .add(new Cartoon(
                "Tintin",
                R.drawable.tintin_thumb,
                "Tintin is a fictional character in The Adventures of Tintin, the comics series by Belgian cartoonist Hergé.",
                "Herge", R.drawable.herge));
        cartoonList
            .add(new Cartoon(
                "Asterix",
                R.drawable.asterix_thumb,
                "Asterix is a fictional character, the titular hero of the French comic book series The Adventures of Asterix. The series portrays him as a diminutive but fearless Gaulish warrior living in the time of Julius Caesar's Gallic Wars",
                "Abbert Underzo", R.drawable.albbert_uderzo));
        cartoonListAdapter = new CartoonListAdapter(this,
            R.layout.cartoon_list_item_view, cartoonList);
        setListAdapter(cartoonListAdapter);
    }
}
```

Our Model contains simple beans of Cartoon class.

CartoonListAdapter:

```
package com.javarticles.android;
```

```
import java.util.List;
```

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;
```

```
public class CartoonListAdapter extends ArrayAdapter {
    private List<Cartoon> cartoonItems;
    private Context context;

    public CartoonListAdapter(Context context, int textViewResourceId, List<Cartoon> cartoonItems) {
```

Course Manual for Android App Development

```
super(context, textViewResourceId, cartoonItems);
this.context = context;
this.cartoonItems = cartoonItems;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {

    View view = convertView;
    if (view == null) {
        LayoutInflater vi = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        view = vi.inflate(R.layout.cartoon_list_item_view, null);
    }

    Cartoon cartoon = cartoonItems.get(position);

    if (cartoon != null) {
        // name
        TextView nameTextView = (TextView) view.findViewById(R.id.name);
        nameTextView.setText(cartoon.getName());

        // thumb image
        ImageView imageView = (ImageView) view.findViewById(R.id.cartoon_pic);
        imageView.setImageResource(cartoon.getCartoonPicRes());

        TextView aboutTextView = (TextView) view.findViewById(R.id.about_cartoon);
        aboutTextView.setText(cartoon.getAbout());
    }

    return view;
}
}
```

6.2- Dialog Box Content :

Now comes the part, we are most interested in, that is, to display the illustrator details on selecting the item.

Our dialog will contain a text field and a 'close' dialog button. We will set the illustrator's name and picture dynamically in the `onListItemClick()` as soon as the list item is selected.

author_dialog.xml:

```
?xmlversion="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="vertical">
```

```
<TextView  
    android:id="@+id/author_name"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/illustrator"/>
```

```
<Button  
    android:id="@+id/close_author_dialog_button"  
    style="?android:attr/buttonStyleSmall"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/close"/>
```

```
</LinearLayout>
```

6.3-Implement onListItemClick :

In onListItemClick, based on the list item selected, we will first get the Cartoon bean the row represents. Next step would be to create a customized dialog. We will create a Dialog Object and set its title. The title would be manufactured based on the cartoon character's name.

Once we create the dialog object, we need to set its content view to author_dialog layout.

Next step would be to populate the dialog fields. We need to find the illustrator's text widget from the dialog's view and set the illustrator's name. We also need to set the picture of the illustrator on the same text field, using textView.setCompoundDrawablesWithIntrinsicBounds.

The dialog box has a 'Close' button to close the dialog. We will set a listener to the 'Close' button to close the dialog using dialog.dismiss(). The final step would be to show the dialog using dialog.show().

CartoonListActivity:

```
packagecom.javarticles.android;  
  
importjava.util.ArrayList;  
importjava.util.List;  
  
importandroid.app.Dialog;  
importandroid.app.ListActivity;
```

Course Manual for Android App Development

```
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

public class CartoonListActivity extends ListActivity {
    private CartoonListAdapter cartoonListAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.cartoon_list_view);
        List<Cartoon> cartoonList = new ArrayList<>();
        cartoonList
            .add(new Cartoon(
                "Tintin",
                R.drawable.tintin_thumb,
                "Tintin is a fictional character in The Adventures of Tintin, the comics series by Belgian cartoonist Hergé.",
                "Herge", R.drawable.herge));
        cartoonList
            .add(new Cartoon(
                "Asterix",
                R.drawable.asterix_thumb,
                "Asterix is a fictional character, the titular hero of the French comic book series The Adventures of Asterix. The series portrays him as a diminutive but fearless Gaulish warrior living in the time of Julius Caesar's Gallic Wars",
                "Abbert Underzo", R.drawable.albbert_uderzo));
        cartoonListAdapter = new CartoonListAdapter(this,
            R.layout.cartoon_list_item_view, cartoonList);
        setListAdapter(cartoonListAdapter);
    }

    @Override
    protected void onItemClick(ListView l, View v, int position, long id) {
        super.onItemClick(l, v, position, id);

        Cartoon cartoon = cartoonListAdapter.getItem((int) position);
        displayIllustratorInDialog(cartoon);
    }

    private void displayIllustratorInDialog(Cartoon cartoon) {
        final Dialog dialog = new Dialog(this);
        dialog.setTitle(cartoon.getName() + "'s "
            + getString(R.string.illustrator));
        dialog.setContentView(R.layout.author_dialog);
        TextView author = (TextView) dialog.findViewById(R.id.author_name);
        author.setCompoundDrawablesWithIntrinsicBounds(0,
            cartoon.getAuthorPicRes(), 0, 0);
    }
}
```

Course Manual for Android App Development

```
author.setText(cartoon.getAuthorName());
finalButton closeButton = (Button) dialog
    .findViewById(R.id.close_author_dialog_button);

closeButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        dialog.dismiss();
    }
});
dialog.show();
}

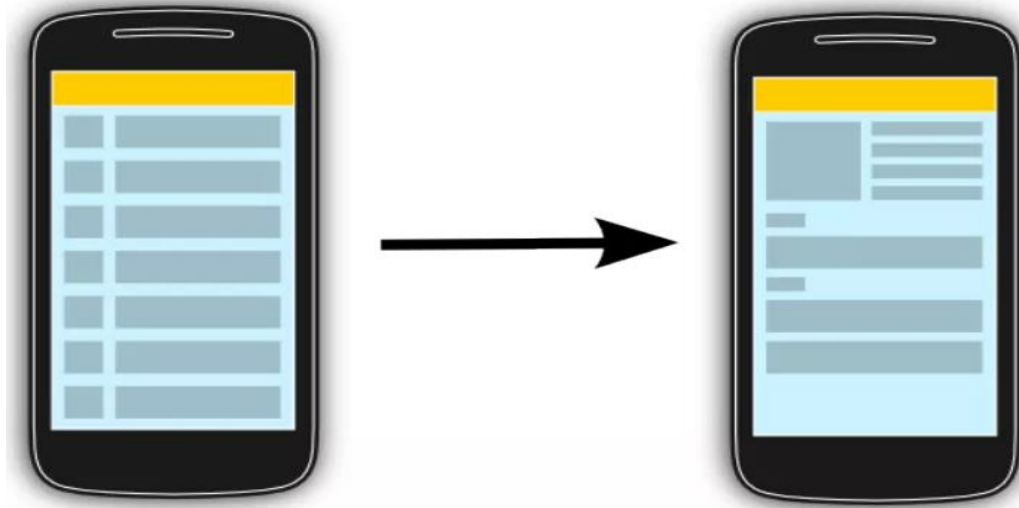
}
```

Run the application and select the list Item

6.4- Decision Structure – Switch Statement :

```
        int i =2;
switch(i){
    case1:{
        Toast.makeText(MainActivity.this, "press
        1",Toast.LENGTH_SHORT).show();
        break;
    }
    case2:{
        Toast.makeText(MainActivity.this, "press
        2",Toast.LENGTH_SHORT).show();
        break;
    }
    case3:{
        Toast.makeText(MainActivity.this, "press
        3",Toast.LENGTH_SHORT).show();
        break;
    }
    default:{
        Toast.makeText(MainActivity.this, "nothing
        press",Toast.LENGTH_SHORT).show();
    }
}
} // END of switch
```


6.5- Android Intents :



Android application components can connect to other Android applications. This connection is based on a task description represented by an Intent object.

Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

```
Intent i = new Intent(this, ActivityTwo.class);  
startActivity(i);
```

6.6- Launching the Browser from an Android Device

Example:

```
import android.content.Intent;  
  
import android.net.Uri;  
  
import android.support.v7.app.AppCompatActivity;  
  
import android.os.Bundle;
```

Course Manual for Android App Development

```
import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button b1, b2;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        b1 = (Button) findViewById(R.id.button);

        b1.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                Intent i = new Intent(android.content.Intent.ACTION_VIEW,

                Uri.parse("http://www.example.com"));

                startActivity(i);

            }

        });

        b2 = (Button) findViewById(R.id.button2);

        b2.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                Intent i = new Intent(android.content.Intent.ACTION_VIEW,
```

Course Manual for Android App Development

```
Uri.parse("tel:9510300000"));

        startActivity(i);
    }
});
}
}
```

Following will be the content of **res/layout/activity_main.xml** file –

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

<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Intent Example"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
android:textSize="30dp"/>
```

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point"
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"/>

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true"/>

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton"/>
```

Course Manual for Android App Development

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Browser"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignEnd="@+id/textView1"
    android:layout_alignLeft="@+id/imageButton"
    android:layout_alignStart="@+id/imageButton"/>
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Phone"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_alignLeft="@+id/button"
    android:layout_alignStart="@+id/button"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2"/>
</RelativeLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<stringname="app_name">My Applicaiton</string>
```

```
</resources>
```

Following is the default content of **AndroidManifest.xml** –

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.saira_000.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

6.7- Building layouts including xml , views, ViewGroups :

- **LinearLayout** – Positions child views in a single row or column depending on the orientation selected. A weight value can be set on each child to specify how much of the layout space that child should occupy relative to other children.

- **TableLayout** – Arranges child views into a grid format of rows and columns. Each row within a table is represented by a TableRow object child, which, in turn, contains a view object for each cell.
- **FrameLayout** – The purpose of the FrameLayout is to allocate an area of screen, typically for the purposes of displaying a single view. If multiple child views are added they will, by default, appear on top of each other positioned in the top left hand corner of the layout area. Alternate positioning of individual child views can be achieved by setting gravity values on each child. For example, setting a center_vertical gravity on a child will cause it to be positioned in the vertical center of the containing FrameLayout view.
- **RelativeLayout** – Probably the most powerful and flexible of the layout managers, this allows child views to be positioned relative both to each other and the containing layout view through the specification of alignments and margins on child views. For example, child View A may be configured to be positioned in the vertical and horizontal center of the containing RelativeLayout view. View B, on the other hand, might also be configured to be centered horizontally within the layout view, but positioned 30 pixels above the top edge of View A, thereby making the vertical position relative to that of View A. The RelativeLayout manager can be of particular use when designing a user interface that must work on a variety of screen sizes and orientations.
- **AbsoluteLayout** – Allows child views to be positioned at specific X and Y coordinates within the containing layout view. Use of this layout is discouraged since it lacks the flexibility to respond to changes in screen size and orientation.
- **GridLayout** – The GridLayout is a relatively new layout manager that was introduced as part of Android 4.0. A GridLayout instance is divided by invisible lines that form a grid containing rows and columns of cells. Child views are then placed in cells and may be configured to cover multiple cells both horizontally and vertically allowing a wide range of layout options to be quickly and easily implemented. Gaps between components in a GridLayout may be implemented by placing a special type of view called a Space view into adjacent cells, or by setting margin parameters.

6.8- Adding Multiple Class File :

Carefully follow my steps to work with the multiple activities and navigate the activities in an Android Studio. I have included the source code below.

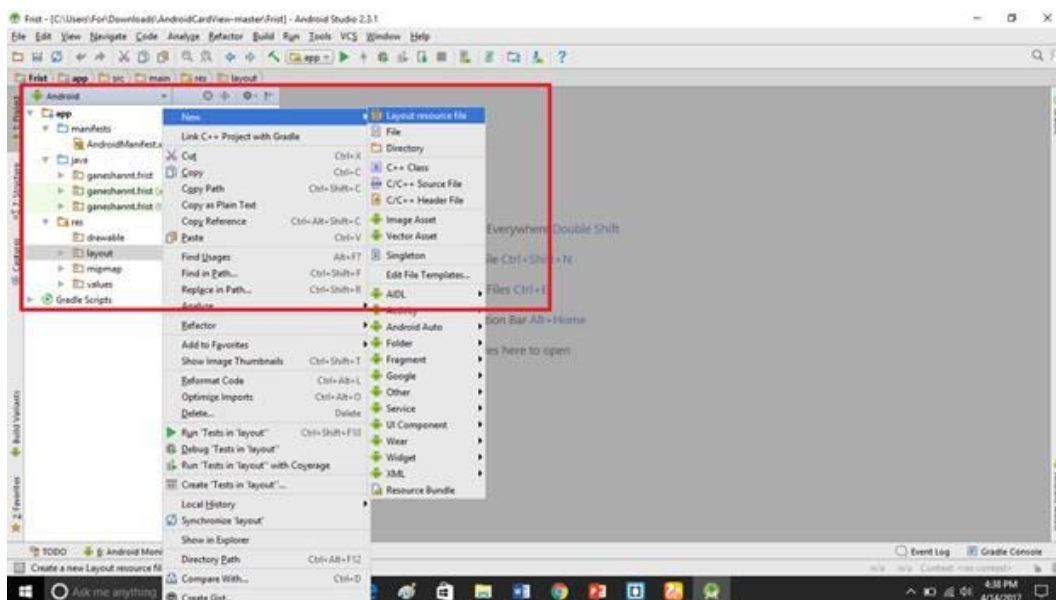
activity_first.xml code

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     xmlns:app="http://schemas.android.com/apk/res-auto"
6.     android:layout_width="match_parent"
7.     android:layout_height="match_parent"
8.     tools:context="ganeshannt.frist.FristActivity">
9.
10.    <Button
11.        android:id="@+id/button2"
12.        android:layout_width="wrap_content"
13.        android:layout_height="wrap_content"
14.        android:onClick="Ganesh"
```

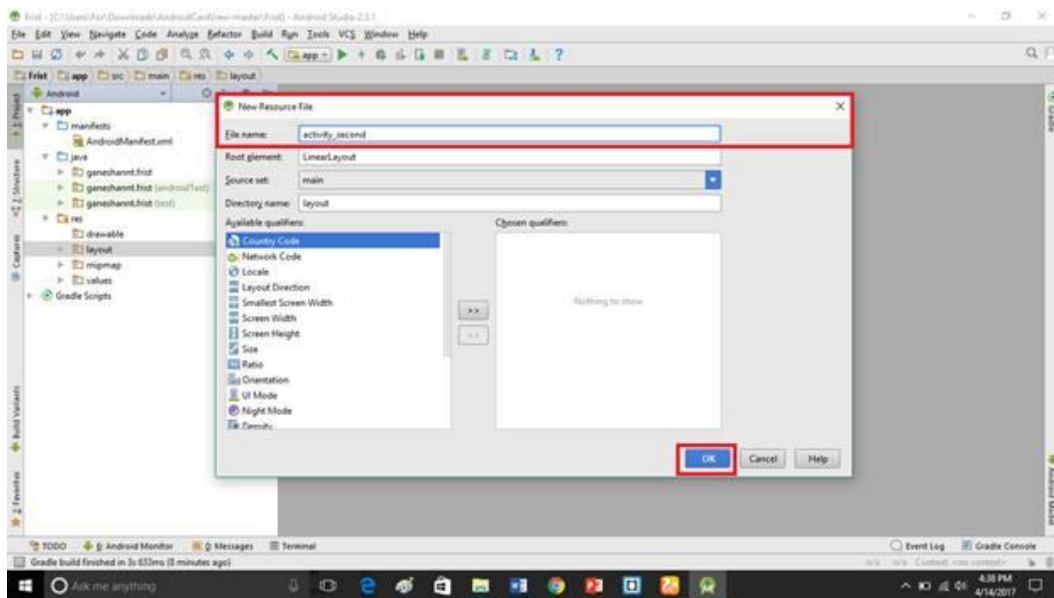
Course Manual for Android App Development

```
15.    android:text="click third activity"
16.    android:textColor="@color/colorPrimary"
17.    app:layout_constraintTop_toTopOf="parent"
18.    tools:layout_editor_absoluteX="168dp"
19.    android:layout_alignParentBottom="true"
20.    android:layout_toEndOf="@+id/text"
21.    android:layout_marginBottom="196dp" />
22.
23.    <TextView
24.        android:layout_width="wrap_content"
25.        android:layout_height="wrap_content"
26.        android:text="This s my first app!"
27.        android:id="@+id/text"
28.        tools:layout_editor_absoluteY="8dp"
29.        tools:layout_editor_absoluteX="8dp" />
30.    <Button
31.        android:layout_width="wrap_content"
32.        android:layout_height="wrap_content"
33.        android:id="@+id/button"
34.        android:text="click second activity"
35.        android:textColor="@color/colorPrimary"
36.        android:onClick="Ganesh"
37.        tools:layout_editor_absoluteX="168dp"
38.        app:layout_constraintTop_toTopOf="parent"
39.        android:layout_above="@+id/button2"
40.        android:layout_alignStart="@+id/button2"
41.        android:layout_marginBottom="40dp" />
42.
43. </RelativeLayout>
```

Create the new activity_second.xml file. (XML File directory :App -> res-> layout). Put your activity name, followed by clicking OK.



Course Manual for Android App Development



Go to activity_second.xml, followed by clicking the text bottom. This XML file contains the designing code for an Android app. In activity_second.xml, copy and paste the code given below.

activity_second.xml code

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:orientation="vertical" android:layout_width="match_parent"
4.     android:layout_height="match_parent">
5.
6.     <TextView
7.         android:layout_width="wrap_content"
8.         android:layout_height="wrap_content"
9.         android:layout_margin="20pt"
10.        android:text="second activity is working..."
11.        android:textAllCaps="true"
12.        android:textColor="@color/colorPrimaryDark"/>
13.
14. </LinearLayout>
```

Repeat the step 7 and create an activity_third.xml. Go to activity_third.xml, followed by clicking text bottom. This XML file contains designing the code for an Android app. In an activity_third.xml, copy and paste the code given below.

activity_third.xml code

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:orientation="vertical" android:layout_width="match_parent"
4.     android:layout_height="match_parent">
5.
```

Course Manual for Android App Development

```
6. <TextView
7.     android:layout_width="wrap_content"
8.     android:layout_height="wrap_content"
9.     android:layout_margin="20pt"
10.    android:text="Third activity is working ....."
11.    android:textAllCaps="true"
12.    android:textColor="@color/colorPrimary"
13. />
14.
15. </LinearLayout>
```

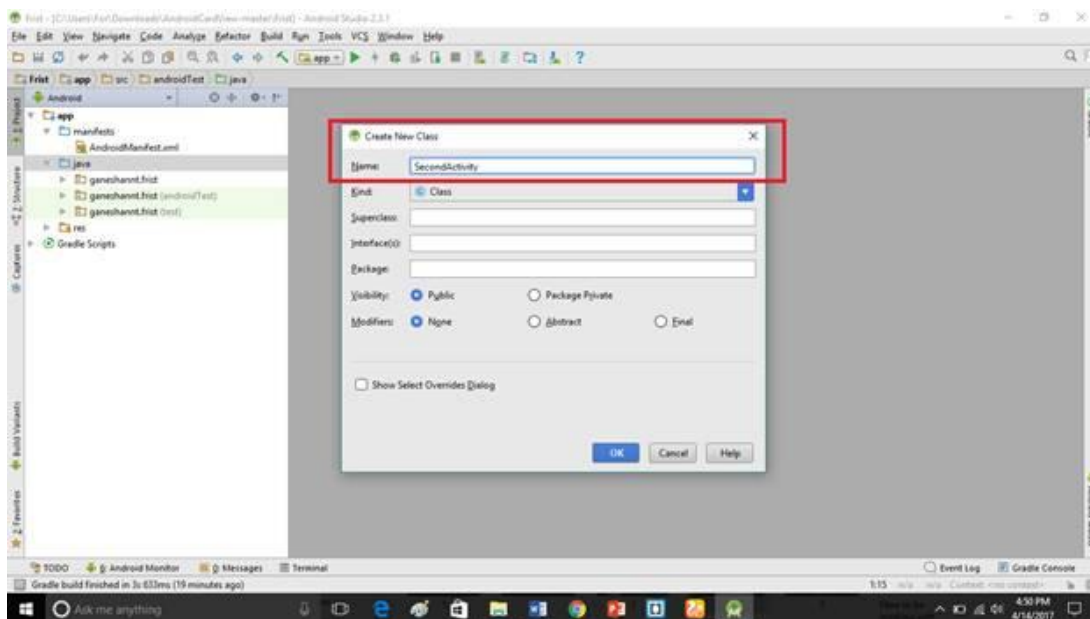
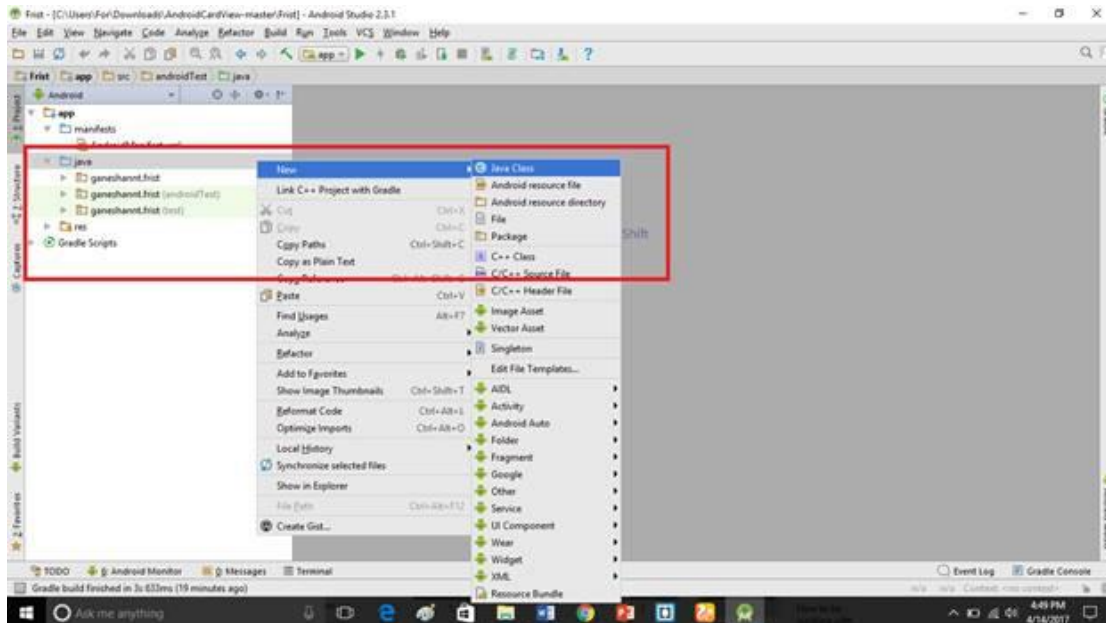
In FirstActivity.java, copy and paste the code given below. Java programming is the backend language for an Android. Do not replace your package name, else an app will not run. The code given below contains my package name. FirstActivity.java code is given below.

```
1. package ganeshannt.frist;
2.
3. import android.content.Intent;
4. import android.os.Bundle;
5. import android.support.v7.app.AppCompatActivity;
6. import android.view.View;
7. import android.widget.Button;
8. import android.widget.TextView;
9.
10. public class FristActivity extends AppCompatActivity {
11.     TextView textView;
12.
13.     @Override
14.     protected void onCreate(Bundle savedInstanceState) {
15.         super.onCreate(savedInstanceState);
16.         setContentView(R.layout.activity_frist);
17.         textView = (TextView) findViewById(R.id.text);
18.
19.     }
20.
21.     public void Ganesh(View View)
22.     {
23.         String button_text;
24.         button_text = ((Button)View).getText().toString();
25.         if(button_text.equals("click second activity"))
26.         {
27.             Intent ganesh = new Intent(this,SecondActivity.class);
28.             startActivity(ganesh);
29.         }
```

Course Manual for Android App Development

```
30.     else if (button_text.equals("click third activity"))
31.     {
32.         Intent mass = new Intent(this,ThirdActivity.class);
33.         startActivity(mass);
34.
35.     }
36. }
37. }
```

Create the new SecondActivity.java file. (Java file directory :App -> Java-> your package name). Put your class name, followed by clicking OK.



Course Manual for Android App Development

In SecondActivity.java, copy and paste the code given below. Do not replace your package name, else an app will not run. The code given below contains my package name.

SecondActivity.java code

```
1. package ganeshannt.frist;
2.
3. import android.app.Activity;
4. import android.os.Bundle;
5. import android.support.annotation.Nullable;
6.
7. /**
8.  * Created by For on 4/14/2017.
9.  */
10.
11. public class SecondActivity extends Activity {
12.     @Override
13.     protected void onCreate(@Nullable Bundle savedInstanceState) {
14.         super.onCreate(savedInstanceState);
15.         setContentView(R.layout.activity_second);
16.
17.     }
18. }
```

Repeat the step 11 and create ThirdActivity.java. Go to ThirdActivity.java, followed by copying and pasting the code given below.

ThirdActivity.java

```
1. package ganeshannt.frist;
2.
3. import android.app.Activity;
4. import android.os.Bundle;
5. import android.support.annotation.Nullable;
6.
7. /**
8.  * Created by For on 4/14/2017.
9.  */
10.
11. public class ThirdActivity extends Activity {
12.     @Override
13.     protected void onCreate(@Nullable Bundle savedInstanceState) {
14.         super.onCreate(savedInstanceState);
15.         setContentView(R.layout.activity_third);
16.
17.     }
18. }
```

Course Manual for Android App Development

Add the code given below into androidminifest.xml within the Application tab. Do not replace all the code, just copy and paste the code given below.

Androidminifest.xml

1. `<activity android:name=".SecondActivity"></activity>`
2. `<activity android:name=".ThirdActivity"></activity>`

6.5- Simple app using the module taught. E.g.: Simple MediaPlayer

Working with as a class work

PRACTICE:

ListView -> Dialog Box -> Switch Statement -> Android Intents details

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<http://www.vogella.com/tutorials/AndroidIntent/article.html>

https://www.tutorialspoint.com/android/android_alert_dialoges.htm

Course Manual for Android App Development

MODULE 7: Implementing Audio in Android apps

CHAPTER OVERVIEW:

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using MediaPlayer APIs

7.1- Creating a Splash Screen

7.2- Playing Music

7.3- Creating a raw folder for music Files

7.4- Using the MediaPlayer Class

7.5- Simple app using the module taught. E.g.: Simple MediaPlayer

OBJECTIVS:

- AndroidCreating a Splash Screen.
- Use Media Player Class

LEARNING OUTCOME:

- Simple MediaPlayer App

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Splash Screen

CORE DISCUSSION:

- Scope of implementation & case study

Simple MediaPlayer

7.1- Creating a Splash Screen:

Step 1 : Create a new project and name it Splashscreen

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we simply added a code to display layout after Splash screen.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="abhiandroid.com.splashscreen.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World by AbhiAndroid!"
        android:textSize="20sp"
        android:layout_centerInParent="true"/>

</RelativeLayout>
```

Step 3: Create a new XML file splashfile.xml for Splash screen and paste the following code in it.

This layout contains your app logo or other product logo that you want to show on splash screen.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

Course Manual for Android App Development

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/splashBackground">
```

```
<ImageView
    android:id="@+id/logo_id"
    android:layout_width="250dp"
    android:layout_height="250dp"
    android:layout_centerInParent="true"
    android:src="@drawable/abhiandroid"/>
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/logo_id"
    android:layout_centerHorizontal="true"
    android:text="Splash Screen"
    android:textSize="30dp"
    android:textColor="@color/blue"/>
```

```
</RelativeLayout>
```

Step 4: Now open app -> java -> package -> MainActivity.java and add the below code.

```
package abhiandroid.com.splashscreen;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
```



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

}
}
```

Step 5: For Splash Screen we will create a separate splash activity. Create a new class in your java package and name it as **SplashActivity.java**.

Step 6: Add this code in **SplashActivity.java** activity. In this code handler is used to hold the screen for specific time and once the handler is out, our main Activity will be launched. We are going to hold the Splash screen for three second's. We will define the seconds in millisecond's after Post Delayed(){} method.

1 second =1000 milliseconds.

Post Delayed method will delay the time for 3 seconds. After the delay time is complete, then your main activity will be launched.

SplashActivity.java

```
package abhiandroid.com.splashscreen;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;

/**
 * Created by AbhiAndroid
 */

public class SplashActivity extends Activity {

    Handler handler;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.splashfile);

    handler=new Handler();
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            Intent intent=new Intent(SplashActivity.this,MainActivity.class);
            startActivity(intent);
            finish();
        }
    },3000);
}
}
```

Step 7: Open AndroidManifest.xml file and make your splashactivity.java class as Launcher activity and mention the Main Activity as another activity.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="abhiandroid.com.splashscreen">

<application
    android:allowBackup="true"
    android:icon="@drawable/abhiandroid"
    android:label="@string/app_name"
    android:supportRtl="true"
    android:theme="@style/AppTheme">
<activity android:name="abhiandroid.com.splashscreen.SplashActivity">
```

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

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name="abhiandroid.com.splashscreen.MainActivity"/>
</application>

</manifest>
```

Designing the main.xml file

Example :

```
<?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:background="#FFCDD2"
        android:orientation="vertical"
        android:padding="16dp">

        <TextView
            android:id="@+id/contact_form_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginBottom="16dp"
            android:layout_marginTop="5dp"
```

Course Manual for Android App Development

```
        android:text="Viral Android"
        android:textColor="#ce3232"
        android:textSize="40sp"
        android:typeface="serif" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/colorPrimaryDark"
        android:hint="Name"
        android:inputType="textPersonName" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:layout_marginTop="10dp"
        android:backgroundTint="@color/colorPrimaryDark"
        android:hint="Email"
        android:inputType="textEmailAddress" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/colorPrimaryDark"
        android:hint="Phone"
        android:inputType="phone" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="150dp"
        android:layout_marginBottom="10dp"
```

Course Manual for Android App Development

```
        android:layout_marginTop="10dp"
        android:backgroundTint="@color/colorPrimaryDark"
        android:gravity="top"
        android:hint="Your Message"
        android:fitsSystemWindows="true"
        android:breakStrategy="balanced"
        android:inputType="textMultiLine"
        android:singleLine="false"
        android:padding="5dp" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:background="@color/colorPrimaryDark"
        android:elevation="4dp"
        android:paddingLeft="70dp"
        android:paddingRight="70dp"
        android:text="Submit"
        android:textColor="#fff" />
</LinearLayout>
```

Colors.xml File

Here I have defined color value for status bar, ActionBar/AppBar/Toolbar and other color also.

res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <colorname="colorPrimary">#F44336</color>
```

Course Manual for Android App Development

```
<colorname="colorPrimaryDark">#D32F2F</color>

<colorname="colorAccent">#FF4081</color>

<colorname="light_black">#504f4f</color>

<colorname="black">#000000</color>

</resources>
```

[view rawcolors.xml](#) hosted with ❤ by [GitHub](#)

Java Activity File

Following is the default code of java activity file.

src/SimpleContactFormUIDesign.java

```
//Creating Simple User Forms User Interface (UI) Design in Android

packageviralandroid.com.androidxmluserinterfacetutorial;

importandroid.os.Bundle;

importandroid.support.v7.app.AppCompatActivity;

importandroid.widget.LinearLayout;

publicclassSimpleContactFormUIDesignextendsAppCompatActivity {

    @Override

    protectedvoidonCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.simple_android_contact_form_ui_design);
```

```
}  
  
}
```

7.2- Playing Music:

How To Add Audio To App In Android Studio

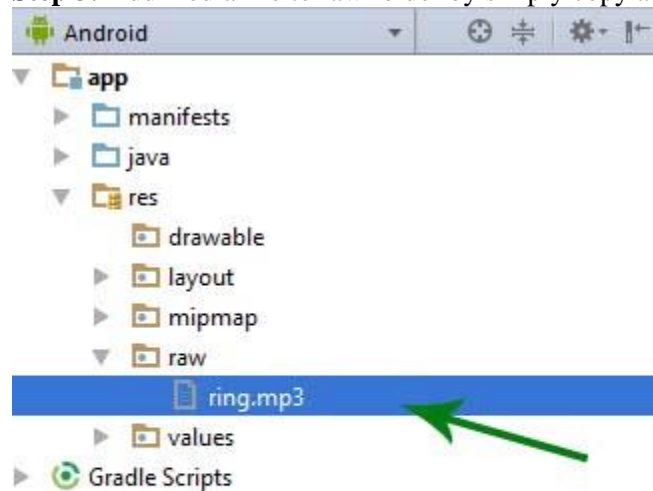
Adding audio clip to android application is a simple task as it also add some further functionality. Here is a step by step on how to play music when App will start.

Adding Audio to app in Android Studio:

Step 1: Open the android studio with the project in which you want to add-on audio clip/media file.

Step 2: Create a raw folder.

Step 3: Add media file to raw folder by simply copy and paste that to raw folder.



Step 4: Here we added a media file “ring.mp3” . Now open the Java File of desired activity, here we are adding audio in MainActivity.

Step 5: Further add this code

```
MediaPlayer ring= MediaPlayer.create(MainActivity.this,R.raw.ring);
```

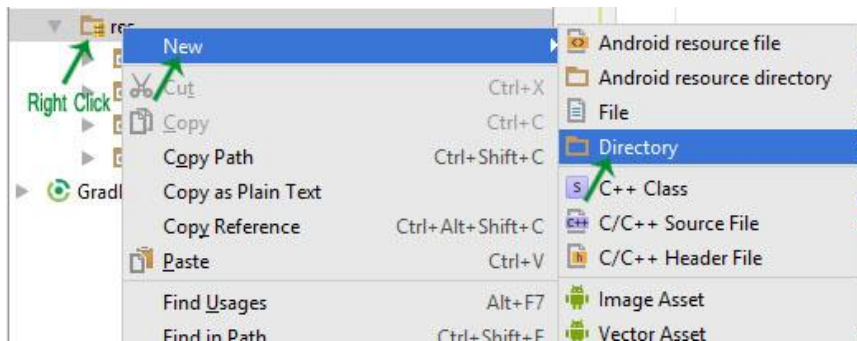
```
ring.start();
```

Step 6: Now run the App and your music will play when App will start.

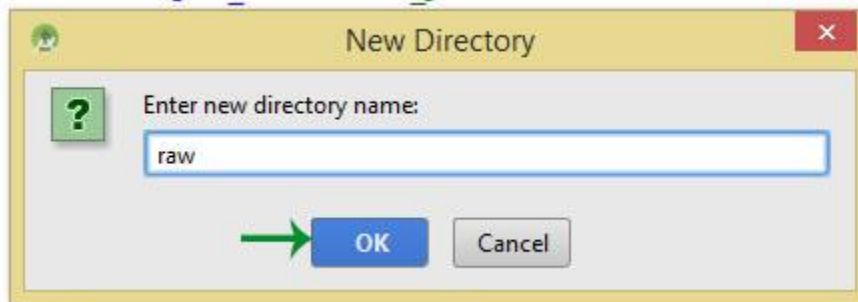
7.3- Creating a raw folder for music Files:

Step 1: There is no pre featured option in Android for adding raw folder unlike Assets folder. Open App folder and select res folder

Step 2: Right click on res folder, select **New> Directory**, then studio will open a dialog box and it will ask you to enter the name.



Step 3: Write “raw” and click OK. Open res folder and you will find your raw folder under it.



Now you have created raw folder in your project.

7.4- Using the MediaPlayer Class:

Example

Here is an example demonstrating the use of MediaPlayer class. It creates a basic media player that allows you to forward, backward, play and pause a song.

Course Manual for Android App Development

To experiment with this example, you need to run this on an actual device to hear the audio sound.

Steps	Description
1	You will use Android studio IDE to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add MediaPlayer code.
3	Modify the res/layout/activity_main to add respective XML components
4	Create a new folder under MediaPlayer with name as raw and place an mp3 music file in it with name as song.mp3
5	Run the application and choose a running android device and install the application on it and verify the results

Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;

import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
```

```
import android.widget.Toast;
import java.util.concurrent.TimeUnit;

public class MainActivity extends Activity {
    private Button b1, b2, b3, b4;
    private ImageView iv;
    private MediaPlayer mediaPlayer;

    private double startTime = 0;
    private double finalTime = 0;

    private Handler myHandler = new Handler();
    private int forwardTime = 5000;
    private int backwardTime = 5000;
    private SeekBar seekbar;
    private TextView tx1, tx2, tx3;

    public static int oneTimeOnly = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        b1 = (Button) findViewById(R.id.button);
        b2 = (Button) findViewById(R.id.button2);
        b3 = (Button) findViewById(R.id.button3);
```

```
b4 =(Button)findViewById(R.id.button4);

iv =(ImageView)findViewById(R.id.imageView);


tx1 =(TextView)findViewById(R.id.textView2);
tx2 =(TextView)findViewById(R.id.textView3);
tx3 =(TextView)findViewById(R.id.textView4);
tx3.setText("Song.mp3");


mediaPlayer =MediaPlayer.create(this, R.raw.song);
seekbar =(SeekBar)findViewById(R.id.seekBar);
seekbar.setClickable(false);
b2.setEnabled(false);


b3.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v){
Toast.makeText(getApplicationContext(),"Playing
        sound",Toast.LENGTH_SHORT).show();
        mediaPlayer.start();


        finalTime = mediaPlayer.getDuration();
        startTime = mediaPlayer.getCurrentPosition();

if(oneTimeOnly ==0){
        seekbar.setMax((int) finalTime);
        oneTimeOnly =1;
}
}
```

```
        tx2.setText(String.format("%d min, %d sec",
TimeUnit.MILLISECONDS.toMinutes((long) finalTime),
TimeUnit.MILLISECONDS.toSeconds((long) finalTime)-
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
        finalTime))))
);
```

```
        tx1.setText(String.format("%d min, %d sec",
TimeUnit.MILLISECONDS.toMinutes((long) startTime),
TimeUnit.MILLISECONDS.toSeconds((long) startTime)-
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
        startTime))))
);
```

```
        seekbar.setProgress((int)startTime);
        myHandler.postDelayed(UpdateSongTime,100);
        b2.setEnabled(true);
        b3.setEnabled(false);
    }
});
```

```
        b2.setOnClickListener(new View.OnClickListener(){
@Override
publicvoid onClick(View v){
Toast.makeText(getApplicationContext(),"Pausing
        sound",Toast.LENGTH_SHORT).show();
```

```
        mediaPlayer.pause();

        b2.setEnabled(false);

        b3.setEnabled(true);
    }
});

    b1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int temp = (int) startTime;

            if ((temp + forwardTime) <= finalTime) {
                startTime = startTime + forwardTime;
                mediaPlayer.seekTo((int) startTime);
                Toast.makeText(getApplicationContext(), "You have Jumped forward 5
                    seconds", Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText(getApplicationContext(), "Cannot jump forward 5
                    seconds", Toast.LENGTH_SHORT).show();
            }
        }
    });

    b4.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int temp = (int) startTime;
```

```
if((temp-backwardTime)>0){
    startTime = startTime - backwardTime;
    mediaPlayer.seekTo((int) startTime);
    Toast.makeText(getApplicationContext(),"You have Jumped backward 5
        seconds",Toast.LENGTH_SHORT).show();
}else{
    Toast.makeText(getApplicationContext(),"Cannot jump backward 5
        seconds",Toast.LENGTH_SHORT).show();
}
}
});
}

privateRunnableUpdateSongTime=newRunnable(){
    publicvoid run(){
        startTime = mediaPlayer.getCurrentPosition();
        tx1.setText(String.format("%d min, %d sec",
            TimeUnit.MILLISECONDS.toMinutes((long) startTime),
            TimeUnit.MILLISECONDS.toSeconds((long) startTime)-
            TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.
                toMinutes((long) startTime)))
    );
        seekbar.setProgress((int)startTime);
        myHandler.postDelayed(this,100);
    }
};
```

```
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

In the below code **abc** indicates the logo of ictdivition.com

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

<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"tools:context=".MainActivity">

    <TextViewandroid:text="Music Palyer"android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/textview"
android:textSize="35dp"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"/>

    <TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Tutorials point"
android:id="@+id/textView"
android:layout_below="@+id/textview"
android:layout_centerHorizontal="true"
android:textColor="#ff7aff24"
```

```
android:textSize="35dp"/>
```

```
<ImageView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:id="@+id/imageView"
```

```
    android:layout_below="@+id/textView"
```

```
    android:layout_centerHorizontal="true"
```

```
    android:src="@drawable/abc"/>
```

```
<Button
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="@string/forward"
```

```
    android:id="@+id/button"
```

```
    android:layout_alignParentBottom="true"
```

```
    android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentStart="true"/>
```

```
<Button
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="@string/pause"
```

```
    android:id="@+id/button2"
```

```
    android:layout_alignParentBottom="true"
```

```
    android:layout_alignLeft="@+id/imageView"
```

```
    android:layout_alignStart="@+id/imageView"/>
```



```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/back"
android:id="@+id/button3"
android:layout_alignTop="@+id/button2"
android:layout_toRightOf="@+id/button2"
android:layout_toEndOf="@+id/button2"/>
```

```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/rewind"
android:id="@+id/button4"
android:layout_alignTop="@+id/button3"
android:layout_toRightOf="@+id/button3"
android:layout_toEndOf="@+id/button3"/>
```

```
<SeekBar
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/seekBar"
android:layout_alignLeft="@+id/textview"
android:layout_alignStart="@+id/textview"
android:layout_alignRight="@+id/textview"
android:layout_alignEnd="@+id/textview"
```

```
android:layout_above="@+id/button"/>
```

```
<TextView
```

```
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
android:textAppearance="?android:attr/textAppearanceSmall"
```

```
android:text="Small Text"
```

```
android:id="@+id/textView2"
```

```
android:layout_above="@+id/seekBar"
```

```
android:layout_toLeftOf="@+id/textView"
```

```
android:layout_toStartOf="@+id/textView"/>
```

```
<TextView
```

```
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
android:textAppearance="?android:attr/textAppearanceSmall"
```

```
android:text="Small Text"
```

```
android:id="@+id/textView3"
```

```
android:layout_above="@+id/seekBar"
```

```
android:layout_alignRight="@+id/button4"
```

```
android:layout_alignEnd="@+id/button4"/>
```

```
<TextView
```

```
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
android:textAppearance="?android:attr/textAppearanceMedium"
```

```
android:text="Medium Text"
```

Course Manual for Android App Development

```
android:id="@+id/textView4"

android:layout_alignBaseline="@+id/textView2"

android:layout_alignBottom="@+id/textView2"

android:layout_centerHorizontal="true"/>

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

```
<resources>

<stringname="app_name">My Application</string>

<stringname="back"><![CDATA[<]]></string>

<stringname="rewind"><![CDATA[<<]]></string>

<stringname="forward"><![CDATA[>>]]></string>

<stringname="pause">||</string>

</resources>
```

Following is the content of **AndroidManifest.xml** file.

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

<manifestxmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.sairamkrishna.myapplication">

<application

android:allowBackup="true"

android:icon="@drawable/ic_launcher"

android:label="@string/app_name"

android:theme="@style/AppTheme">

<activity
```

Course Manual for Android App Development

```
android:name="com.example.sairamkrishna.myapplication.MainActivity"
android:label="@string/app_name">

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

</activity>

</application>
</manifest>
```

7.5- Simple app using the module taught. E.g.: Simple MediaPlayer: Class work (Practical)

PRACTICE:

Splash Screen ->Playing Music ->music Files ->MediaPlayer Class

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<https://developer.android.com/guide/>

[https://www.tutorialspoint.com/android/android_mediaplayer.htm /](https://www.tutorialspoint.com/android/android_mediaplayer.htm/)

Course Manual for Android App Development

MODULE 8: Android Security

CHAPTER OVERVIEW:

Android incorporates industry-leading security features and works with developers and device implementers to keep the Android platform and ecosystem safe. A robust security model is essential to enable a vigorous ecosystem of applications and devices built on and around the Android platform and supported by cloud services. As a result, through its entire development lifecycle, Android has been subject to a rigorous security program.

8.1- Android Security Concept 8.2- Using & Defining Permissions (practical)

OBJECTIVES:

- Permission concept in Android.
- Workflow before API 23

LEARNING OUTCOME:

- Permission concept in Android
- Workflow before API 23
- Runtime permissions

NEW TOPICS INTRODUCED IN THE CHAPTER:

Workflow before API 23

CORE DISCUSSION:

- Historical approach

Android is designed to be open. Android applications use advanced hardware and software, as well as local and served data, exposed through the platform to bring innovation and value to consumers. To realize that value, the platform offers an application environment that protects the confidentiality, integrity, and availability of users, data, applications, the device, and the network.

8.1- Android Security Concept:

The Android system installs every Android application with a unique user and group ID. Each application file is private to this generated user, e.g., other applications cannot access these files. In addition, each Android application is started in its own process.

Therefore, by means of the underlying Linux kernel, every Android application is isolated from other running applications.

If data should be shared, the application must do this explicitly via an Android component which handles the sharing of the data, e.g., via a service or a content provider.

Permission concept in Android

Android contains a permission system and predefined permissions for certain tasks. Every application can request required permissions. For example, an application may declare that it requires network access. It can also define new permissions.

System permissions have different levels, e.g., protection levels.

The permission concept has changed since API 23. Before API level 23 the user was asked during installation, after API level the user is asked during runtime.

An Android application declares the required permissions in its Android manifest. It can also define additional permissions which it can use to restrict access to certain components.

Workflow before API 23

Before API 23 the requested permissions are presented to the user before installing the application. The user needs to decide if these permissions shall be given to the application.

If the user denies a required permission, the related application cannot be installed. The check of the permission is only performed during installation. Permissions cannot be denied or granted after the installation.

Runtime permissions

Android 6.0 Marshmallow (API 23) introduced a new runtime permission model. If your application targets Android 6.0, you must use the new permissions model.

The two most important protection levels are normal and dangerous permissions:

- Normal permissions are permissions which are deemed harmless for the users privacy or the operation of other applications. For example, the permission to set the time zone. Normal permission are automatically granted to the application. See *Normal permissions* for a complete list.
- Dangerous permissions affect the users private information, or could potentially affect his data or the operation of other application. For example, the ability to read the users contact data. Dangerous permissions must be granted by the user at runtime to the app.

Course Manual for Android App Development

You can check, if you have the permission, via the `checkSelfPermission` method.

8.2- Using & Defining Permissions (practical)

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<https://source.android.com/security/>

MODULE 9: Android Networking and Background tasks

CHAPTER OVERVIEW:

One common task for most Android apps is connecting to the Internet. Most network-connected Android apps use HTTP to send and receive data. This article shows you how to write a simple application that connects to the Internet, send HTTP GET request & display the response

9.1- HttpURLConnection

9.2- JSON Parsing

9.3-Asynctask Vs Loaders

9.4- Looper, Handler and Handler Thread

9.5- Network Communication using any library (Retrofit)

9.6- Sending Email, SMS Phone Call

9.7- Simple app using the module taught. E.g.: Weather Forecasting (Practical)

OBJECTIVES:

- About HTTP/HTTPS POST & GET Requests
- To create a recursive descent parser for your own JSON streams
- Network Communication using any library (Retrofit)

LEARNING OUTCOME:

- Secure Communication with HTTPS
- Response Handling
- Posting Content
- Performance
- Handling Network Sign-On
- HTTP Authentication
- Sessions with Cookies
- HTTP Methods
- Proxies
- IPv6 Support
- Response Caching
- Avoiding Bugs In Earlier Releases
- Number Handling
- AsyncTask
- AsyncTask usage

Course Manual for Android App Development

- AsyncTask parameters
- Executing an AsyncTask
- Cancelling an AsyncTask
- Limitations of AsyncTask
- Handler
- HandlerThread
- Create Java Classes for Resources
- Creating the Retrofit instance
- Define the Endpoints
- RxJava
- Using Authentication Headers
- Email
- SMS
- Phone Call

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Network Communication Retrofit library

CORE DISCUSSION:

- **Fundamental technology**

Retrofit Android

Retrofit is type-safe REST client for Android and Java which aims to make it easier to consume RESTful web services. We'll not go into the details of Retrofit 1.x versions and jump onto **Retrofit 2** directly which has a lot of new features and a changed internal API compared to the previous versions.

Retrofit 2 by default leverages OkHttp as the networking layer and is built on top of it.

Course Manual for Android App Development

- **Alternative technology**

Volley

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster. Volley is available on GitHub.

Volley offers the following benefits:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Transparent disk and memory response caching with standard HTTP cache coherence.
- Support for request prioritization.
- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
- Ease of customization, for example, for retry and backoff.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Debugging and tracing tools.

Volley excels at RPC-type operations used to populate a UI, such as fetching a page of search results as structured data. It integrates easily with any protocol and comes out of the box with support for raw strings, images, and JSON. By providing built-in support for the features you need, Volley frees you from writing boilerplate code and allows you to concentrate on the logic that is specific to your app.

9.1- HttpURLConnection:

we are going to learn about HTTP/HTTPS POST & GET Requests. Http is an underlying protocol used by world wide web. POST and GET are two most commonly used HTTP methods for request and response between the client and the server. GET method basically requests data from specified resource, whereas Post method submits data to be processed to a specified resource.

A `URLConnection` with support for HTTP-specific features. See the spec for details.

Uses of this class follow a pattern:

1. Obtain a new `HttpURLConnection` by calling `URL.openConnection()` and casting the result to `HttpURLConnection`.
2. Prepare the request. The primary property of a request is its URI. Request headers may also include metadata such as credentials, preferred content types, and session cookies.
3. Optionally upload a request body. Instances must be configured with `setDoOutput(true)` if they include a request body. Transmit data by writing to the stream returned by `getOutputStream()`.
4. Read the response. Response headers typically include metadata such as the response body's content type and length, modified dates and session cookies. The response body may be read from the stream returned by `getInputStream()`. If the response has no body, that method returns an empty stream.

Course Manual for Android App Development

5. **Disconnect.** Once the response body has been read, the `URLConnection` should be closed by calling `disconnect()`. Disconnecting releases the resources held by a connection so they may be closed or reused.

For example, to retrieve the webpage at `http://www.android.com/`:

```
URL url = new URL("http://www.android.com/");

URLConnection urlConnection = (URLConnection) url.openConnection();

try {

    InputStream in = new BufferedInputStream(urlConnection.getInputStream());

    readStream(in);

} finally {

    urlConnection.disconnect();

}
```

Secure Communication with HTTPS

Calling `openConnection()` on a URL with the "https" scheme will return an `HttpsURLConnection`, which allows for overriding the default `HostnameVerifier` and `SSLSocketFactory`. An application-supplied `SSLSocketFactory` created from an `SSLContext` can provide a custom `X509TrustManager` for verifying certificate chains and a custom `X509KeyManager` for supplying client certificates. See `HttpsURLConnection` for more details.

Response Handling

`URLConnection` will follow up to five HTTP redirects. It will follow redirects from one origin server to another. This implementation doesn't follow redirects from HTTPS to HTTP or vice versa.

If the HTTP response indicates that an error occurred, `getInputStream()` will throw an `IOException`. Use `getErrorStream()` to read the error response. The headers can be read in the normal way using `getHeaderFields()`,

Posting Content

To upload data to a web server, configure the connection for output using `setDoOutput(true)`.

Course Manual for Android App Development

For best performance, you should call either `setFixedLengthStreamingMode(int)` when the body length is known in advance, or `setChunkedStreamingMode(int)` when it is not. Otherwise `URLConnection` will be forced to buffer the complete request body in memory before it is transmitted, wasting (and possibly exhausting) heap and increasing latency.

For example, to perform an upload:

```
URLConnection urlConnection = (URLConnection) url.openConnection();

try {
    urlConnection.setDoOutput(true);
    urlConnection.setChunkedStreamingMode(0);

    OutputStream out = new BufferedOutputStream(urlConnection.getOutputStream());
    writeStream(out);

    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
    readStream(in);
} finally {
    urlConnection.disconnect();
}
```

Performance

The input and output streams returned by this class are **not buffered**. Most callers should wrap the returned streams with `BufferedInputStream` or `BufferedOutputStream`. Callers that do only bulk reads or writes may omit buffering.

When transferring large amounts of data to or from a server, use streams to limit how much data is in memory at once. Unless you need the entire body to be in memory at once, process it as a stream (rather than storing the complete body as a single byte array or string).

To reduce latency, this class may reuse the same underlying `Socket` for multiple request/response pairs. As a result, HTTP connections may be held open longer than necessary. Calls to `disconnect()` may return the socket to a pool of connected sockets.

Course Manual for Android App Development

By default, this implementation of `HttpURLConnection` requests that servers use gzip compression and it automatically decompresses the data for callers of `getInputStream()`. The `Content-Encoding` and `Content-Length` response headers are cleared in this case. Gzip compression can be disabled by setting the acceptable encodings in the request header:

```
urlConnection.setRequestProperty("Accept-Encoding", "identity");
```

Setting the `Accept-Encoding` request header explicitly disables automatic decompression and leaves the response headers intact; callers must handle decompression as needed, according to the `Content-Encoding` header of the response.

`getContentLength()` returns the number of bytes transmitted and cannot be used to predict how many bytes can be read from `getInputStream()` for compressed streams. Instead, read that stream until it is exhausted, i.e. when `read()` returns `-1`.

Handling Network Sign-On

Some Wi-Fi networks block Internet access until the user clicks through a sign-on page. Such sign-on pages are typically presented by using HTTP redirects. You can use `getURL()` to test if your connection has been unexpectedly redirected. This check is not valid until **after** the response headers have been received, which you can trigger by calling `getHeaderFields()` or `getInputStream()`. For example, to check that a response was not redirected to an unexpected host:

```
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

try {

    InputStream in = new BufferedInputStream(urlConnection.getInputStream());

    if (!url.getHost().equals(urlConnection.getURL().getHost())) {

        // we were redirected! Kick the user out to the browser to sign on?

    }

    ...

} finally {

    urlConnection.disconnect();

}
```

Course Manual for Android App Development

HTTP Authentication

URLConnection supports HTTP basic authentication. Use Authenticator to set the VM-wide authentication handler:

```
Authenticator.setDefault(new Authenticator() {  
  
    protected PasswordAuthentication getPasswordAuthentication() {  
  
        return new PasswordAuthentication(username, password.toCharArray());  
  
    }  
  
});
```

Unless paired with HTTPS, this is **not** a secure mechanism for user authentication. In particular, the username, password, request and response are all transmitted over the network without encryption.

Sessions with Cookies

To establish and maintain a potentially long-lived session between client and server, HttpURLConnection includes an extensible cookie manager. Enable VM-wide cookie management using CookieHandler and CookieManager:

```
CookieManager cookieManager = new CookieManager();  
  
CookieHandler.setDefault(cookieManager);
```

By default, CookieManager accepts cookies from the origin server only. Two other policies are included: ACCEPT_ALL and ACCEPT_NONE. Implement CookiePolicy to define a custom policy.

The default CookieManager keeps all accepted cookies in memory. It will forget these cookies when the VM exits. Implement CookieStore to define a custom cookie store.

In addition to the cookies set by HTTP responses, you may set cookies programmatically. To be included in HTTP request headers, cookies must have the domain and path properties set.

By default, new instances of HttpCookie work only with servers that support RFC 2965 cookies. Many web servers support only the older specification, RFC 2109. For compatibility with the most web servers, set the cookie version to 0.

Course Manual for Android App Development

For example, to receive `www.twitter.com` in French:

```
HttpCookie cookie = new HttpCookie("lang", "fr");

cookie.setDomain("twitter.com");

cookie.setPath("/");

cookie.setVersion(0);

cookieManager.getCookieStore().add(new URI("http://twitter.com/"), cookie);
```

HTTP Methods

`URLConnection` uses the GET method by default. It will use POST if `setDoOutput(true)` has been called. Other HTTP methods (OPTIONS, HEAD, PUT, DELETE and TRACE) can be used with `setRequestMethod(String)`.

Proxies

By default, this class will connect directly to the origin server. It can also connect via an HTTP or SOCKS proxy. To use a proxy, use `URLConnection.Proxy` when creating the connection.

IPv6 Support

This class includes transparent support for IPv6. For hosts with both IPv4 and IPv6 addresses, it will attempt to connect to each of a host's addresses until a connection is established.

Response Caching

Android 4.0 (Ice Cream Sandwich, API level 15) includes a response cache.

See `android.net.http.HttpResponseCache` for instructions on enabling HTTP caching in your application.

Avoiding Bugs In Earlier Releases

Prior to Android 2.2 (Froyo), this class had some frustrating bugs. In particular, calling `close()` on a readable `InputStream` could poison the connection pool. Work around this by disabling connection pooling:

```
private void disableConnectionReuseIfNecessary() {

    // Work around pre-Froyo bugs in HTTP connection reuse.

    if (Integer.parseInt(Build.VERSION.SDK) < Build.VERSION_CODES.FROYO) {
```

```
System.setProperty("http.keepAlive", "false");  
  
}  
  
}
```

Each instance of `URLConnection` may be used for one request/response pair. Instances of this class are not thread safe.

9.2- JSON Parsing:

To create a recursive descent parser for your own JSON streams, first create an entry point method that creates a `JsonReader`.

Next, create handler methods for each structure in your JSON text. You'll need a method for each object type and for each array type.

- Within **array handling** methods, first call `beginArray()` to consume the array's opening bracket. Then create a while loop that accumulates values, terminating when `hasNext()` is false. Finally, read the array's closing bracket by calling `endArray()`.
- Within **object handling** methods, first call `beginObject()` to consume the object's opening brace. Then create a while loop that assigns values to local variables based on their name. This loop should terminate when `hasNext()` is false. Finally, read the object's closing brace by calling `endObject()`.

When a nested object or array is encountered, delegate to the corresponding handler method.

When an unknown name is encountered, strict parsers should fail with an exception. Lenient parsers should call `skipValue()` to recursively skip the value's nested tokens, which may otherwise conflict.

If a value may be null, you should first check using `peek()`. Null literals can be consumed using either `nextNull()` or `skipValue()`.

Example

Suppose we'd like to parse a stream of messages such as the following:

```
[  
  
  {  
  
    "id": 912345678901,  
  
    "text": "How do I read JSON on Android?",  
  
    "geo": null,  
  
    "user": {  
  
      "name": "android_newb",
```


Course Manual for Android App Development

```
[
  {
    "followers_count": 41
  },
  {
    "id": 912345678902,
    "text": "@android_newb just use android.util.JsonReader!",
    "geo": [50.454722, -104.606667],
    "user": {
      "name": "jesse",
      "followers_count": 2
    }
  }
]
```

This code implements the parser for the above structure:

```
public List<Message> readJsonStream(InputStream in) throws IOException {  
    JsonReader reader = new JsonReader(new InputStreamReader(in, "UTF-8"));  
    try {  
        return readMessagesArray(reader);  
    } finally {  
        reader.close();  
    }  
}
```

```
public List<Message> readMessagesArray(JsonReader reader) throws IOException {
```

```
List<Message> messages = new ArrayList<Message>();

reader.beginArray();

while (reader.hasNext()) {
    messages.add(readMessage(reader));
}

reader.endArray();

return messages;
}

public Message readMessage(JsonReader reader) throws IOException {
    long id = -1;

    String text = null;

    User user = null;

    List<Double> geo = null;

    reader.beginObject();

    while (reader.hasNext()) {
        String name = reader洗洗洗Name();

        if (name.equals("id")) {
            id = reader.nextLong();
        } else if (name.equals("text")) {
            text = reader.nextString();
        } else if (name.equals("geo") && reader.peek() != JsonToken.NULL) {
            geo = readDoublesArray(reader);
        }
    }
}
```

```
    } else if (name.equals("user")) {  
        user = readUser(reader);  
    } else {  
        reader.skipValue();  
    }  
}  
  
reader.endObject();  
  
return new Message(id, text, user, geo);  
}  
  
public List<Double> readDoublesArray(JsonReader reader) throws IOException {  
    List<Double> doubles = new ArrayList<Double>();  
  
    reader.beginArray();  
    while (reader.hasNext()) {  
        doubles.add(reader.nextDouble());  
    }  
    reader.endArray();  
    return doubles;  
}  
  
public User readUser(JsonReader reader) throws IOException {  
    String username = null;  
    int followersCount = -1;
```

```
reader.beginObject();

while (reader.hasNext()) {

    String name = reader.nextName();

    if (name.equals("name")) {

        username = reader.nextString();

    } else if (name.equals("followers_count")) {

        followersCount = reader.nextInt();

    } else {

        reader.skipValue();

    }

}

reader.endObject();

return new User(username, followersCount);

}
```

Number Handling

This reader permits numeric values to be read as strings and string values to be read as numbers. For example, both elements of the JSON array [1, "1"] may be read using either `nextInt()` or `nextString()`. This behavior is intended to prevent lossy numeric conversions: double is JavaScript's only numeric type and very large values like 9007199254740993 cannot be represented exactly on that platform. To minimize precision loss, extremely large values should be written and read as strings in JSON.

Each `JsonReader` may be used to read a single JSON stream. Instances of this class are not thread safe.

9.3- AsyncTask Vs Loaders:

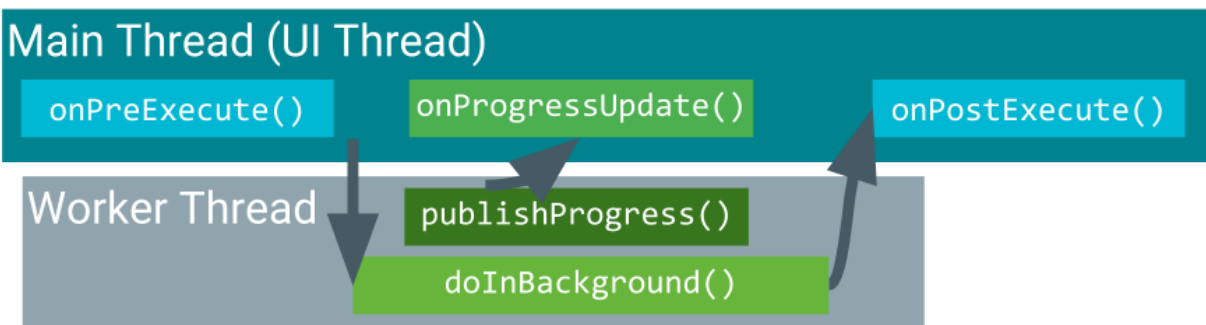
AsyncTask

Use the `AsyncTask` class to implement an asynchronous, long-running task on a worker thread. (A *worker thread* is any thread which is not the main or UI thread.) `AsyncTask` allows you to perform background operations and publish results on the UI thread without manipulating threads or handlers. When `AsyncTask` is executed, it goes through four steps:

Course Manual for Android App Development

1. `onPreExecute()` is invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the UI.
2. `doInBackground(Params...)` is invoked on the background thread immediately after `onPreExecute()` finishes. This step performs a background computation, returns a result, and passes the result to `onPostExecute()`. The `doInBackground()` method can also call `publishProgress(Progress...)` to publish one or more units of progress.
3. `onProgressUpdate(Progress...)` runs on the UI thread after `publishProgress(Progress...)` is invoked. Use `onProgressUpdate()` to report any form of progress to the UI thread while the background computation is executing. For instance, you can use it to pass the data to animate a progress bar or show logs in a text field.
4. `onPostExecute(Result)` runs on the UI thread after the background computation has finished.

For complete details on these methods, see the `AsyncTask` reference. Below is a diagram of their calling order.



AsyncTask usage

To use the `AsyncTask` class, define a subclass of `AsyncTask` that overrides the `doInBackground(Params...)` method (and usually the `onPostExecute(Result)` method as well). This section describes the parameters and usage of `AsyncTask`, then shows a complete example.

AsyncTask parameters

In your subclass of `AsyncTask`, provide the data types for three kinds of parameters:

- "Params" specifies the type of parameters passed to `doInBackground()` as an array.
- "Progress" specifies the type of parameters passed to `publishProgress()` on the background thread. These parameters are then passed to the `onProgressUpdate()` method on the main thread.
- "Result" specifies the type of parameter that `doInBackground()` returns. This parameter is automatically passed to `onPostExecute()` on the main thread.

Specify a data type for each of these parameter types, or use `Void` if the parameter type will not be used. For example:

```
public class MyAsyncTask extends AsyncTask <String, Void, Bitmap>{ }
```

In this class declaration:

Course Manual for Android App Development

- The "Params" parameter type is String, which means that MyAsyncTask takes one or more strings as parameters in doInBackground(), for example to use in a query.
- The "Progress" parameter type is Void, which means that MyAsyncTask won't use the publishProgress() or onProgressUpdate() methods.
- The "Result" parameter type is Bitmap. MyAsyncTask returns a Bitmap in doInBackground(), which is passed into onPostExecute().

Example of an AsyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

The example above goes through three of the four basic AsyncTask steps:

- doInBackground() downloads content, a long-running task. It computes the percentage of files downloaded from the index of the for loop and passes it to publishProgress(). The check for isCancelled() inside the for loop ensures that if the task has been cancelled, the system does not wait for the loop to complete.
- onProgressUpdate() updates the percent progress. It is called every time the publishProgress() method is called inside doInBackground(), which updates the percent progress.
- doInBackground() computes the total number of bytes downloaded and returns it. onPostExecute() receives the returned result and passes it into onPostExecute(), where it is displayed in a dialog.

The parameter types used in this example are:

- URL for the "Params" parameter type. The URL type means you can pass any number of URLs into the call, and the URLs are automatically passed into the doInBackground() method as an array.
- Integer for the "Progress" parameter type.
- Long for the "Result" parameter type.

Executing an AsyncTask

Course Manual for Android App Development

After you define a subclass of `AsyncTask`, instantiate it on the UI thread. Then call `execute()` on the instance, passing in any number of parameters. (These parameters correspond to the "Params" parameter type discussed above).

For example, to execute the `DownloadFilesTask` task defined above, use the following line of code:

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Cancelling an AsyncTask

You can cancel a task at any time, from any thread, by invoking the `cancel()` method.

- The `cancel()` method returns `false` if the task could not be cancelled, typically because it has already completed normally. Otherwise, `cancel()` returns `true`.
- To find out whether a task has been cancelled, check the return value of `isCancelled()` periodically from `doInBackground(Object[])`, for example from inside a loop as shown in the example above. The `isCancelled()` method returns `true` if the task was cancelled before it completed normally.
- After an `AsyncTask` task is cancelled, `onPostExecute()` will not be invoked after `doInBackground()` returns. Instead, `onCancelled(Object)` is invoked. The default implementation of `onCancelled(Object)` simply invokes `onCancelled()` and ignores the result.
- By default, in-process tasks are allowed to complete. To allow `cancel()` to interrupt the thread that's executing the task, pass `true` for the value of `mayInterruptIfRunning`.

Limitations of AsyncTask

`AsyncTask` is impractical for some use cases:

- Changes to device configuration cause problems.

When device configuration changes while an `AsyncTask` is running, for example if the user changes the screen orientation, the activity that created the `AsyncTask` is destroyed and re-created.

The `AsyncTask` is unable to access the newly created activity, and the results of the `AsyncTask` aren't published.

- Old `AsyncTask` objects stay around, and your app may run out of memory or crash.
If the activity that created the `AsyncTask` is destroyed, the `AsyncTask` is not destroyed along with it. For example, if your user exits the application after the `AsyncTask` has started, the `AsyncTask` keeps using resources unless you call `cancel()`.

When to use `AsyncTask`:

- Short or interruptible tasks.
- Tasks that don't need to report back to UI or user.
- Low-priority tasks that can be left unfinished.

For all other situations, use `AsyncTaskLoader`, which is part of the `Loader` framework.

9.4- Looper, Handler and Handler Thread:

Looper

```
public final class Looper
extends Object
java.lang.Object

↳ android.os.Looper
```

Class used to run a message loop for a thread. Threads by default do not have a message loop associated with them; to create one, call `prepare()` in the thread that is to run the loop, and then `loop()` to have it process messages until the loop is stopped.

Most interaction with a message loop is through the `Handler` class.

This is a typical example of the implementation of a `Looper` thread, using the separation of `prepare()` and `loop()` to create an initial `Handler` to communicate with the `Looper`.

```
class LooperThread extends Thread {

    public Handler mHandler;

    public void run() {

        Looper.prepare();

        mHandler = new Handler() {

            public void handleMessage(Message msg) {

                // process incoming messages here

            }

        };

    }

};
```



```
        Looper.loop();  
  
    }  
  
}
```

Handler

```
public class Handler  
    extends Object  
    java.lang.Object
```

```
↳ android.os.Handler
```

Known direct subclasses

AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

There are two main uses for a Handler: (1) to schedule messages and runnables to be executed at some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

Scheduling messages is accomplished with the `post(Runnable)`, `postAtTime(Runnable, long)`, `postDelayed(Runnable, long)`, `sendMessage(Message)`, `sendMessageAtTime(Message, long)`, and `sendMessageDelayed(Message, long)` methods. The *post* versions allow you to enqueue Runnable objects to be called by the message queue when they are received; the *sendMessage* versions allow you to enqueue a Message object containing a bundle of data that will be processed by the Handler's `handleMessage(Message)` method (requiring that you implement a subclass of Handler).

When posting or sending to a Handler, you can either allow the item to be processed as soon as the message queue is ready to do so, or specify a delay before it gets processed or absolute time for it to be processed. The latter two allow you to implement timeouts, ticks, and other timing-based behavior.

When a process is created for your application, its main thread is dedicated to running a message queue that takes care of managing the top-level application objects (activities, broadcast receivers, etc) and any windows they create. You can create your own threads, and communicate back with the main application thread through a Handler. This is done by calling the same *post* or *sendMessage* methods as before, but from your new thread. The given Runnable or Message will then be scheduled in the Handler's message queue and processed when appropriate.

HandlerThread

```
public class HandlerThread
extends Thread
java.lang.Object
```

```
↳ java.lang.Thread
```

```
↳ android.os.HandlerThread
```

There are 2 main ways to use HandlerThreads.

1. Create a new HandlerThread, create a new Handler using this HandlerThread and post your tasks on this handler.

Extend HandlerThread inside your CustomHandlerThread, create a Handler to process your task. You would take this approach if you know the task you want to perform and just need to pass in parameters. An example would be creating a HandlerThread to download images or perform networking operations.

9.5- Network Communication using any library (Retrofit):

Overview

Retrofit is a type-safe REST client for Android (or just Java) developed by Square. The library provides a powerful framework for authenticating and interacting with APIs and sending network requests with OkHttp.

This library makes downloading JSON or XML data from a web API fairly straightforward. Once the data is downloaded then it is parsed into a Plain Old Java Object (POJO) which must be defined for each "resource" in the response.

Setup

Make sure to require Internet permissions in your AndroidManifest.xml file:

```
<manifestxmlns:android="http://schemas.android.com/apk/res/android">
<uses-permissionandroid:name="android.permission.INTERNET" />
</manifest>
```

Add the following to your app/build.gradle file:

```
dependencies {
    compile 'com.google.code.gson:gson:2.8.2'
    compile 'com.squareup.retrofit2:retrofit:2.3.0'
    compile 'com.squareup.retrofit2:converter-gson:2.3.0'
```

```
}
```

Note: if you are upgrading from Retrofit 2 beta 1 or beta2, your package imports will need to be changed from `import retrofit.XXXX` to `import retrofit2.XXXX`. You will also need to update your `OkHttp` imports from `import okhttp.XXXX` to `import okhttp3.XXXX` because of the new `OkHttp3` dependency:

```
find . -name '*.java' -exec gsed -i 's/import retrofit\./import retrofit2./g'\{\} +
```

```
find . -name '*.java' -exec gsed -i 's/import com.squareup.okhttp/import okhttp3/g'\{\} +
```

If you intend to use `RxJava` with `Retrofit 2`, you will also need to include the `RxJava` adapter:

```
dependencies {  
    compile 'io.reactivex:rxjava:1.1.6'  
    compile 'io.reactivex:rxandroid:1.2.1'  
    compile 'com.squareup.retrofit2:adapter-rxjava:2.1.0'  
}
```

In the past, `Retrofit` relied on the `Gson` library to serialize and deserialize `JSON` data. `Retrofit 2` now supports many different parsers for processing network response data, including `Moshi`, a library build by `Square` for efficient `JSON` parsing. However, there are a few limitations, so if you are not sure which one to choose, use the `Gson` converter.

Create Java Classes for Resources

There are two approaches discussed in this guide. The first way is the manual approach, which requires you to learn how to use the `Gson` library. The second approach is you can also auto-generate the Java classes you need by capturing the `JSON` output and using `jsonschema2pojo`. We encourage you to follow the manual way to best understand how the auto-generated code approach works.

Automated approach - Auto-generating the Java classes.

Assuming you have the `JSON` response already, go to jsonschema2pojo.org. Make sure to select `JSON` as the Source Type:

Source type:

☐ JSON Schema ☒ JSON

Set the Annotation style to `Gson`.

Annotation style:

☐ Jackson 2.x ☐ Jackson 1.x
☒ Gson ☐ None

Next, paste the `JSON` output into the textbox:

jsonschema2pojo

 Star 2,536
  Share 184
  Tweet

Generate Plain Old Java Objects from JSON or JSON-Schema.

```

1 {
2   "login": "codepath",
3   "id": 3710273,
4   "avatar_url": "https://avatars2.githubusercontent.com/u/3710273",
5   "gravatar_id": "",
6   "url": "https://api.github.com/users/codepath",
7   "html_url": "https://github.com/codepath",
8   "followers_url": "https://api.github.com/users/codepath/followers",
9   "following_url": "https://api.github.com/users/codepath/following",
10  "gists_url": "https://api.github.com/users/codepath/gists",
11  "starred_url": "https://api.github.com/users/codepath/starred",
12  "subscriptions_url": "https://api.github.com/users/codepath/subscriptions",
13  "organizations_url": "https://api.github.com/users/codepath/orgs",
14  "repos_url": "https://api.github.com/users/codepath/repos",
15  "events_url": "https://api.github.com/users/codepath/events",
16  "received_events_url": "https://api.github.com/users/codepath/received_events",
17  "type": "Organization",
18  "site_admin": false,
19  "name": "CodePath",
20  "company": null,
21  "blog": "http://codepath.com",
22  "location": "San Francisco, CA",
23  "email": "contact@codepath.com",
24  "hireable": null,
25  "bio": "Mobile training, community and mentorship",
26  "public_repos": 101,
27  "public_gists": 0,
28  "followers": 0,
29  "following": 0,
30  "created_at": "2013-02-26T23:20:32Z",
31  "updated_at": "2017-02-13T10:41:34Z"
32 }

```

Package

Class name

Source type:

☐ JSON Schema
 ☒ JSON

Annotation style:


☐ Jackson 2.x
 ☐ Jackson 1.x
 ☒ Gson
 ☐ None

☐ Generate builder methods
 ☒ Use primitive types
 ☐ Use long integers
 ☐ Use double numbers
 ☐ Use Joda dates
 ☐ Use Commons-Lang3
 ☒ Include getters and setters
 ☐ Include constructors
 ☐ Include `hashCode` and `equals`
☐ Include `toString`
☐ Include JSR-303 annotations
 ☐ Allow additional properties
 ☐ Make classes serializable
 ☐ Make classes parcelable

Property word delimiters:

Click the Preview button. You should see the top section look sort of like the following:

Preview

Copy to Clipboard  

```
-----code.sample.model.Example.java-----
-

package code.sample.model;

import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

public class Example {

    @SerializedName("login")
    @Expose
    private String login;
    @SerializedName("id")
    @Expose
    private int id;
    @SerializedName("avatar_url")
    @Expose
    private String avatarUrl;
    @SerializedName("gravatar_id")
    @Expose
    private String gravatarId;
    @SerializedName("url")
    @Expose
    private String url;
    @SerializedName("html_url")
    @Expose
    private String htmlUrl;
    @SerializedName("followers_url")
    @Expose
    private String followersUrl;
    @SerializedName("following_url")
    @Expose
    private String followingUrl;
    @SerializedName("gists_url")
    @Expose
    private String gistsUrl;
}
```

Paste the generated class into your project under a models sub-package. Rename the class name Example to reflect your model name. For this example, we will call this file and class the UserModel.

Note: Android does not come normally with many of the javax.annotation library by default. If you wish to keep the @Generated annotation, you will need to add this dependency. See this [Stack Overflow](#) discussion for more context. Otherwise, you can delete that annotation and use the rest of the generated code.

```
dependencies {
    provided 'org.glassfish:javax.annotation:10.0-b28'
}
```

Creating the Retrofit instance

To send out network requests to an API, we need to use the [Retrofit builder] (<http://square.github.io/retrofit/2.x/retrofit/retrofit2/Retrofit.Builder.html>) class and specify the base URL for the service.

```
// Trailing slash is needed
publicstaticfinalStringBASE_URL="http://api.myservice.com/";
Retrofit retrofit =newRetrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
    .build();
```

Note also that we need to specify a factory for deserializing the response using the Gson library. The order in which the converters are added will be the sequence in which they are attempted to be processed

Course Manual for Android App Development

as discussed in this [video talk](#). If we wish to pass in a custom Gson parser instance, it can be specified too:

```
Gson gson =newGsonBuilder()
    .setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
    .create();

Retrofit retrofit =newRetrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create(gson))
    .build();
```

Define the Endpoints

With Retrofit 2, endpoints are defined inside of an interface using special retrofit annotations to encode details about the parameters and request method. In addition, the return value is always a parameterized `Call<T>` object such as `Call<User>`. If you do not need any type-specific response, you can specify return value as simply `Call<ResponseBody>`.

For instance, the interface defines each endpoint in the following way:

```
publicinterfaceMyApiEndpointInterface {
// Request method and URL specified in the annotation

@GET("users/{username}")
Call<User>getUser(@Path("username") Stringusername);

@GET("group/{id}/users")
Call<List<User>>groupList(@Path("id") intgroupId, @Query("sort") Stringsort);

@POST("users/new")
Call<User>createUser(@BodyUseruser);
}
```

Notice that each endpoint specifies an annotation of the HTTP method (GET, POST, etc.) and method that will be used to dispatch the network call. Note that the parameters of this method can also have special annotations:

Annotation	Description
@Path	variable substitution for the API endpoint (i.e. username will be swapped for {username} in the URL endpoint).
@Query	specifies the query key name with the value of the annotated parameter.
@Body	payload for the POST call (serialized from a Java object to a JSON string)

Course Manual for Android App Development

Annotation	Description
@Header	specifies the header with the value of the annotated parameter

Changing the base URL

Normally, the base URL is defined when you instantiated an Retrofit instance. Retrofit 2 allows you to override the base URL specified by changing it in the annotation (i.e. if you need to keep one specific endpoint using an older API endpoint)

```
@POST("https://api.github.com/api/v3")
```

There are also others that allow you to modify the base URL using relative paths (and not the fully qualified URL) as discussed in this blog article.

Adding headers

Notice that there is a @Headers and @Header annotation. The Headers can be used to provide pre-defined ones:

```
@Headers({ "Cache-Control: max-age=640000", "User-Agent: My-App-Name" })  
@GET("/some/endpoint")
```

We can also add headers as a parameter to the endpoint:

```
@Multipart  
@POST("/some/endpoint")  
Call<SomeResponse> someEndpoint(@Header("Cache-Control") int maxAge)
```

Form data

If we wish to submit form-encoded data, we can use the FormUrlEncoded annotation. The @Field annotation will denote what payload will be submitted as form data.

```
@FormUrlEncoded  
@POST("/some/endpoint")  
Observable<SomeResponse> someEndpoint(@Field("code") String code);
```

Multipart forms

If we need to upload images or files, we need to send by using Multipart forms. We will mark the endpoint with @Multipart, and label at least one parameter with @Part.

```
@Multipart  
@POST("/some/endpoint")  
Call<Response> uploadImage(@Part("description") String description, @Part("image") RequestBody image)
```

Assuming we have a reference to the file, we can create a RequestBody object:

```
MediaType MEDIA_TYPE_PNG = MediaType.parse("image/png");  
file = new File("/storage/emulated/0/Pictures/MyApp/test.png");
```

Course Manual for Android App Development

```
RequestBody requestBody =RequestBody.create(MEDIA_TYPE_PNG, file);
```

```
Call<Response> call = apiService.uploadImage("test", requestBody);
```

If you need to specify a unique filename for your multipart upload, there is currently an issue in Retrofit 2 tracked in this [ticket](#). Alternatively, you can create a multi-part RequestBody according to this [OkHttp recipe guide](#) and pass it along as one of the @Part annotated parameters:

```
RequestBody requestBody =newMultipartBody.Builder()
    .setType(MultipartBody.FORM)
    .addFormDataPart("title", "Square Logo")
    .addFormDataPart("image", "logo-square.png",
RequestBody.create(MEDIA_TYPE_PNG, newFile("website/static/logo-square.png")))
    .build();
```

Form URL encoding

If we wish to POST form-encoded name/value pairs, we can use the @FormUrlEncoded and @FieldMap annotations:

```
@FormUrlEncoded
@POST("some/endpoint")
Call<SomeResponse> someEndpoint(@FieldMapMap<String, String> names);
```

POSTing JSON data

Retrofit 2 will use the converter library chosen to handle the deserialization of data from a Java object. If you annotate the parameter with a @Body parameter, this work will be done automatically. If you are using the Gson library for instance, any field belonging to the class will be serialized for you. You can change this name using the @SerializedName decorator:

```
publicclassUser {

    @SerializedName("id")
    int mId;

    @SerializedName("name")
    String mName;

    publicUser(intid, Stringname ) {
        this.mId = id;
        this.mName = name;
    }
}
```

Our endpoint would look like the following:

```
@POST("/users/new")
Call<User> createUser(@BodyUser user);
```

We could invoke this API call as follows:

```
User user =newUser(123, "John Doe");
Call<User> call = apiService.createuser(user);
call.enqueue(newCallback<User>() {
```


Course Manual for Android App Development

```
@Override
public void onResponse(Call<User>call, Response<User>response) {

}

@Override
public void onFailure(Call<User>call, Throwable t) {

}
```

The resulting network call would POST this data:

```
{"name":"John Doe","id":123}
```

Upgrading from Retrofit 1

If you are trying to upgrade from Retrofit 1, you may remember that the last parameter had to be a Callback type if you wanted to define the API call to run asynchronously instead of synchronously:

```
public interface MyApiEndpointInterface {
    // Request method and URL specified in the annotation
    // Callback for the parsed response is the last parameter

    @GET("users/{username}")
    void getUser(@Path("username") String username, Callback<User>cb);

    @GET("group/{id}/users")
    void groupList(@Path("id") int groupId, @Query("sort") String sort, Callback<List<User>>cb);

    @POST("users/new")
    void createUser(@Body User user, Callback<User>cb);
}
```

Retrofit 1 relied on this Callback type as a last parameter to determine whether the API call would be dispatched asynchronously instead of synchronously. To avoid having two different calling patterns, this interface was consolidated in Retrofit 2. You now simply define the return value with a parameterized Call<T>, as shown in the previous section.

Accessing the API

We can bring this all together by constructing a service leveraging the MyApiEndpointInterface interface with the defined endpoints:

```
MyApiEndpointInterface apiService =
    retrofit.create(MyApiEndpointInterface.class);
```

If we want to consume the API asynchronously, we call the service as follows:

```
String username = "sarahjean";
Call<User> call = apiService.getUser(username);
call.enqueue(new Callback<User>() {
    @Override
    public void onResponse(Call<User>call, Response<User>response) {
        int statusCode = response.code();
        User user = response.body();
    }
});
```

```
}

@Override
public void onFailure(Call<User> call, Throwable t) {
    // Log error here since request failed
}
});
```

Shown above, Retrofit will download and parse the API data on a background thread, and then deliver the results back to the UI thread via the `onResponse` or `onFailure` method.

Note also that `OkHttp`, which dispatches the callback on the worker thread, callbacks in Retrofit are dispatched on the main thread. Because UI updates can only be done on the main thread, the approach used by Retrofit can make it easier to make changes to your views.

If you are using Retrofit in a Background Service instead of an Activity or Fragment, you can run the network call synchronously within the same thread by using the `execute()` method.

```
try {
    Response<User> response = call.execute();
} catch (IOException e) {
    // handle error
}
```

RxJava

Retrofit 2 also supports RxJava extensions. You will need to create an RxJava Adapter. By default, all network calls are synchronous:

```
RxJavaCallAdapterFactory rxAdapter = RxJava2CallAdapterFactory.create();
```

If you wish to default network calls to be asynchronous, you need to use `createWithScheduler()`.

```
RxJavaCallAdapterFactory rxAdapter
```

```
= RxJava2CallAdapterFactory.createWithScheduler(Schedulers.io());
```

You can then instantiate the Retrofit instance:

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com")
    .addConverterFactory(GsonConverterFactory.create())
    .addCallAdapterFactory(rxAdapter)
    .build();
```

Instead of creating `Call` objects, we will use `Observable` types.

```
public interface MyApiEndpointInterface {
```

```
    // Request method and URL specified in the annotation
```

```
    // Callback for the parsed response is the last parameter
```

```
    @GET("/users/{username}")
```

```
    Observable<User> getUser(@Path("username") String username);
```

```
    @GET("/group/{id}/users")
```

```
    Observable<List<User>> groupList(@Path("id") int groupId, @Query("sort") String sort);
```

```
    @POST("/users/new")
```

```
    Observable<User> createUser(@Body User user);
```

Course Manual for Android App Development

```
}
```

Consistent with the RxJava framework, we need to create a subscriber to handle the response. The methods `onCompleted()`, `onError()`, and `onNext()` need to be added. Using the Android RxJava library, we can also designate that we will handle this event on the UI main thread. **Note:** If you intend to override the default network call behavior, you can specify `subscribeOn()`. Otherwise, it can be omitted. **Note:** As the RxJava rewrote their API, the term "Subscription" used here shall be replaced with "Disposable". As the word "Subscription" had conflict intentions.

```
String username = "sarahjean";
Observable<User> call = apiService.getUser(username);
Disposable disposable = call
    .subscribeOn(Schedulers.io()) // optional if you do not wish to override the default behavior
    .observeOn(AndroidSchedulers.mainThread()).subscribeWith(new Disposable<User>() {
        @Override
        public void onCompleted() {

        }

    })
```

```
    @Override
    public void onError(Throwable e) {
        // cast to retrofit.HttpException to get the response code
        if (e instanceof HttpException) {
            HttpException response = (HttpException)e;
            int code = response.code();
        }
    }
}
```

```
    @Override
    public void onNext(User user) {
    }
});
```

Note that if you are running any API calls in an activity or fragment, you will need to unsubscribe on the `onDestroy()` method. The reasons are explained in this [Wiki article](#):

```
// MyActivity
private Disposable disposable;

protected void onCreate(Bundle savedInstanceState) {
    this.disposable = observable.subscribe(this);
}

...

protected void onDestroy() {
    this.disposable.dispose();
    super.onDestroy();
}
```

Note also that subscribing an observer to the observable is what triggers the network request. For more information about how to attach multiple observers before dispatching the network requests, see this [section](#).

Using Authentication Headers

Headers can be added to a request using an Interceptor. To send requests to an authenticated API, add headers to your requests using an interceptor as outlined below:

```
// Define the interceptor, add authentication headers
Interceptor interceptor = new Interceptor() {
    @Override
    public okhttp3.Response intercept(Chain chain) throws IOException {
        Request newRequest = chain.request().newBuilder().addHeader("User-Agent", "Retrofit-Sample-App").build();
        return chain.proceed(newRequest);
    }
};

// Add the interceptor to OkHttpClient
OkHttpClient.Builder builder = new OkHttpClient.Builder();
builder.interceptors().add(interceptor);
OkHttpClient client = builder.build();

// Set the custom client when building adapter
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com")
    .addConverterFactory(GsonConverterFactory.create())
    .client(client)
    .build();
```

Notice that in Retrofit 2 the interceptor has to be added to a custom OkHttpClient. In Retrofit 1, it could be set directly by the builder class.

9.6- Sending Email, SMS Phone Call

Email

Before starting Email Activity, You must know Email functionality with intent, Intent is carrying data from one component to another component with-in the application or outside the application.

To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc. For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients.

Following section explains different parts of our Intent object required to send an email.

Intent Object - Action to send Email

You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

Course Manual for Android App Development

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

Intent Object - Data/Type to send Email

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows –

```
emailIntent.setData(Uri.parse("mailto:"));  
emailIntent.setType("text/plain");
```

Intent Object - Extra to send Email

Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client. You can use following extra fields in your email –

Sr.No.	Extra Data & Description
1	EXTRA_BCC A String[] holding e-mail addresses that should be blind carbon copied.
2	EXTRA_CC A String[] holding e-mail addresses that should be carbon copied.
3	EXTRA_EMAIL A String[] holding e-mail addresses that should be delivered to.
4	EXTRA_HTML_TEXT A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
5	EXTRA_SUBJECT A constant string holding the desired subject line of a message.
6	EXTRA_TEXT A constant CharSequence that is associated with the Intent, used with ACTION_SEND

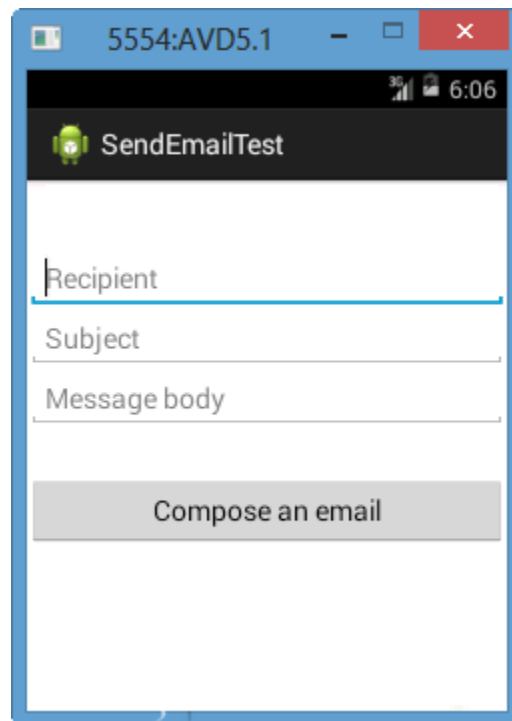
Course Manual for Android App Development

	to supply the literal data to be sent.
7	EXTRA_TITLE A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.

Here is an example showing you how to assign extra data to your intent –

```
emailIntent.putExtra(Intent.EXTRA_EMAIL , new String[]{"Recipient"});  
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");  
emailIntent.putExtra(Intent.EXTRA_TEXT , "Message Body");
```

The out-put of above code is as below shown an image



EMAIL EXAMPLE

Course Manual for Android App Development

Example

Following example shows you in practical how to use Intent object to launch Email client to send an Email to the given recipients.

To Email experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you might get struggle with emulator which may not work properly. Second you will need to have an Email client like GMail(By default every android version having Gmail client App) or K9mail installed on your device.

Step	Description
1	You will use Android studio to create an Android application and name it as <i>Ictdevition</i> under a package <i>com.example.ictdevition</i> .
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending email.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to launch Email Client.
4	Modify <i>res/values/strings.xml</i> to define required constant values
5	Modify <i>AndroidManifest.xml</i> as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.Ictdevition/MainActivity.java**.

```
package com.example.ictdevition;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
```

```
import android.content.Intent;

import android.util.Log;

import android.view.Menu;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;


public class MainActivity extends Activity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);


        Button startBtn = (Button) findViewById(R.id.sendEmail);

        startBtn.setOnClickListener(new View.OnClickListener() {

            public void onClick(View view) {

                sendEmail();

            }

        });

    }


    protected void sendEmail() {

        Log.i("Send email", "");

        String[] TO = {""};

        String[] CC = {""};

        Intent emailIntent = new Intent(Intent.ACTION_SEND);
```



```
emailIntent.setData(Uri.parse("mailto:"));

emailIntent.setType("text/plain");

emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);

emailIntent.putExtra(Intent.EXTRA_CC, CC);

emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Your subject");

emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message goes here");

try{

    startActivity(Intent.createChooser(emailIntent, "Send mail..."));

    finish();

    Log.i("Finished sending email...", "");

} catch (android.content.ActivityNotFoundException ex) {

    Toast.makeText(MainActivity.this, "There is no email client installed.", Toast.LENGTH_SHORT).show();

}

}

}
```

Following will be the content of **res/layout/activity_main.xml** file –

Here abc indicates about ictdevition logo

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    android:orientation="vertical">

    <TextView

        android:id="@+id/textView1"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"
```

```
android:text="Sending Mail Example"

android:layout_alignParentTop="true"

android:layout_centerHorizontal="true"

android:textSize="30dp"/>

<TextView

android:id="@+id/textView2"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="Tutorials point "

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout_above="@+id/imageButton"

android:layout_alignRight="@+id/imageButton"

android:layout_alignEnd="@+id/imageButton"/>

<ImageButton

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout_centerVertical="true"

android:layout_centerHorizontal="true"/>

<Button

android:id="@+id/sendEmail"

android:layout_width="fill_parent"
```

Course Manual for Android App Development

```
android:layout_height="wrap_content"

android:text="@string/compose_email"/>

</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants –

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

<resources>

<stringname="app_name">Ictdevition</string>

<stringname="compose_email">Compose Email</string>

</resources>
```

Following is the default content of **AndroidManifest.xml** –

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

<manifestxmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.Ictdevition">

<application

android:allowBackup="true"

android:icon="@drawable/ic_launcher"

android:label="@string/app_name"

android:theme="@style/AppTheme">

<activity

android:name="com.example.ictdevition.MainActivity"

android:label="@string/app_name">

<intent-filter>
```

```
<actionandroid:name="android.intent.action.MAIN"/>


<categoryandroid:name="android.intent.category.LAUNCHER"/>

</intent-filter>

</activity>

</application>

</manifest>
```

Let's try to run your **ictdevition** application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android Studio, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Android studio installer will display following window to select an option where you want to run your Android application.

SMS

In Android, you can use SmsManager API or devices Built-in SMS application to send SMS's. In this tutorial, we shows you two basic examples to send SMS message –

SmsManager API

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

Built-in SMS application

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

Of course, both need **SEND_SMS permission**.

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below –

Sr.No.	Method & Description
--------	----------------------

Course Manual for Android App Development

1	<code>ArrayList<String> divideMessage(String text)</code> This method divides a message text into several fragments, none bigger than the maximum SMS message size.
2	<code>static SmsManager getDefault()</code> This method is used to get the default instance of the SmsManager
3	<code>void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)</code> This method is used to send a data based SMS to a specific application port.
4	<code>void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)</code> Send a multi-part text based SMS.
5	<code>void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)</code> Send a text based SMS.

Phone Call

Android provides Built-in applications for phone calls, in some occasions we may need to make a phone call through our application. This could easily be done by using implicit Intent with appropriate actions. Also, we can use PhoneStateListener and TelephonyManager classes, in order to monitor the changes in some telephony states on the device.

This chapter lists down all the simple steps to create an application which can be used to make a Phone Call. You can use Android Intent to make phone call by calling built-in Phone Call functionality of the Android. Following section explains different parts of our Intent object required to make a call.

Intent Object - Action to make Phone Call

You will use **ACTION_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION_CALL action

```
Intent phoneIntent =newIntent(Intent.ACTION_CALL);
```

You can use **ACTION_DIAL** action instead of ACTION_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

Intent Object - Data/Type to make Phone Call

To make a phone call at a given number 91-000-000-0000, you need to specify **tel:** as URI using setData() method as follows –

```
phoneIntent.setData(Uri.parse("tel:91-000-000-0000"));
```

The interesting point is that, to make a phone call, you do not need to specify any extra data or data type.

Example

Following example shows you in practical how to use Android Intent to make phone call to the given mobile number.

Course Manual for Android App Development

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as <i>My Application</i> under a package <i>com.example.saira_000.myapplication</i> .
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of making a call.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to Call 91-000-000-0000 number
4	No need to define default string constants.Android studio takes care of default constants.
5	Modify <i>AndroidManifest.xml</i> as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.saira_000.myapplication;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
```

Course Manual for Android App Development

```
import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        button = (Button) findViewById(R.id.buttonCall);

        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View arg0) {
                Intent callIntent = new Intent(Intent.ACTION_CALL);

                callIntent.setData(Uri.parse("tel:0377778888"));

                if (ActivityCompat.checkSelfPermission(MainActivity.this,
                    Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
                    return;
                }

                startActivity(callIntent);
            }
        });
    }
}
```


Course Manual for Android App Development

Following will be the content of **res/layout/activity_main.xml** file –

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

<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    android:orientation="vertical">


    <Button

        android:id="@+id/buttonCall"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="call 0377778888"/>


    </LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants –

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

<resources>

    <stringname="app_name">My Application</string>

</resources>
```

Following is the default content of **AndroidManifest.xml** –

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

<manifestxmlns:android="http://schemas.android.com/apk/res/android"

    package="com.example.saira_000.myapplication">


    <uses-permissionandroid:name="android.permission.CALL_PHONE"/>
```

Course Manual for Android App Development


```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">

    <activity
        android:name="com.example.saira_000.myapplication.MainActivity"
        android:label="@string/app_name">

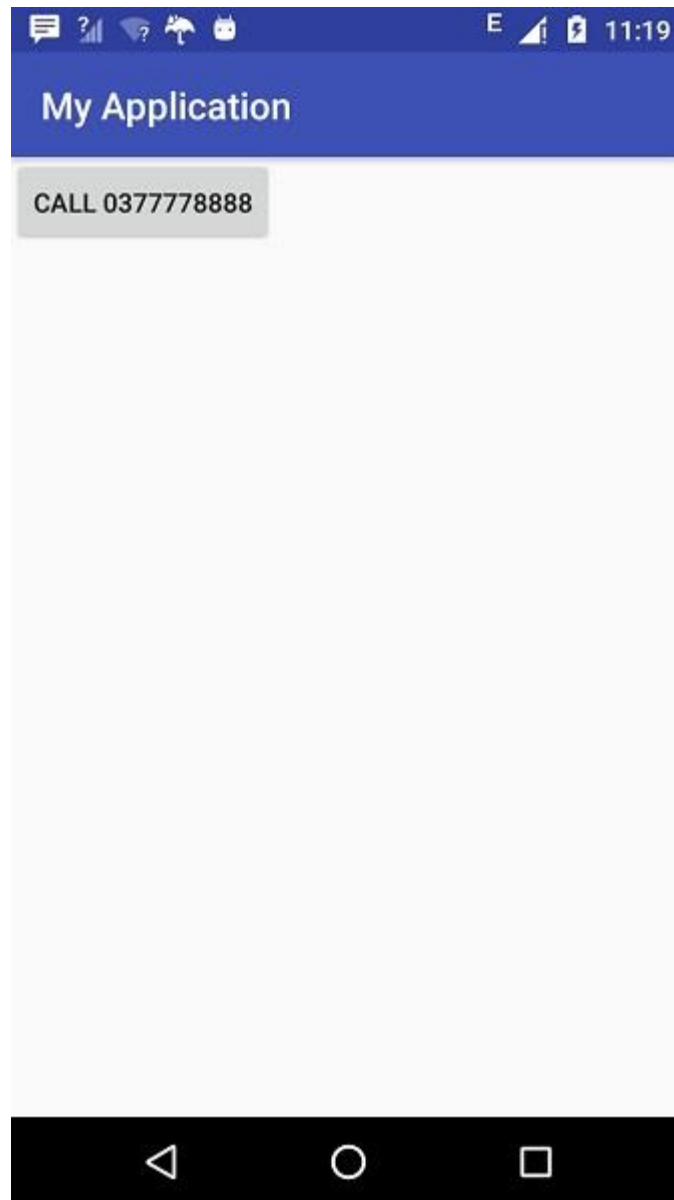
        <intent-filter>
            <actionandroid:name="android.intent.action.MAIN"/>
            <categoryandroid:name="android.intent.category.LAUNCHER"/>
        </intent-filter>

    </activity>

</application>
</manifest>
```

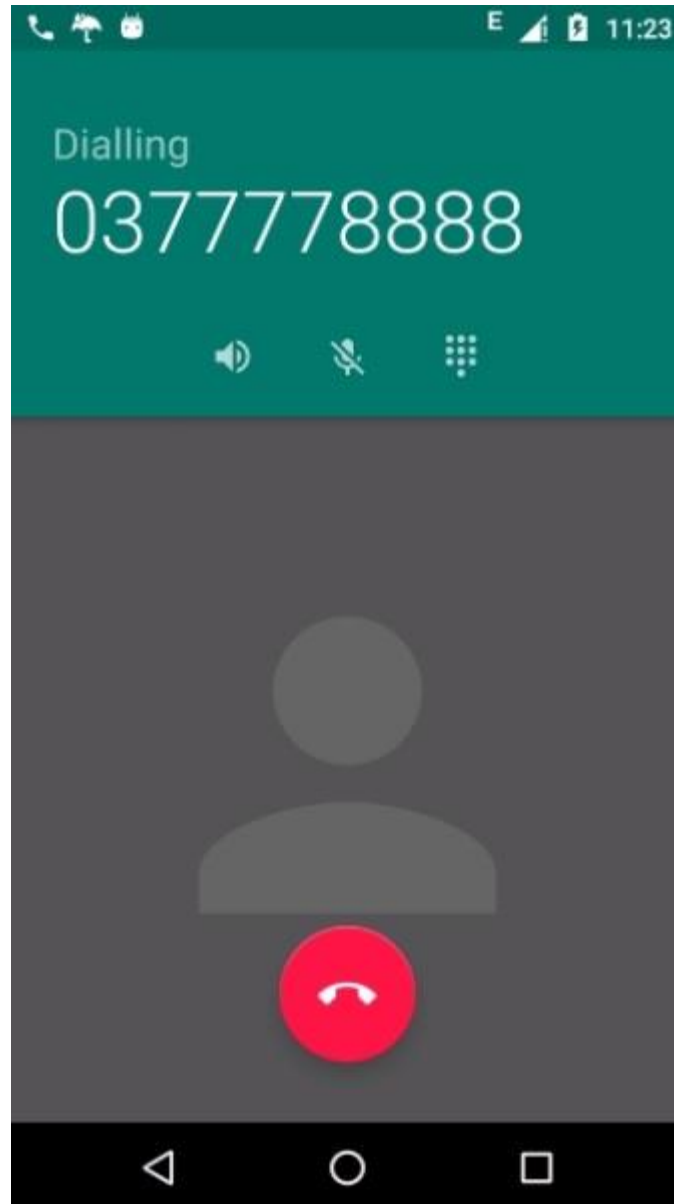
Let's try to run your **My Application** application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run  icon from the toolbar. Select your mobile device as an option and then check your mobile device which will display following screen –

Course Manual for Android App Development



Now use **Call** button to make phone call as shown below –

Course Manual for Android App Development



9.7- Simple app using the module taught. E.g.: Weather Forecasting (Practical)

Course Manual for Android App Development

PRACTICE:

Response Handling->Handling Network Sign-On-> music Files ->HTTP Authentication

- ->Executing an AsyncTask->HandlerThread->RxJava->Email->SMS->Phone Call

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<https://www.androidhive.info/2012/01/android-json-parsing-tutorial/>

<https://www.journaldev.com/13639/retrofit-android-example-tutorial>

<https://developer.android.com/training/volley/>

Course Manual for Android App Development

MODULE 10: Android Storage and Content Providers

CHAPTER OVERVIEW:

A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object. Together, providers and provider clients offer a consistent, standard interface to data that also handles inter-process communication and secure data access.

10.1- Using the SQLite Database

10.2- Content Provider Basics

10.3- Loaders

10.4- Firebase Cloud Storage, Database and Authentication

10.5- Firebase Notification and Analytics

10.6- AdMob Integration

10.7- Simple app using the module taught. Ex: Quiz app using Firebase

OBJECTIVES:

- Saving data to a database is ideal for repeating or structured data, such as contact information
- Cloud Storage is built for app developers who need to store and serve user-generated content, such as photos or videos.
- AdMob Integration

LEARNING OUTCOME:

- Save Data using SQLite
- What is SQLite?
- Why SQLite?
- How to manage SQLite database
- Accessing a provider
- Content URIs
- Cloud Storage
- Firebase Real-time Database
- Firebase Authentication
- Firebase Cloud Messaging and Notification
- Google Analytics for Firebase
- Import the Mobile Ads SDK

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Firebase Notification and Analytics

CORE DISCUSSION:

- **Fundamental technology**

10.1- Using the SQLite Database:

Save Data using SQLite

This page assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android. The APIs you'll need to use a database on Android are available in the `android.database.sqlite` package.

Overview on SQLite and Database design

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is one of the fastest-growing database engines around, but that's growth in terms of popularity, not anything to do with its size. The source code for SQLite is in the public domain.

What is SQLite?

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not need to configure it in your system.

SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. SQLite accesses its storage files directly.

Why SQLite?

- SQLite does not require a separate server process or system to operate (serverless).
- SQLite comes with zero-configuration, which means no setup or administration needed.
- A complete SQLite database is stored in a single cross-platform disk file.
- SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.

Course Manual for Android App Development

- SQLite is self-contained, which means no external dependencies.
- SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
- SQLite supports most of the query language features found in SQL92 (SQL2) standard.
- SQLite is written in ANSI-C and provides simple and easy-to-use API.
- SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

There are few unsupported features of SQL92 in SQLite which are listed in the following table.

Sr.No.	Feature & Description
1	RIGHT OUTER JOIN Only LEFT OUTER JOIN is implemented.
2	FULL OUTER JOIN Only LEFT OUTER JOIN is implemented.
3	ALTER TABLE The RENAME TABLE and ADD COLUMN variants of the ALTER TABLE command are supported. The DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT are not supported.
4	Trigger support FOR EACH ROW triggers are supported but not FOR EACH STATEMENT triggers.
5	VIEWS VIEWS in SQLite are read-only. You may not execute a DELETE, INSERT, or UPDATE statement on a view.

Course Manual for Android App Development

6	GRANT and REVOKE The only access permissions that can be applied are the normal file access permissions of the underlying operating system.
---	---

SQLite Commands

The standard SQLite commands to interact with relational databases are similar to SQL. They are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their operational nature –

DDL - Data Definition Language

Sr.No.	Command & Description
1	CREATE Creates a new table, a view of a table, or other object in database.
2	ALTER Modifies an existing database object, such as a table.
3	DROP Deletes an entire table, a view of a table or other object in the database.

Course Manual for Android App Development

DML - Data Manipulation Language

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DQL - Data Query Language

Sr.No.	Command & Description
1	SELECT Retrieves certain records from one or more tables

How to manage SQLite database

SQLiteManager is a database manager for SQLite databases. You can manage any SQLite database created on any platform with SQLiteManager. SQLiteManager displays an entire database in one window.

Many operations do display dialog boxes to complete, but the main browsing and manipulation functions all happen in one window. The main window has a tab panel that takes you to Design, Manage, and SQL panels. From here, the Design panel lets you browse the objects in your database.

10.2- Content Provider Basics:

A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object. Together, providers and provider clients offer a consistent, standard interface to data that also handles inter-process communication and secure data access.

Typically you work with content providers in one of two scenarios; you may want to implement code to access an existing content provider in another application, or you may want to create a new content provider in your application to share data with other applications. This topic covers the basics of working with existing content providers.

- Overview

A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database. A row represents an instance of some type of data the provider collects, and each column in the row represents an individual piece of data collected for an instance.

A content provider coordinates access to the data storage layer in your application for a number of different APIs and components as illustrated in figure 1, these include:

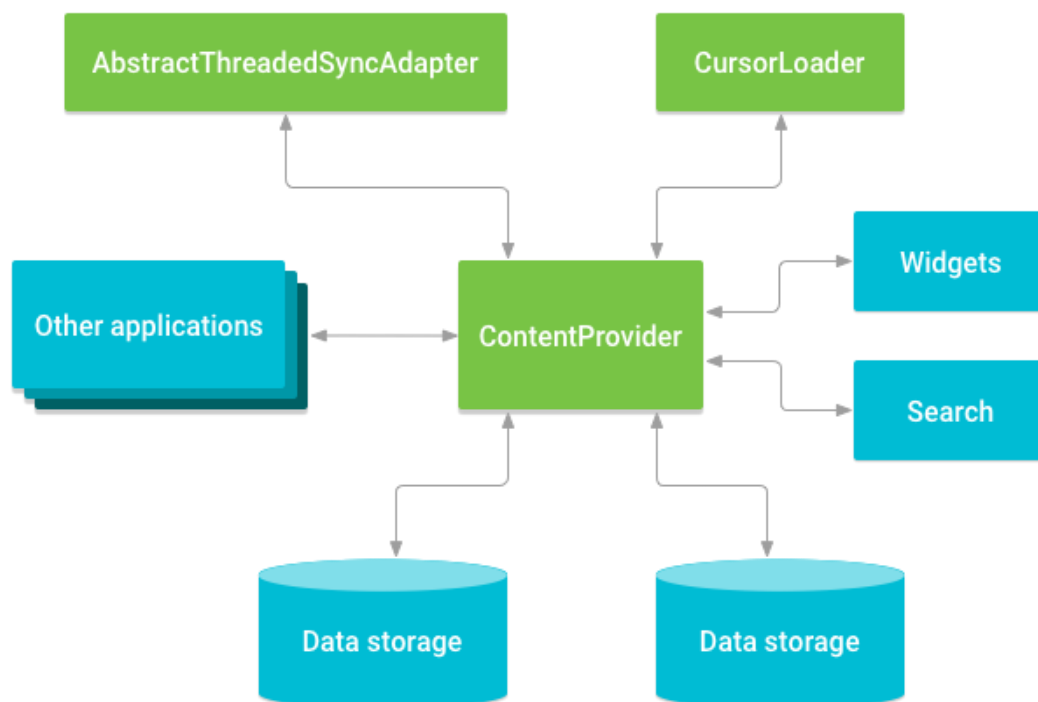


Figure: 1

Course Manual for Android App Development

- Sharing access to your application data with other applications
- Sending data to a widget
- Returning custom search suggestions for your application through the search framework using `SearchRecentSuggestionsProvider`
- Synchronizing application data with your server using an implementation of `AbstractThreadedSyncAdapter`
- Loading data in your UI using a `CursorLoader`

- Accessing a provider

When you want to access data in a content provider, you use the `ContentResolver` object in your application's `Context` to communicate with the provider as a client. The `ContentResolver` object communicates with the provider object, an instance of a class that implements `ContentProvider`. The provider object receives data requests from clients, performs the requested action, and returns the results. This object has methods that call identically-named methods in the provider object, an instance of one of the concrete subclasses of `ContentProvider`. The `ContentResolver` methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

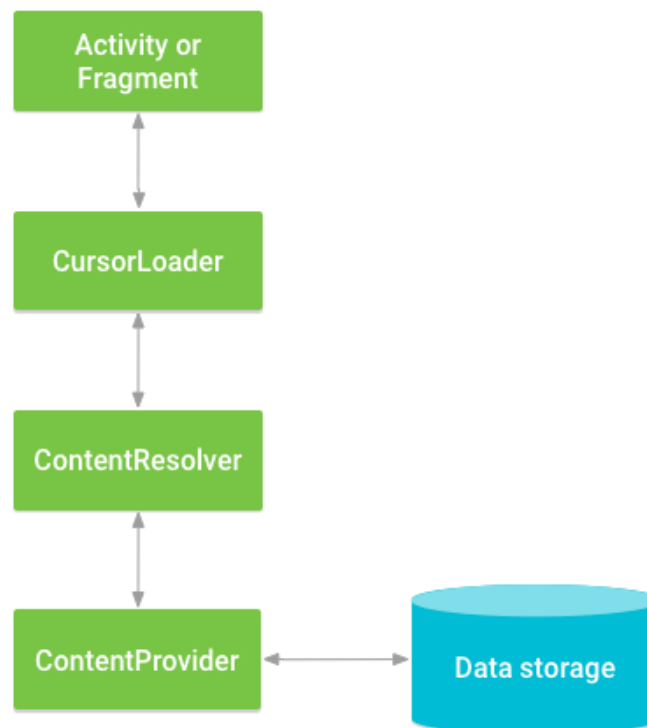


Figure: 2

Course Manual for Android App Development

A common pattern for accessing a ContentProvider from your UI uses a CursorLoader to run an asynchronous query in the background. The Activity or Fragment in your UI call a CursorLoader to the query, which in turn gets the ContentProvider using the ContentResolver. This allows the UI to continue to be available to the user while the query is running. This pattern involves the interaction of a number of different objects, as well as the underlying storage mechanism, as illustrated in figure 2.

- Content URIs

Table 1: Sample user dictionary table.

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the words table
    mProjection,                      // The columns to return for each row
    mSelectionClause,                 // Selection criteria
    mSelectionArgs,                   // Selection criteria
    mSortOrder);                     // The sort order for the returned rows
```

Course Manual for Android App Development

Table 2 shows how the arguments to `query(Uri,projection,selection,selectionArgs, sortOrder)` match an SQL SELECT statement:

query() argument	SELECT keyword/parameter	Notes
<code>Uri</code>	<code>FROM table_name</code>	<code>Uri</code> maps to the table in the provider named <code>table_name</code> .
<code>projection</code>	<code>col,col,col,...</code>	<code>projection</code> is an array of columns that should be included for each row retrieved.
<code>selection</code>	<code>WHERE col = value</code>	<code>selection</code> specifies the criteria for selecting rows.
<code>selectionArgs</code>	(No exact equivalent. Selection arguments replace ? placeholders in the selection clause.)	
<code>sortOrder</code>	<code>ORDER BY col,col,...</code>	<code>sortOrder</code> specifies the order in which rows appear in the returned <code>Cursor</code> .

- Content URIs

A **content URI** is a URI that identifies data in a provider. Content URIs include the symbolic name of the entire provider (its **authority**) and a name that points to a table (a **path**). When you call a client method to access a table in a provider, the content URI for the table is one of the arguments.

In the preceding lines of code, the constant `CONTENT_URI` contains the content URI of the user dictionary's "words" table. The `ContentResolver` object parses out the URI's authority, and uses it to "resolve" the provider by comparing the authority to a system table of known providers. The `ContentResolver` can then dispatch the query arguments to the correct provider.

The `ContentProvider` uses the path part of the content URI to choose the table to access. A provider usually has a **path** for each table it exposes.

In the previous lines of code, the full URI for the "words" table is:

```
content://user_dictionary/words
```

where the `user_dictionary` string is the provider's authority, and the `words` string is the table's path. The string `content://` (the **scheme**) is always present, and identifies this as a content URI.

Course Manual for Android App Development

Many providers allow you to access a single row in a table by appending an ID value to the end of the URI. For example, to retrieve a row whose `_ID` is 4 from user dictionary, you can use this content URI:

```
Uri singleUri =ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);
```

10.3- Loaders:

The Loader API lets you load data from a content provider or other data source for display in an `FragmentActivity` or `Fragment`. If you don't understand why you need the Loader API to perform this seemingly trivial operation, then first consider some of the problems you might encounter without loaders:

- If you fetch the data directly in the activity or fragment, your users will suffer from lack of responsiveness due to performing potentially slow queries from the UI thread.
- If you fetch the data from another thread, perhaps with `AsyncTask`, then you're responsible for managing both the thread and the UI thread through various activity or fragment lifecycle events, such as `onDestroy()` and configuration changes.

Loaders solve these problems and includes other benefits. For example:

- Loaders run on separate threads to prevent janky or unresponsive UI.
- Loaders simplify thread management by providing callback methods when events occur.
- Loaders persist and cache results across configuration changes to prevent duplicate queries.
- Loaders can implement an observer to monitor for changes in the underlying data source. For example, `CursorLoader` automatically registers a `ContentObserver` to trigger a reload when data changes.

10.4- Firebase Cloud Storage, Database and Authentication:



- Cloud Storage

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content. On the server, you can use [Google Cloud Storage](#), to access the same files.

- Firebase Real-time Database

Store and sync data with our NoSQL cloud database. Data is synced across all clients in realtime, and remains available when your app goes offline. The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

- Firebase Authentication

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

10.5- Firebase Notification and Analytics:

- Firebase Cloud Messaging and Notification

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost.

Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.

- Google Analytics for Firebase

Google Analytics for Firebase is a free app measurement solution that provides insight on app usage and user engagement.

At the heart of Firebase is Google Analytics for Firebase, a free and unlimited analytics solution. Analytics integrates across Firebase features and provides you with unlimited reporting for up to 500 distinct events that you can define using the Firebase SDK. Analytics reports help you understand clearly how your users behave, which enables you to make informed decisions regarding app marketing and performance optimizations.

10.6- AdMob Integration:

AdMob by Google helps you monetize your mobile app through in-app advertising. Ads can be displayed as banner ads, interstitial ads, video ads, or native ads—which are seamlessly added to platform native UI components. On Android, you can additionally display in-app purchase ads, which allow users to purchase advertised products from within your app.

Before you can display ads within your app, you'll need to create an AdMob account and activate one or more Ad Unit IDs. This is a unique identifier for the places in your app where ads are displayed.

AdMob uses the Google Mobile Ads SDK. The Google Mobile Ads SDK helps app developers gain insights about their users, drive more in-app purchases, and maximize ad revenue. In order to do so, the default integration of the Mobile Ads SDK collects information such as device information, publisher-provided location information, and general in-app purchase information such as item purchase price and currency.

Course Manual for Android App Development

Import the Mobile Ads SDK

Apps can import the Google Mobile Ads SDK with a Gradle dependency that points to Google's Maven repository. In order to use that repository, you need to reference it in the app's project-level `build.gradle` file. Open yours and look for an `allprojects` section:

Example project-level `build.gradle` (excerpt)

```
allprojects {  
    repositories {  
        jcenter()  
        maven {  
            url "https://maven.google.com"  
        }  
    }  
}
```

Add the `maven` directive above if it's not already present.

Next, open the app-level `build.gradle` file for your app, and look for a "dependencies" section.

Example app-level `build.gradle` (excerpt)

```
dependencies {  
    compile fileTree(dir:'libs', include:['*.jar'])  
    compile 'com.android.support:appcompat-v7:26.1.0'  
    compile 'com.google.android.gms:play-services-ads:12.0.1'  
}
```

Add the line in bold above, which instructs Gradle to pull in the latest version of the Mobile Ads SDK. Once that's done, save the file and perform a Gradle sync.

To learn more visit: <https://developers.google.com/admob/android/quick-start>

Course Manual for Android App Development

10.7- Simple app using the module taught. Ex: Quiz app using Firebase:

PRACTICE:

SQLite database->Firebase Real-time Database
Firebase Cloud Messaging and Notification->Import the Mobile Ads SDK

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<https://firebase.google.com/docs/analytics/>

<https://www.androidhive.info/2016/02/android-how-to-integrate-google-admob-in-your-app/>

<https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/>

Course Manual for Android App Development

MODULE 11: Android Services and Google APIs

CHAPTER OVERVIEW:

At the heart of Firebase is Google Analytics for Firebase, a free and unlimited analytics solution. Analytics integrates across Firebase features and provides you with unlimited reporting for up to 500 distinct events that you can define using the Firebase SDK. Analytics reports help you understand clearly how your users behave, which enables you to make informed decisions regarding app marketing and performance optimizations.

11.1- Android Services

11.2- Location and Map API

11.3- Marker, Marker Options and Clustering

11.4- Google Direction API

11.5- Google Geocoding and GeoFencing

11.6- Simple App Using the Module Taught

OBJECTIVES:

- A Service is an application component that can perform long-running operations in the background.
- The Google Places API for Android

LEARNING OUTCOME:

- Google Places API for Android
- Google Map API for Android
- Add a marker
- Customize a marker
- Google Maps Android Marker Clustering Utility
- Geofencing

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Google Direction API

CORE DISCUSSION:

- **Fundamental technology**

11.1- Android Services:

A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:

Foreground

A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services must display a Notification. Foreground services continue running even when the user isn't interacting with the app.

Background

A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

Note: If your app targets API level 26 or higher, the system imposes restrictions on running background services when the app itself is not in the foreground. In most cases like this, your app should use a scheduled job instead.

Bound

A service is *bound* when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Although this documentation generally discusses started and bound services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding. It's simply a matter of whether you implement a couple of callback methods: `onStartCommand()` to allow components to start it and `onBind()` to allow binding.

Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application) in the same way that any component can use an activity—by starting it with an Intent. However, you can declare the service as *private* in the manifest file and block access from other

Course Manual for Android App Development

applications. This is discussed more in the section about Declaring the service in the manifest.

- There can be two forms of a service. The lifecycle of service can follow two different paths: started or bound.

1. Started
2. Bound

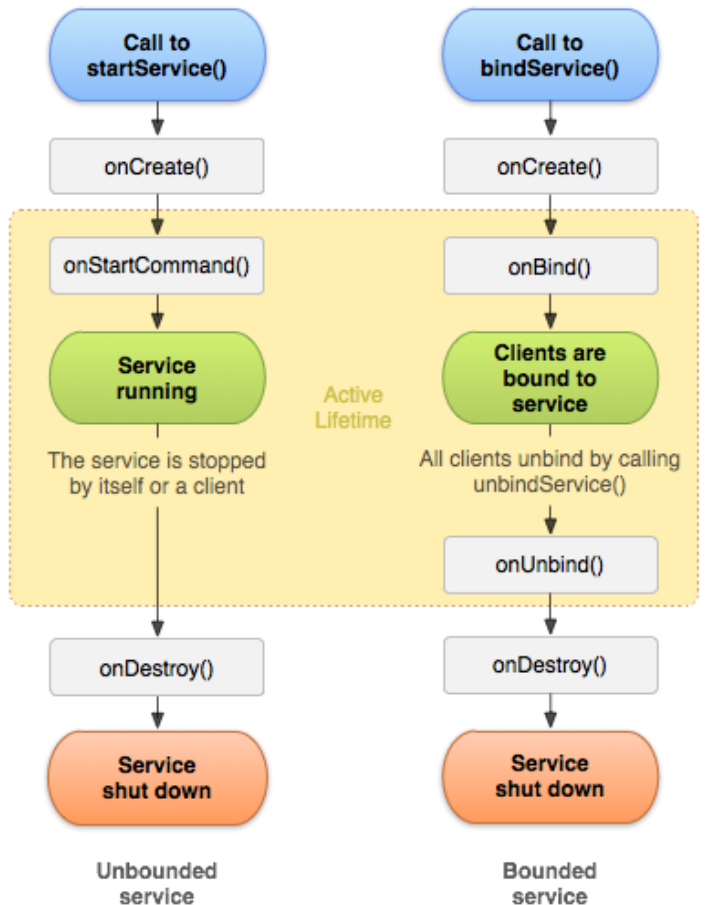
1) Started Service

A service is started when component (like activity) calls **startService()** method, now it runs in the background indefinitely. It is stopped by **stopService()** method. The service can stop itself by calling the **stopSelf()** method.

2) Bound Service

A service is bound when another component (e.g. client) calls **bindService()** method. The client can unbind the service by calling the **unbindService()** method.

The service cannot be stopped until all clients unbind the service.



11.2- Location and Map API:

- Google Places API for Android

The Google Places API for Android allows you to build location-aware apps that respond contextually to the local businesses and other places near the device. This means you can build rich apps based on places that mean something to the user, to complement the straightforward geographic-based services offered by the Android location services.

Course Manual for Android App Development

API overview

Help your customers explore where they are and what's around them:

- Place Picker allows users to select a place on an interactive map.
- Current Place returns a list of places where the user's device is last known to be located along with an indication of the relative likelihood for each place.
- Place Autocomplete automatically fills in the name and/or address of a place as users type.
- Place Photos returns high-quality images of a place.
- Place IDs and Details return and display more detailed information about a place.

Differentiate your app by supplying up-to-date local information:

- Place Add adds a place to Google's Places database, for immediate retrieval from within your own app and for visibility with other apps after moderation.
- Place Report creates a report indicating the current location of the device, helping Google build a local model of the world.
- Place IDs stores the unique ID for one or more places for retrieval of place information on demand.

- Google Map API for Android

With the Google Maps Android API, you can add maps based on Google Maps data to your application. The API automatically handles access to Google Maps servers, data downloading, map display, and response to map gestures. You can also use API calls to add markers, polygons, and overlays to a basic map, and to change the user's view of a particular map area. These objects provide additional information for map locations, and allow user interaction with the map. The API allows you to add these graphics to a map:

- Icons anchored to specific positions on the map (Markers).
- Sets of line segments (Polylines).
- Enclosed segments (Polygons).
- Bitmap graphics anchored to specific positions on the map (Ground Overlays).
- Sets of images which are displayed on top of the base map tiles (Tile Overlays).

11.3- Marker, Marker Options and Clustering:

Markers identify locations on the map. The default marker uses a standard icon, common to the Google Maps look and feel. It's possible to change the icon's color, image or anchor point via the API. Markers are objects of type `Marker`, and are added to the map with the `GoogleMap.addMarker(markerOptions)` method.

Markers are designed to be interactive. They receive click events by default, and are often used with event listeners to bring up info windows. Setting a marker's `draggable` property to `true` allows the user to change the position of the marker. Use a long press to activate the ability to move the marker.

By default, when a user taps a marker, the map toolbar appears at the bottom right of the map, giving the user quick access to the Google Maps mobile app. You can disable the toolbar. For more information, see the guide to controls.

Add a marker

The following example demonstrates how to add a marker to a map. The marker is created at coordinates 10,10, and displays the string 'Hello world' in an info window when clicked.

```
@Override
public void onMapReady(GoogleMap map){
    map.addMarker(new MarkerOptions()
        .position(new LatLng(10,10))
        .title("Hello world"));
}
```

Customize a marker

```
private static final LatLng MELBOURNE = new LatLng(-37.813, 144.962);
private Marker melbourne = mMap.addMarker(new MarkerOptions()
    .position(MELBOURNE)
    .title("Melbourne")
    .snippet("Population: 4,137,400")
    .icon(BitmapDescriptorFactory.fromResource(R.drawable.arrow)));
```


Google Maps Android Marker Clustering Utility

The marker clustering utility helps you manage multiple markers at different zoom levels. To be precise, the 'markers' are actually 'items' at this point, and only become 'Markers' when they're rendered. But for the sake of clarity, this document will name them 'markers' throughout.

When a user views the map at a high zoom level, the individual markers show on the map. When the user zooms out, the markers gather together into clusters, to make viewing the map easier.

11.4- Google Direction API:

The API returns the most efficient routes when calculating directions. Travel time is the primary factor optimized, but the API may also take into account other factors such as distance, number of turns and many more when deciding which route is the most efficient. Calculating directions is a time and resource intensive task. Whenever possible, use the service described here to calculate known addresses ahead of time and store the results in a temporary cache of your own design.

11.5- Google Geocoding and GeoFencing:

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

You can also use the Google Maps Geocoding API to find the address for a given place ID. Geocoding request and response (latitude/longitude lookup). The following example requests the latitude and longitude of "1600 Amphitheatre Parkway, Mountain View, CA", and specifies that the output must be in JSON format.

```
https://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&key=YOUR_API_KEY
```

Below is a sample reverse geocoding response, in JSON: {

```
"results":[
  {
    "address_components":[
      {
        "long_name":"1600",
        "short_name":"1600",
```

Course Manual for Android App Development

```
    "types":["street_number"]
  },
  {
    "long_name":"Amphitheatre Pkwy",
    "short_name":"Amphitheatre Pkwy",
    "types":["route"]
  }, .....
}
```

Geofencing

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

PRACTICE:

Places API ->Map API ->Customize a marker

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<https://developers.google.com/maps/documentation/directions/intro>

<https://developer.android.com/training/location/geofencing.html>

MODULE 12: Custom Widgets and the Canvas API

CHAPTER OVERVIEW:

The Canvas class holds the "draw" calls. To draw something, you need 4 basic components: A Bitmap to hold the pixels, a Canvas to host the draw calls (writing into the bitmap), a drawing primitive (e.g. Rect, Path, text, Bitmap), and a paint (to describe the colors and styles for the drawing).

12.1- Defining custom widget

12.2- Custom View Components

12.3- Compound Widgets

12.4- Canvas API

12.5- Persisting view data

12.6- Simple App Using the Module Taught

OBJECTIVES:

- Android offers a sophisticated and powerful componentized model for building your UI.
- Android Canvas API

LEARNING OUTCOME:

- Canvas API
- Persisting view data

CORE DISCUSSION:

- **Fundamental technology**

12.1- Defining custom widget:

Sometimes you want to show a certain type of data and there is already a suitable view in the basic widget set. But if you want UI customization or a different user interaction, you may need to extend a widget. Suppose that there were no Button widget in the basic widget set in the Android SDK and you want to make one. You would extend the **TextView** class to get all the capabilities related to the text like setting text, text color, text size, and text style and so on. Then you will start your customization work, to give your new widget the look and feel of a button. This is what happens in the Android SDK the Button class extends the **TextView** class. Or you could in theory extend the View class to start from scratch.

12.2- Custom View Components:

Android offers a sophisticated and powerful componentized model for building your UI, based on the fundamental layout classes: View and ViewGroup. To start with, the platform includes a variety of prebuilt View and ViewGroup subclasses — called widgets and layouts, respectively — that you can use to construct your UI.

A partial list of available widgets includes Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, and the more special-purpose AutoCompleteTextView, ImageSwitcher, and TextSwitcher.

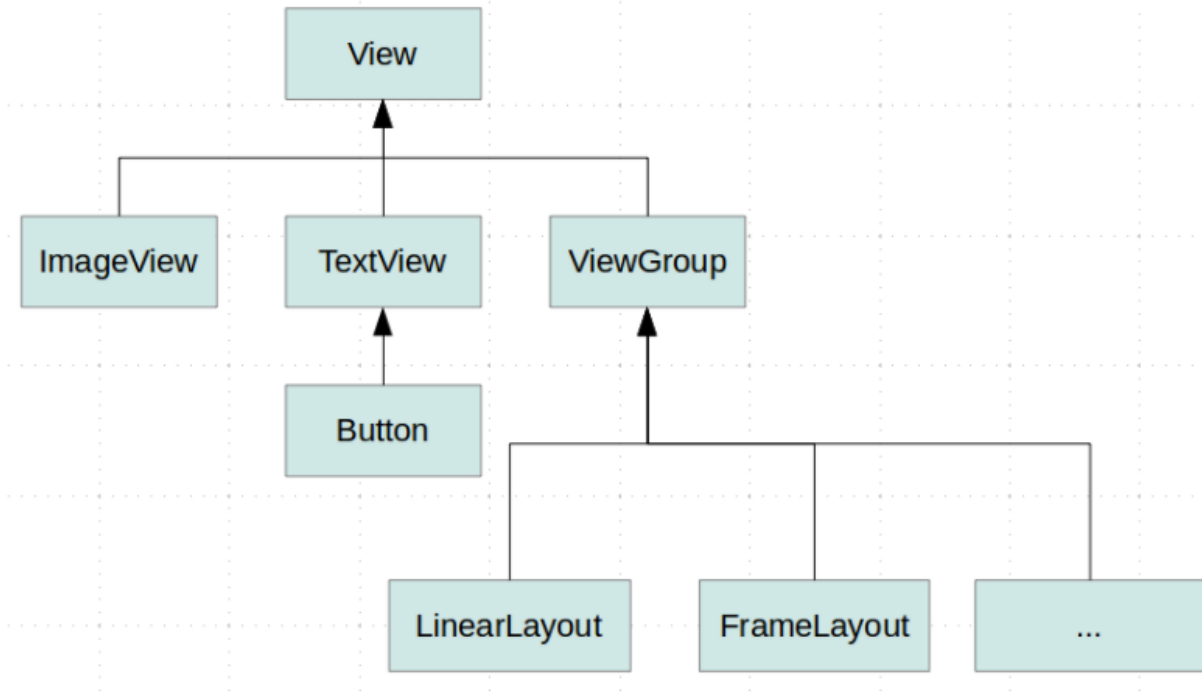
Among the layouts available are LinearLayout, FrameLayout, RelativeLayout, and others. For more examples, see Common Layout Objects.

The Android framework provides several default views. The base class a view is the View. Views are responsible for measuring, layouting and drawing themselves and their child elements (in case of a ViewGroup). Views are also responsible for saving their UI state and handling touch events. Developers can also create custom views and use them in their application.

It is possible to create custom views by:

- Compound views - combining views with a default wiring
- Custom views - creating your own views
 - by extending an existing view, e.g. Button
 - by extending the View class

The following image shows the default view hierarchy of Android.



Views are typically created to provide a user interface experience which is not possible with the default views. Using custom views allows the developer to do certain performance optimization, i.e., in case of a custom layout the development can optimize the layout manager for his use case.

12.3- Compound Widgets:

Compound views (also known as *Compound Components*) are pre-configured `ViewGroups` based on existing views with some predefined view interaction.

Compound views also allow you to add custom API to update and query the state of the compound view.

For such a control you define a layout file and assign it to your compound view. In the implementation of your compound view you predefine the view interaction. You would define a layout file and extend the corresponding `ViewGroup` class. In this class you inflate the layout file and implement the View connection logic.

12.4- Canvas API:

The Canvas API allows to create complex graphical effects. You paint on a Bitmap surface.

The Canvas class provides the drawing methods to draw on a bitmap. The Paint class specifies how you draw on the bitmap.

The Canvas object contains the bitmap on which you draw. It also provides methods for drawing operations:

- drawARGB() for drawing a color
- drawBitmap() to draw a Bitmap
- drawText() to draw a text
- drawRoundRect() to draw a rectangle with rounded corners and much more.

For drawing on the Canvas object you use an object of type Paint.

The Paint class allows to specify the color, font and certain effects for the drawing operation.

The setStyle() method allows to specify how it should be drawn. Option are to paint: * Only the outline (Paint.Style.STROKE) * the filled part (Paint.Style.FILL) * both the outline and the filled part (Paint.Style.STROKE_AND_FILL)

You can set the alpha channel of the Paint via the setAlpha() method.

Via *Shaders* you can define that the Paint is filled with more than one color.

12.5- Persisting view data:

Most standard views can save their state so that it can be persisted by the system. The Android system calls the `onSaveInstanceState()` method and `onRestoreInstanceState()` to save and restore the view state.

The convention is to extend `View.BaseSavedState` as a static inner class in the view for persisting the data.

Android searches based on the ID of the view in the layout for the view and pass a `Bundle` to the view which the view can use to restore its state.

You should save and restore the user interface state as the user left it, e.g. the scroll position or the active selection.

PRACTICE:

Canvas API -> Persisting view data

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFERENCE LINK:

<https://developer.android.com/guide/topics/ui/custom-components.html>

Course Manual for Android App Development

MODULE 13: Advanced Android Concepts

CHAPTER OVERVIEW:

Advanced Android Development is an instructor-led course created by the Google Developers Training team. Developers taking the course learn advanced Android programming concepts and build a variety of apps.

13.1- Meaningful motions

13.2- Android Gradle

13.3- Espresso (Android Testing Support Library)

13.4- Configuration of the Gradle build file for Espresso

13.5- Adaptive Design

13.6- Creating an Android Wear Interface for Weather App

OBJECTIVS:

- Android includes the transitions framework.
- Android Gradle
- Espresso automatically synchronizes your test actions with the user interface of your application.

LEARNING OUTCOME:

- Alternative layouts
- Avoid hard-coded layout sizes
- Use the smallest width qualifier

NEW TERAMS INTRUDUCE IN THE CHAPTER:

- Meaningful motions
- - Espresso (Android Testing Support Library)

CORE DISCUSSION:

- **Fundamental technology**

13.1- Meaningful motions:

When your UI changes in response to user action, you should animate the layout transitions. These animations give users feedback on their actions and help keep them oriented to the UI.

Android includes the *transitions framework*, which enables you to easily animate changes between two view hierarchies. The framework animates the views at runtime by changing some of their property values over time. The framework includes built-in animations for common effects and lets you create custom animations and transition lifecycle callbacks.

Animations can add visual cues that notify users about what's going on in your app. They are especially useful when the UI changes state, such as when new content loads or new actions become available. Animations also add a polished look to your app, which gives it a higher quality look and feel.

There are 3 types of Animations:

1. *Property Animations*—They are used to alter property of objects (Views or non view objects). We specify certain properties (like `translateX`, `TextScaleX`) of the objects to change. Various characteristics of animation which can be manipulated are animation duration, whether to reverse it and for how many times we want to repeat the animation etc. They were introduced in Android 3.0 (API level 11).
2. *View Animations*—They are used to do simple animations like changing size, position, rotation, control transparency. They are easy to build and are very fast but have their own constraints. For eg—Their state changes but their property does not change. View animations will be covered in part 2.
3. *Drawable Animations*—This is used to do animation using drawables. An XML file specifying various list of drawables is made which are run one by one just like a roll of a film. This is not much used so I won't cover it.

The superclass of the animation API is the `Animator` class. The `ObjectAnimator` class can be used to modify attributes of an object.

You can also add an `AnimatorListener` class to your `Animator` class. This listener is called in the different phases of the animation. You can use this listener to perform actions before or after a certain animation, e.g. add or remove a `View` from a `ViewGroup`.

The `animate()` method on a `View` object returns an `ViewPropertyAnimator` object for the view. It provides a fluent API to typical animations which can be performed on views.

The following code shows an example.

```
myView.animate().translationX(400);
```

Course Manual for Android App Development

```
// if an animation is slow you can try to activate a hardware layer which
// uses a cache
// watch-out: this might not always result in a correct animation

myView.animate().translationX(400).withLayer();
```

You can also register action, which are executed before the start or after the end of the animation.

```
// StartAction
myView.animate().translationX(100).withStartAction(new Runnable(){
public void run(){
    viewer.setTranslationX(100-myView.getWidth());
// do something
    }
});

// EndAction
myView.animate().alpha(0).withEndAction(new Runnable(){
public void run(){
// rRemove the view from the parent layout
    parent.removeView(myView);
    }
});
```

13.2- Android Gradle:

Gradle is an open source build **automation system**. Gradle can automate the **building, testing, publishing, deployment** and more of software packages or other types of projects such as generated static websites, generated documentation and anything else

Gradle is an automated build toolkit that allows the way in which projects are built to be configured and managed through a set of build configuration files. This includes defining how a project is to be built, what dependencies need to be fulfilled for the project to build successfully and what the end result (or results) of the build process should be.

The strength of Gradle lies in the flexibility that it provides to the developer. The Gradle system is a self-contained, command-line based environment that can be integrated into other environments through the use of plug-ins. In the case of Android Studio, Gradle integration is provided through the appropriately named Android Studio Plug-in.

Course Manual for Android App Development

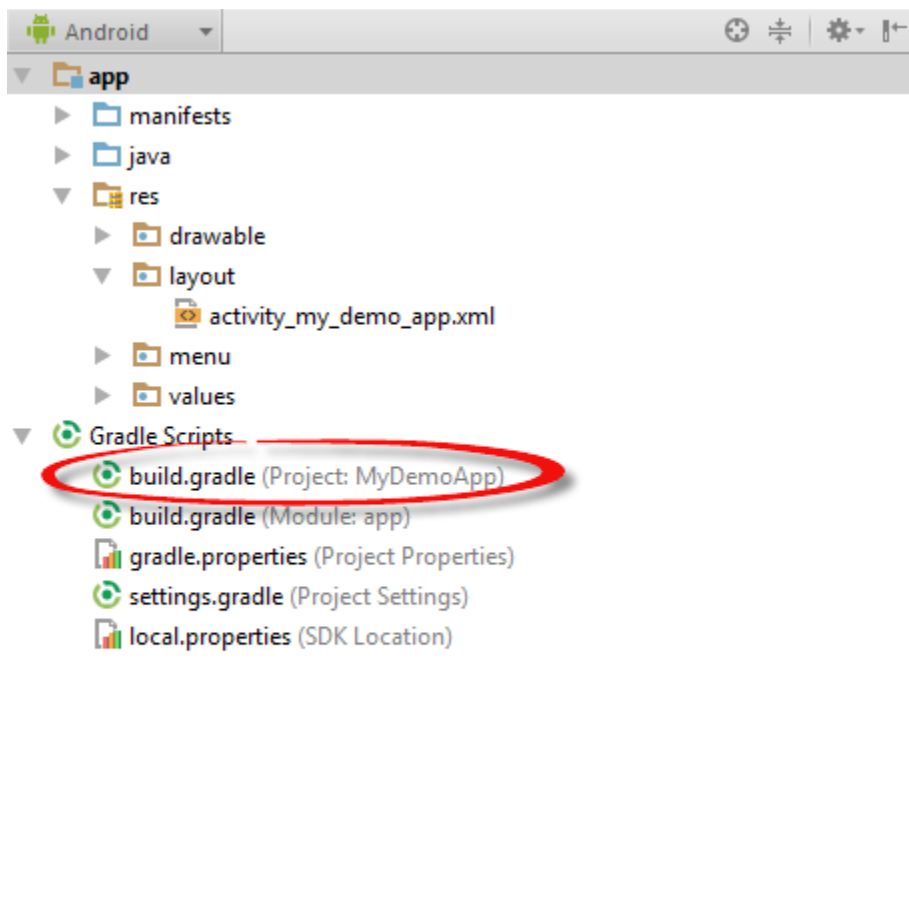
Although the Android Studio Plug-in allows Gradle tasks to be initiated and managed from within Android Studio, the Gradle command-line wrapper can still be used to build Android Studio based projects, including on systems on which Android Studio is not installed.

The configuration rules to build a project are declared in Gradle build files and scripts based on the Groovy programming language.

Gradle brings a number of powerful features to building Android application projects. Some of the key features are as follows:

Sensible Defaults, Dependencies, Build Variants, Manifest Entries, APK Signing, ProGuard Support.

A completed Android Studio project contains everything needed to build an Android application and consists of modules, libraries, manifest files and Gradle build files. Each project contains one top-level Gradle build file. This file is listed as build.gradle (Project: <project name>) and can be found in the project tool window as highlighted in below figure:



13.3- Espresso (Android Testing Support Library):

Espresso is a testing framework for Android to make it easy to write reliable user interface tests.

The framework also ensures that your activity is started before the tests run. It also let the test wait until all observed background activities have finished.

It is intended to test a single application but can also be used to test across applications. If used for testing outside your application, you can only perform black box testing, as you cannot access the classes outside of your application.

Espresso has basically three components:

- *ViewMatchers* - allows to find view in the current view hierarchy
- *ViewActions* - allows to perform actions on the views
- *ViewAssertions* - allows to assert state of a view

The case construct for Espresso tests is the following:

Base Espresso Test

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

- Finds the view
- Performs an action on the view
- Validates a assertioin

13.4- Configuration of the Gradle build file for Espresso:

To use Espresso for your tests, add the following dependency to the Gradle build file of your app.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])

    testCompile 'junit:junit:4.12'

    // Android runner and rules support
    androidTestCompile 'com.android.support.test:runner:0.5'
    androidTestCompile 'com.android.support.test:rules:0.5'

    // Espresso support
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
}
```

```
    })

    // add this for intent mocking support
    androidTestCompile 'com.android.support.test.espresso:espresso-intents:2.2.2'

    // add this for webview testing support
    androidTestCompile 'com.android.support.test.espresso:espresso-web:2.2.2'

}
```

Ensure that the `android.support.test.runner.AndroidJUnitRunner` is specified as value for the `testInstrumentationRunner` parameter in the build file of your app. Via the `packagingOptions` you may have to exclude `LICENSE.txt`, depending on the libraries you are using. The following listing is an example for that.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion '22.0.1'
    defaultConfig {
        applicationId "com.example.android.testing.espresso.BasicSample"
        minSdkVersion 10
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    packagingOptions {
        exclude 'LICENSE.txt'
    }
    lintOptions {
        abortOnError false
    }
}

dependencies {
    // as before.....
}
```

13.5- Adaptive Design:

Flexible layouts

By default, Android resizes your app layout to fit the current screen. To ensure your layout resizes well for even small variations in screen size, you need to implement your layout with flexibility in mind. The core principle you must follow is to avoid hard-coding the position and size of your UI components. Instead, allow view sizes to stretch and specify view positions relative to the parent view or other sibling views so your intended order and relative sizes remain the same as the layout grows.

Course Manual for Android App Development

Alternative layouts

A flexible layout is very important, but you should also design different layouts that optimize the user experience for the available space on different devices such as phones and tablets. So Android allows you to provide alternative layout files that the system applies at runtime based on the current device's screen size.

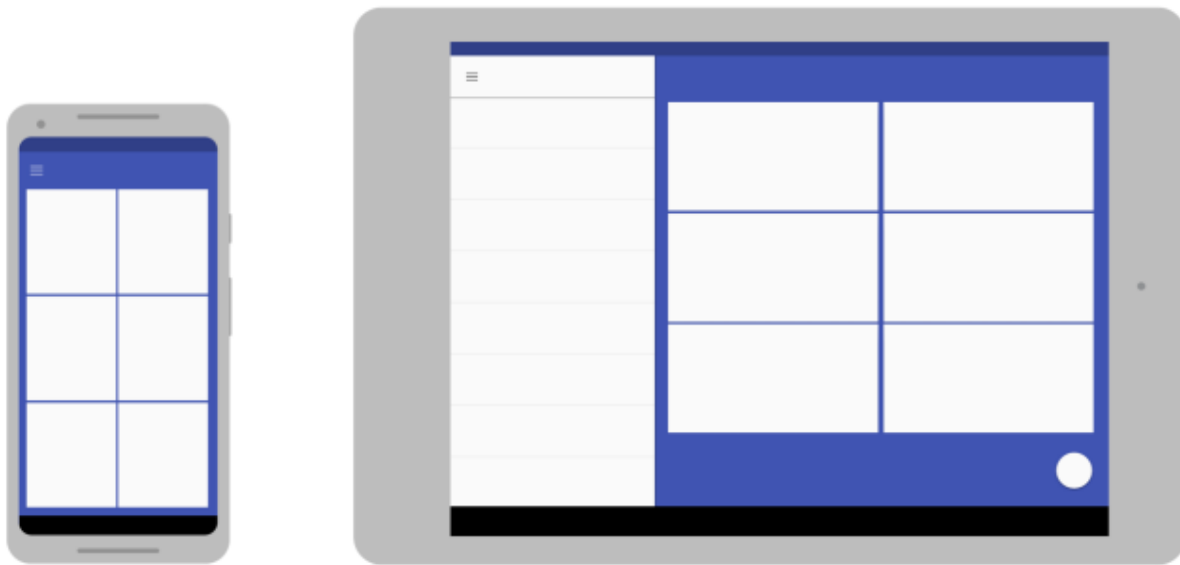


Figure 1. The same app uses a different layout for different screen sizes

Android devices come in all shapes and sizes, so your app's layout needs to be flexible. That is, instead of defining your layout with rigid dimensions that assume a certain screen size and aspect ratio, your layout should gracefully respond to different screen sizes and orientations.

By supporting as many screens as possible, your app can be made available to the greatest number of users with different devices, using a single APK. Additionally, making your app flexible for different screen sizes ensures that your app can handle window configuration changes on the device, such as when the user enables multi-window mode.

This page shows you how to support different screen sizes with the following techniques:

- Use view dimensions that allow the layout to resize
- Create alternative UI layouts according to the screen configuration
- Provide bitmaps that can stretch with the views

Course Manual for Android App Development

Avoid hard-coded layout sizes

To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap_content" and "match_parent" for the width and height of most view components, instead of hard-coded sizes.

"wrap_content" tells the view to set its size to whatever is necessary to fit the content within that view.

"match_parent" makes the view expand to as much as possible within the parent view.

For example:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/lorem_ipsum"/>
```

Use the smallest width qualifier

The "smallest width" screen size qualifier allows you to provide alternative layouts for screens that have a minimum width measured in density-independent pixels(dp or dip).

By describing the screen size as a measure of density-independent pixels, Android allows you to create layouts that are designed for very specific screen dimensions while avoiding any concerns you might have about different pixel densities.

For example, you can create a layout named main_activity that's optimized for handsets and tablets by creating different versions of the file in directories as follows:

```
res/layout/main_activity.xml      # For handsets (smaller than 600dp available width)
```

```
res/layout-sw600dp/main_activity.xml # For 7" tablets (600dp wide and bigger)
```

The smallest width qualifier specifies the smallest of the screen's two sides, regardless of the device's current orientation, so it's a simple way to specify the overall screen size available for your layout.

Here's how other smallest width values correspond to typical screen sizes:

- 320dp: a typical phone screen (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, etc).
- 480dp: a large phone screen ~5" (480x800 mdpi).
- 600dp: a 7" tablet (600x1024 mdpi).
- 720dp: a 10" tablet (720x1280 mdpi, 800x1280 mdpi, etc).
-

- res/layout/my_layout.xml // layout for normal screen size ("default")
- res/layout-small/my_layout.xml // layout for small screen size
- res/layout-large/my_layout.xml // layout for large screen size
- res/layout-xlarge/my_layout.xml // layout for extra large screen size
- res/layout-xlarge-land/my_layout.xml // layout for extra large in landscape orientation

Course Manual for Android App Development

-
- res/drawable-mdpi/my_icon.png // bitmap for medium density
- res/drawable-hdpi/my_icon.png // bitmap for high density
- res/drawable-xhdpi/my_icon.png // bitmap for extra high density

13.6- Creating an Android Wear Interface for Weather App:

- **Scope of implementation & case study**

1. Flexible layouts

PRACTICE:

Alternative layouts->Avoid hard-coded layout sizes->smallest width qualifier

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

http://www.techotopia.com/index.php/An_Overview_of_Gradle_in_Android_Studio

<https://developers.google.com/training/courses/android-advanced>

Course Manual for Android App Development

MODULE 14: Android Application Developments

CHAPTER OVERVIEW:

Whether you're new to programming or an experienced developer, we have a range of courses to teach you Android app development, from your first app to advanced topics such as localization, media, advanced graphics, and performance.

14.1- Android App Development on Windows and Linux

14.2- Android App Development on Android Market

OBJECTIVS:

- Android App Development on Windows and Linux.
- Publishing is the general process that makes your Android applications available to users.

LEARNING OUTCOME:

- Preparing your app for release
- Releasing your app to users
- Releasing through an app marketplace
- Releasing your apps on Google Play

CORE DISCUSSION:

- **Historical approach :**

Android software development is the process by which new applications are created for devices running the Android operating system. Officially, apps can be written using Java, C++ or Kotlin using the Android software development kit (SDK). Third party tools, development environments and language support have also continued to evolve and expand since the initial SDK was released in 2008.

- **Fundamental technology**

Android Studio : Android Studio is the officialintegrated development environment (IDE) for Google's Androidoperating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systemsIt is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

Course Manual for Android App Development

Eclipse IDE: Eclipse IDE (integrated development environment). Included in the Android SDK download, the Eclipse IDE provides the “hands-on” controls you need for writing your app using Java, the Android SDK and the Android ADT. Android ADT (Android Development). Apr 25, 2013

- **Scope of implementation & case study**

1. Android App Development on Android Market
2. Android App Development on Windows and Linux

14.1- Android App Development on Windows and Linux:

There are so many technologies, tools, platforms and languages available to contemporary software developers and programmers, that it's certainly impossible to find someone in the industry who at the same time: works on a single platform, *speaks* a single *programming dialect* and is worth enough their salt to be paid for their bacon, coffee and beers. The world is harsh; the race is tough. Starting with Android doesn't mean to abandon any previous source of income, joy, enlightenment or whatever.

System Requirements		
Windows	Mac	Linux
<ul style="list-style-type: none">• Microsoft® Windows® 7/8/10 (32- or 64-bit)• 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator• 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)• 1280 x 800 minimum screen resolution	<ul style="list-style-type: none">• Mac® OS X® 10.10 (Yosemite) or higher, up to 10.13 (macOS High Sierra)• 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator• 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)• 1280 x 800 minimum screen resolution	<ul style="list-style-type: none">• GNOME or KDE desktop<ul style="list-style-type: none">Tested on Ubuntu® 14.04 LTS, Trusty Tahr (64-bit distribution capable of running 32-bit applications)• 64-bit distribution capable of running 32-bit applications• GNU C Library (glibc) 2.19 or later• 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator• 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)• 1280 x 800 minimum screen resolution

Start using Android Studio today

Android Studio includes all the tools you need to build apps for Android.

DOWNLOAD ANDROID STUDIO
3.1 FOR WINDOWS (758 MB)

VERSION: 3.1.0.16

RELEASE DATE: MARCH 26, 2018

Course Manual for Android App Development

Select a different platform

Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	android-studio-ide-173.4670197-windows.exe	758 MB (794,898,984 bytes)	d787baa8d9282cbbc671b82b127c3e6bfe9f73a5674ceed17a9e68f926aac526
	android-studio-ide-173.4670197-windows.zip No installer	854 MB (895,661,491 bytes)	8df4490f161bc59ff7e88fa84dbb59a7a9e3dd5bfb75339a2a8c646a26b890a9
Windows (32-bit)	android-studio-ide-173.4670197-windows32.zip No installer	853 MB (895,131,994 bytes)	79fd3f284142887b09401f57793ea6e30595625d94a2a12c845e4a0735339143
Mac	android-studio-ide-173.4670197-mac.dmg	847 MB (889,166,178 bytes)	920f1505fd6b8c23d8c56c1ebc72bbea2c99c3c1badc9747372a1fc9d7add04c
Linux	android-studio-ide-173.4670197-linux.zip	852 MB (894,043,298 bytes)	de2587bf5471695c8a411f979e6247c0d552ce14f5ca9de82ddeb3dc946f14c4

Get just the command line tools

If you do not need Android Studio, you can download the basic Android command line tools below. You can use the included sdkmanager to download other SDK packages.

These tools are included in Android Studio.

14.2- Android App Development on Android Market:

Google's developer program for its Google Play app store, previously called the Android App Market, has only a few simple requirements for you to register and start testing and selling your apps. Requirements don't include any proof of prior coding experience or existing apps, because Google offers some app testing and does not have strict governance for app approval.

Publishing is the general process that makes your Android applications available to users. When you publish an Android application you perform two main tasks:

- You prepare the application for release.
During the preparation step you build a release version of your application, which users can download and install on their Android-powered devices.
- You release the application to users.

During the release step you publicize, sell, and distribute the release version of your application to users.

Course Manual for Android App Development

Usually, you release your application through an application marketplace, such as Google Play. However, you can also release applications by sending them directly to users or by letting users download them from your own website.

The publishing process is typically performed after you finish testing your application in a debug environment. Also, as a best practice, your application should meet all of your release criteria for functionality, performance, and stability before you begin the publishing process.

Preparing your app for release

Preparing your application for release is a multi-step process that involves the following tasks:

- Configuring your application for release.

At a minimum you need to remove `Log` calls and remove the `android:debuggable` attribute from your manifest file. You should also provide values for the `android:versionCode` and `android:versionName` attributes, which are located in the `<manifest>` element. You may also have to configure several other settings to meet Google Play requirements or accommodate whatever method you're using to release your application.

If you are using Gradle build files, you can use the `release` build type to set your build settings for the published version of your app.

- Building and signing a release version of your application.

You can use the Gradle build files with the `release` build type to build and sign a release version of your application. See [Building and Running from Android Studio](#).

- Testing the release version of your application.

Before you distribute your application, you should thoroughly test the release version on at least one target handset device and one target tablet device.

- Updating application resources for release.

You need to be sure that all application resources such as multimedia files and graphics are updated and included with your application or staged on the proper production servers.

- Preparing remote servers and services that your application depends on.

If your application depends on external servers or services, you need to be sure they are secure and production ready.

You may have to perform several other tasks as part of the preparation process. For example, you will need to get a private key for signing your application. You will also need to create an icon for your application, and you may want to prepare an End User License Agreement (EULA) to protect your person, organization, and intellectual property.

When you are finished preparing your application for release you will have a signed .apk file that you can distribute to users.

To learn how to prepare your application for release, see [Preparing for Release](#) in the Dev Guide. This topic provides step-by-step instructions for configuring and building a release version of your application.

Releasing your app to users

You can release your Android applications several ways. Usually, you release applications through an application marketplace such as Google Play, but you can also release applications on your own website or by sending an application directly to a user.

Releasing through an app marketplace

If you want to distribute your apps to the broadest possible audience, releasing through an app marketplace such as Google Play is ideal.

Google Play is the premier marketplace for Android apps and is particularly useful if you want to distribute your applications to a large global audience. However, you can distribute your apps through any app marketplace you want or you can use multiple marketplaces.

Releasing your apps on Google Play

Google Play is a robust publishing platform that helps you publicize, sell, and distribute your Android applications to users around the world. When you release your applications through Google Play you have access to a suite of developer tools that let you analyze your sales, identify market trends, and control who your applications are being distributed to. You also have access to several revenue-enhancing features such as in-app billing and application licensing. The rich array of tools and features, coupled with numerous end-user community features, makes Google Play the premier marketplace for selling and buying Google Play applications.

Releasing your application on Google Play is a simple process that involves three basic steps:

- Preparing promotional materials.

To fully leverage the marketing and publicity capabilities of Google Play, you need to create promotional materials for your application, such as screenshots, videos, graphics, and promotional text.

- Configuring options and uploading assets.

Google Play lets you target your application to a worldwide pool of users and devices. By configuring various Google Play settings, you can choose the countries you want to reach, the listing languages you want to use, and the price you want to charge in each country. You can also configure listing details such as the application type, category, and content rating. When you are done configuring options you can upload your promotional materials and your application as a draft (unpublished) application.

- Publishing the release version of your application.

If you are satisfied that your publishing settings are correctly configured and your uploaded application is ready to be released to the public, you can simply click **Publish** in the Play Console and within minutes your application will be live and available for download around the world.

You've just written a great Android app and you're ready to make some money from it. The most common route is probably to pay \$25 to get it listed on Google Play. However, there are many alternatives to Google Play, each with their own audience that can drive more downloads.

Course Manual for Android App Development

Some of these app stores are catered to a smaller audience while others are more localized. Most of them don't charge you for listing your app on their store. It can therefore pay off to publish your app in several app stores.

Amazon Appstore, SlideME, 1Mobile Market, Samsung Galaxy Apps, Mobile9

SUGGESTED READING LIST: (Books & Links)

Android Application Development for Dummies - Book by Donn Felker

Professional Android 2 Application Development - Book by Reto Meier

The Busy Coder's Guide to Android Development - Book by Mark Murphy

<https://commonsware.com/Android/Android-4.7-CC.pdf>

REFARANCE LINK:

<https://developers.google.com/training/android/>

Course Manual for Android App Development

MODULE 15: Project Work

2 Completed projects have to be completed and delivered for every student (First one is s a group project and the last one is the individual project) (26 hrs.)

In this stage of training we will make few real life project that will give the student a hands on experience to work with real life requirement of projects following every steps of software development life cycle.

Projects in the phase of training will contain mail portion of this training and their implementation. Projects will at least cover the following training modules and implementation of them:

- MVC
- Auto Layout
- Concept of JAVA
- Storyboards (Scene, segue, Navigation Controller)
- TableView & CollectionView
- Local Storage (SQLite, Core Data)
- Networking (XML. JSON Parsing)
- Animation, Location Framework (MapKit)