

# MANIPAL INSTITUTE OF TECHNOLOGY

Manipal – 576 104

## DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY



### CERTIFICATE

This is to certify that Ms./Mr. ....  
Reg.No. .... Section: .... Roll No: .... has  
satisfactorily completed the lab exercises prescribed for **Mobile Application  
Development Lab [ICT 3268]** of Third Year B. Tech. CCE Degree at MIT, Manipal, in  
the academic year 20\_\_- 20\_\_.

Date: .....

Signature of the faculty



## **CONTENTS**

<b>LAB NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>	<b>SIGNATURE</b>	<b>REMARKS</b>
	COURSE OBJECTIVES, OUTCOMES AND EVALUATION PLAN	I		
	INSTRUCTIONS TO THE STUDENTS	II		
1	INTRODUCTION TO ANDROID	4		
2	ACTIVITY, LAYOUTS	20		
3	ACTIVITY, LAYOUTS AND INTENT FILTERS CONTINUED	47		
4	INPUT CONTROLS-BUTTONS, CHECK BOX, RADIO BUTTONS AD TOGGLES	64		
5	INPUT CONTROLS-SPINNER, PICKERS	84		
6	INTRODUCTION TO MENU	98		
7	CONTEXT AND POP UP MENU	112		
8	SQLITE AND SHARED PREFERENCES	126		
9	SECURITY AND PERMISSIONS	140		
10	SERVICES, BROADCAST RECEIVER	152		
11	CAMERA, BLUETOOTH AND WI-FI	164		
12	PROJECT IMPLENTATION & TESTING	187		
	REFERENCES	190		



### Course Objectives

- To design and develop mobile applications using android, Ios and windows.
- To familiarize with mobile UI design.
- To learn Database Connectivity to mobile applications

### Course Outcomes

At the end of this course, students will be able to

- Design and develop mobile UI with good aesthetic sense of designing and latest technical know-how's.
- Have a good understanding of mobile UI controls required for developing mobile application.
- Learn how to develop fully functional real world mobile application

### Evaluation plan

<b>Split up of 60 marks for Regular Lab Evaluation</b>
Six regular evaluations will be carried out in alternate weeks. Each evaluation is for 10 marks and following is the split up:  Record : 4 Marks  Evaluation: 4 Marks  Execution: 2 Marks  Total = 10 Marks  Total Internal Marks: $6 * 10 = 60$ Marks
<b>End Semester Lab evaluation: 20 marks (Duration 2 hrs)</b>
Program write up: 10 Marks Program execution: 10 Marks Total: $10+10 = 20$ Marks
<b>Mini project evaluation: 20 marks</b>

## INSTRUCTIONS TO THE STUDENTS

### Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

### In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

### General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
  - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
  - Comments should be used to give the statement of the problem.
  - Statements within the program should be properly indented.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- In case a student misses a lab, he/ she must ensure that the experiment is completed before the next evaluation with the permission of the faculty concerned.
- Students missing out lab on genuine reasons like conference, sports or activities assigned by the Department or Institute will have to take **prior permission** from the HOD to attend **additional lab**(with other batch) and complete it **before** the student goes on leave. The student could be awarded marks for the write up for that day provided he submits it during the **immediate** next lab.

- Students who fall sick should get permission from the HOD for evaluating the lab records. However attendance will not be given for that lab.
- Students will be evaluated only by the faculty with whom they are registered even though they carry out additional experiments in other batch.
- Presence of the student during the lab end semester exams is mandatory even if the student assumes he has scored enough to pass the examination
- Minimum attendance of 75% is mandatory to write the final exam.
- If the student loses his book, he/she will have to rewrite all the lab details in the lab record.
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

### **THE STUDENTS SHOULD NOT**

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

**LAB NO: 1**

**Date:**

## **BASICS OF ANDROID MOBILE APPLICATION DEVELOPMENT TOOL**

### **Objectives**

- To familiarize with mobile application development tool.
- To gain knowledge about how to develop simple mobile application using android features.

### **What is Android?**



Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit SDK was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 **Jelly Bean**. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.



The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

### Android Applications

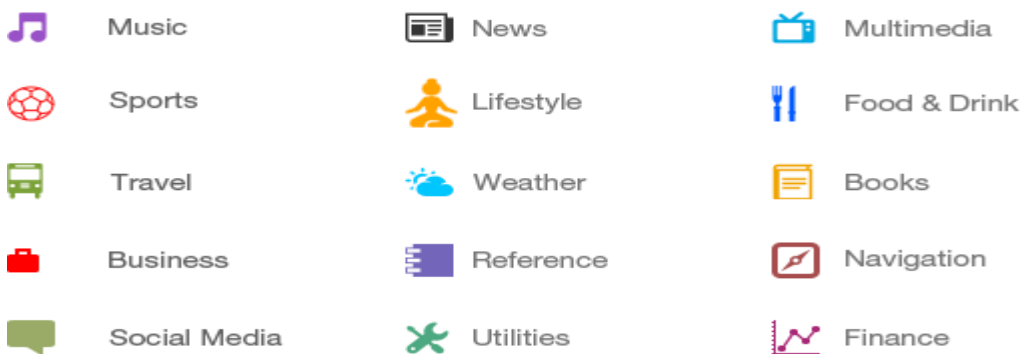
Android applications are usually developed in the Java language using the Android Software Development Kit.

Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play**, **SlideME**, **Opera Mobile Store**, **Mobango**, **F-droid** and the **Amazon Appstore**.

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide.

### Categories of Android applications

There are many android applications in the market. The top categories are –



### History of Android

The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop and Marshmallow. Let's understand the android history in a sequence.



### What is API level?

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

Table 1: Version of Android Platform

Platform Version	API Level	VERSION_CODE
Android 6.0	23	MARSHMALLOW
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT

Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ÉCLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE

Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

## Android- Environment Setup

Application developer can using following operating system to run Android application

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Android Studio

Here last two components are optional and if you are working on Windows machine then these components make your life easy while doing Java based application development. So let us have a look how to proceed to set required environment.

## Set-up Java Development Kit (JDK)

You can download the latest version of Java JDK from Oracle's Java site – Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA\_HOME environment variables to refer to the directory that contains **java** and **javac**, typically java\_install\_dir/bin and java\_install\_dir respectively.

If you are running Windows and installed the JDK in C:\jdk1.8.0\_102, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%  
set JAVA_HOME=C:\jdk1.8.0_102
```

Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.

On Linux, if the SDK is installed in /usr/local/jdk1.8.0\_102 and you use the C shell, you would put the following code into your **.cshrc** file.

```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH  
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

Alternatively, if you use Android studio, then it will know automatically where you have installed your Java.

### **Android Architecture**

Android architecture or Android software stack is categorized into five parts as shown in the Figure 1:

1. linux kernel
2. native libraries (middleware),
3. Android Runtime
4. Application Framework
5. Applications

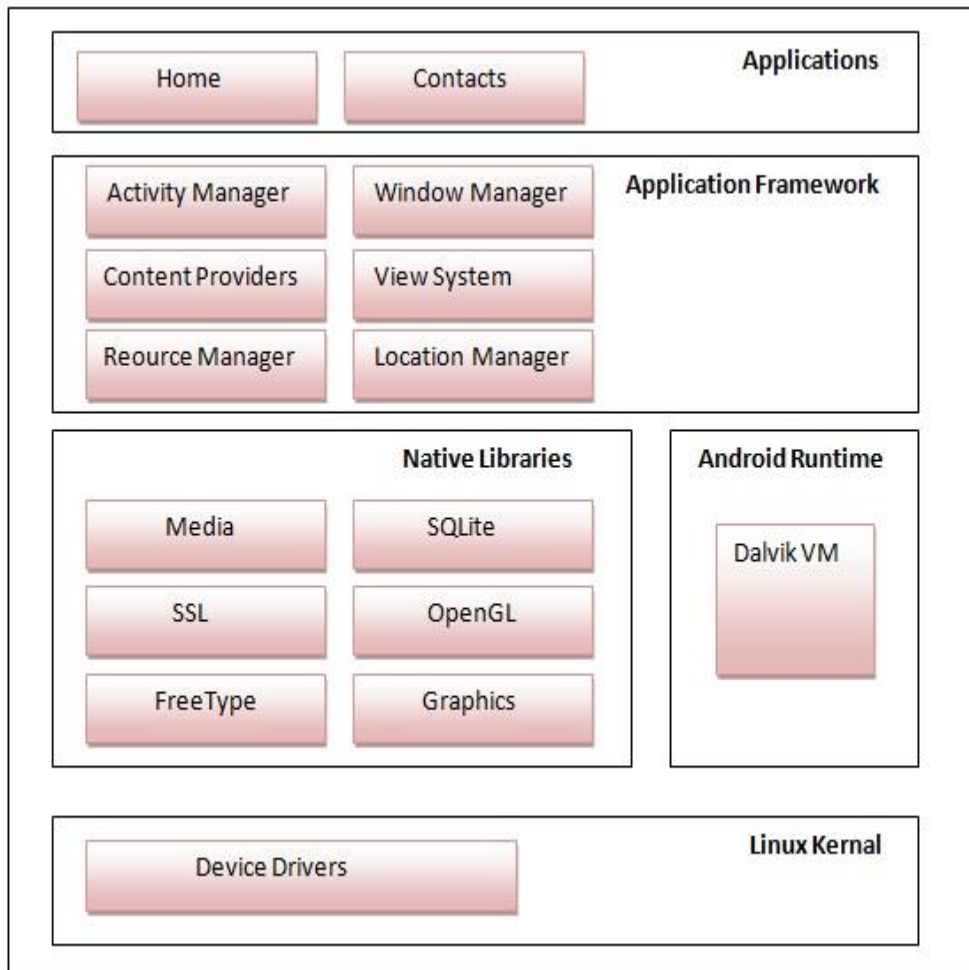


Figure 1: Android Architecture

### 1) Linux kernel

It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.

### 2) Native Libraries

On the top of linux kernel, there are Native libraries such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc. The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

### 3) Android Runtime

In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

### 4) Android Framework

On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

### 5) Applications

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using Linux

---

## Steps to create an android project

This lesson shows how to create a new Android project with Android Studio and describes some of the files in the project.

1. In Android Studio, create a new project:
  - If you don't have a project opened, in the **Welcome to Android Studio** window, click **Start a new Android Studio project**.
  - If you have a project opened, select **File > New Project**.
2. In the **New Project** screen, enter the following values:

- **Application Name:** "My First App"
- **Company Domain:** "example.com"

Android Studio fills in the package name and project location for you, but you can edit these if you'd like.

3. Click **Next**.
4. In the **Target Android Devices** screen, keep the default values and click **Next**.

The **Minimum Required SDK** is the earliest version of Android that your app supports, which is indicated by the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the core feature set, enable that feature only when running on the versions that support it (see Supporting Different Platform Versions).

5. In the **Add an Activity to Mobile** screen, select **Empty Activity** and click **Next**.
6. In the **Customize the Activity** screen, keep the default values and click **Finish**.

After some processing, Android Studio opens and displays a "Hello World" app with default files. You will add functionality to some of these files in the following lessons.

Now take a moment to review the most important files. First, be sure that the **Project** window is open (select **View** > **Tool Windows** > **Project**) and the **Android** view is selected from the drop-down list at the top. You can then see the following files:

**app > java > com.example.myfirstapp > MainActivity.java**

This file appears in Android Studio after the New Project wizard finishes. It contains the class definition for the activity you created earlier. When you build and run the app, the Activity starts and loads the layout file that says "Hello World!"

**app > res > layout > activity\_main.xml**

This XML file defines the layout of the activity. It contains a TextView element with the text "Hello world!".

**app > manifests > AndroidManifest.xml**



The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll revisit this file as you follow these lessons and add more components to your app.

### Gradle Scripts > build.gradle

Android Studio uses Gradle to compile and build your app. There is a build.gradle file for each module of your project, as well as a build.gradle file for the entire project. Usually, you're only interested in the build.gradle file for the module, in this case the app or application module. For more information about this file, see Building Your Project with Gradle.


### Run on a Real Device

Set up your device as follows:

1. Connect your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device. For help installing drivers, see the OEM USB Drivers document.
2. Enable **USB debugging** on your device by going to **Settings > Developer options**.

**Note:** On Android 4.2 and newer, **Developer options** is hidden by default. To make it available, go to **Settings > About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.

Run the app from Android Studio as follows:


1. In Android Studio, select your project and click **Run**  from the toolbar.
2. In the **Select Deployment Target** window, select your device, and click **OK**.

Android Studio installs the app on your connected device and starts it.

## Run on an Emulator

Before you run your app on an emulator, you need to create an Android Virtual Device (AVD) definition. An AVD definition defines the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.

Create an AVD Definition as follows:


1. Launch the Android Virtual Device Manager by selecting **Tools > Android > AVD Manager**, or by clicking the AVD Manager icon  in the toolbar.
2. In the **Your Virtual Devices** screen, click **Create Virtual Device**.
3. In the **Select Hardware** screen, select a phone device, such as Nexus 6, and then click **Next**.
4. In the **System Image** screen, choose the desired system image for the AVD and click **Next**.

If you don't have a particular system image installed, you can get it by clicking the **download** link.

5. Verify the configuration settings (for your first AVD, leave all the settings as they are), and then click **Finish**.

For more information about using AVDs, see [Create and Manage Virtual Devices](#).

Run the app from Android Studio as follows:

1. In **Android Studio**, select your project and click **Run**  from the toolbar.
2. In the **Select Deployment Target** window, select your emulator and click **OK**.

It can take a few minutes for the emulator to start. You may have to unlock the screen. When you do, *My First App* appears on the emulator screen.



### **Lab exercises**

Develop an Android application for the following program

1. Create an Android application to show the demo of displaying text with justifying elements.
2. Find the “hello word” text in the XML document and modify the text.

[OBSERVATION SPACE – LAB 1]

[OBSERVATION SPACE – LAB 1]

[OBSERVATION SPACE – LAB 1]

[OBSERVATION SPACE – LAB 1]

**LAB NO: 2**

**Date:**

## **INTRODUCTION TO ACTIVITY AND LAYOUTS IN ANDROID**

### **Objectives**

- To apply the concepts of layouts to enrich the user interface.
- To understand different activity and fragments to use in android application.

### **Introduction to Activities**

The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a `main()` method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

### **The concept of activities**

The Activity class serves as the entry point for an app's interaction with the user, providing the window in which the app draws its UI. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. You implement an activity as a subclass of the Activity class. Generally, one activity implements one screen in an app. For instance, one of an app's activities may implement the *Preferences* screen, while another activity implements the *Compose Email* screen.

Most apps contain multiple screens, which means they comprise multiple activities. Typically, one activity in an app is specified as the *main activity*, which is the first screen to appear when the user launches the app. Each activity can then start another activity in order to perform different actions. For example, the main activity in a simple e-mail app



may provide the screen that shows an e-mail inbox. From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

Although activities work together to form a cohesive user experience in an app, each activity is only loosely bound to the other activities; there are usually minimal dependencies among the activities in an app. In fact, activities often start up activities belonging to other apps. For example, a browser app might launch the Share activity of a social-media app.

To use activities in your app, you must register information about them in the app's manifest, and you must manage activity lifecycles appropriately. The rest of this document introduces these subjects.

### **Configuring the manifest**

For your app to be able to use activities, you must declare the activities, and certain of their attributes, in the manifest.

### **Declare activities**

To declare your activity, open your manifest file and add an `<activity>` element as a child of the `<application>` element. For example:

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```

The only required attribute for this element is `android:name`, which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme. For more information about these and other attributes, see the `<activity>` element reference documentation.

**Note:** After you publish your app, you should not change activity names. If you do, you might break some functionality, such as app shortcuts. For more information on changes to avoid after publishing, see [Things That Cannot Change](#).

### Declare intent filters

Intent filters are a very powerful feature of the Android platform. They provide the ability to launch an activity not only based not on an *explicit* request, but also an *implicit* one. For example, an explicit request might tell the system to “Start the Send Email activity in the Gmail app”. By contrast, an implicit request tells the system to “Start a Send Email screen in any activity that can do the job.” When the system UI asks a user which app to use in performing a task, that’s an intent filter at work.

You can take advantage of this feature by declaring an `<intent-filter>` attribute in the `<activity>` element. The definition of this element includes an `<action>` element and, optionally, a `<category>` element and/or a `<data>` element. These elements combine to specify the type of intent to which your activity can respond. For example, the following code snippet shows how to configure an activity that sends text data, and receives requests from other activities to do so:

```
<activity android:name=".ExampleActivity"
  android:icon="@drawable/app_icon">
  <intent-filter>
    <action android:name="android.intent.action.SEND" />
```

```

    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
</intent-filter>
<activity>

```

In this example, the `<action>` element specifies that this activity sends data. Declaring the `<category>` element as `DEFAULT` enables the activity to receive launch requests. The `<data>` element specifies the type of data that this activity can send. The following code snippet shows how to call the activity described above:

```

// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
// Start the activity
startActivity(sendIntent);

```

If you intend for your app to be self-contained and not allow other apps to activate its activities, you don't need any other intent filters. Activities that you don't want to make available to other applications should have no intent filters, and you can start them yourself using explicit intents. For more information about how your activities can respond to intents, see [Intents and Intent Filters](#).

## Declare permissions

You can use the manifest's `<activity>` tag to control which apps can start a particular activity. A parent activity cannot launch a child activity unless both activities have the the same permissions in their manifest. If you declare a `<uses-permission>` element for a particular activity, the calling activity must have a matching `<uses-permission>` element.

For example, if your app wants to use a hypothetical app named SocialApp to share a post on social media, SocialApp itself must define the permission that an app calling it must have:

```
<manifest>
<activity android:name="...."
    android:permission="com.google.socialapp.permission.SHARE_POST"
/>
```

Then, to be allowed to call SocialApp, your app must match the permission set in SocialApp's manifest:

```
<manifest>
    <uses-permission android:name="com.google.socialapp.permission.SHARE_POST" />
</manifest>
```

### Managing the activity lifecycle

Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states. The following sections introduce these callbacks.

#### onCreate()

You must implement this callback, which fires when the system creates your activity. Your implementation should initialize the essential components of your activity: For example, your app should create views and bind data to lists here. Most importantly, this

is where you must call `setContentView()` to define the layout for the activity's user interface. When `onCreate()` finishes, the next callback is always `onStart()`.

### **onStart()**

As `onCreate()` exits, the activity enters the Started state, and the activity becomes visible to the user. This callback contains what amounts to the activity's final preparations for coming to the foreground and becoming interactive.

### **onResume()**

The system invokes this callback just before the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, and captures all user input. Most of an app's core functionality is implemented in the `onResume()` method.

The `onPause()` callback always follows `onResume()`.

### **onPause()**

The system calls `onPause()` when the activity loses focus and enters a Paused state. This state occurs when, for example, the user taps the Back or Overlay button. When the system calls `onPause()` for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.

An activity in the Paused state may continue to update the UI if the user is expecting the UI to update. Examples of such an activity include one showing a navigation map screen or a media player playing. Even if such activities lose focus, the user expects their UI to continue updating.

You should **not** use `onPause()` to save application or user data, make network calls, or execute database transactions. For information about saving data, see Saving and restoring activity state.

Once `onPause()` finishes executing, the next callback is either `onStop()` or `onRestart()`, depending on what happens after the activity enters the Paused state.

### **`onStop()`**

The system calls `onStop()` when the activity is no longer visible to the user. This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is entering a Resumed state and is covering the stopped activity. In all of these cases, the stopped activity is no longer visible at all.

The next callback that the system calls is either `onRestart()`, if the activity is coming back to interact with the user, or by `onDestroy()` if this activity is completely terminating.

### **`onRestart()`**

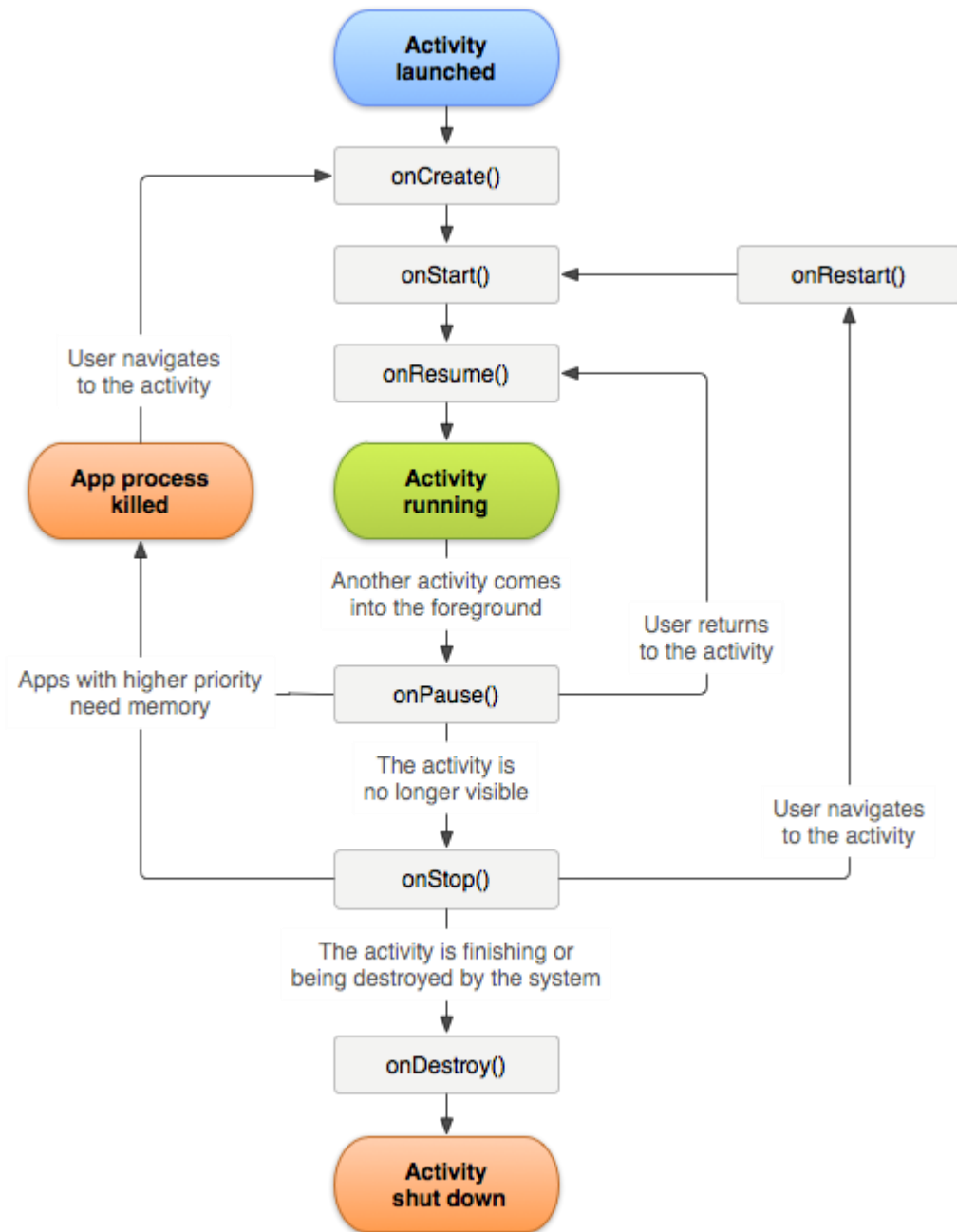
The system invokes this callback when an activity in the Stopped state is about to restart. `onRestart()` restores the state of the activity from the time that it was stopped.

This callback is always followed by `onStart()`.

### **`onDestroy()`**

The system invokes this callback before an activity is destroyed.

This callback is the final one that the activity receives. `onDestroy()` is usually implemented to ensure that all of an activity's resources are released when the activity, or the process containing it, is destroyed.

**Activity-lifecycle concepts****Figure 3.** A simplified illustration of the activity lifecycle.

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six

callbacks: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, and `onDestroy()`. The system invokes each of these callbacks as an activity enters a new state. Figure 3 presents a visual representation of this paradigm. As the user begins to leave the activity, the system calls methods to dismantle the activity. In some cases, this dismantlement is only partial; the activity still resides in memory (such as when the user switches to another app), and can still come back to the foreground. If the user returns to that activity, the activity resumes from where the user left off. The system's likelihood of killing a given process—along with the activities in it—depends on the state of the activity at the time. Activity state and ejection from memory provides more information on the relationship between state and vulnerability to ejection.

### **UI Overview**

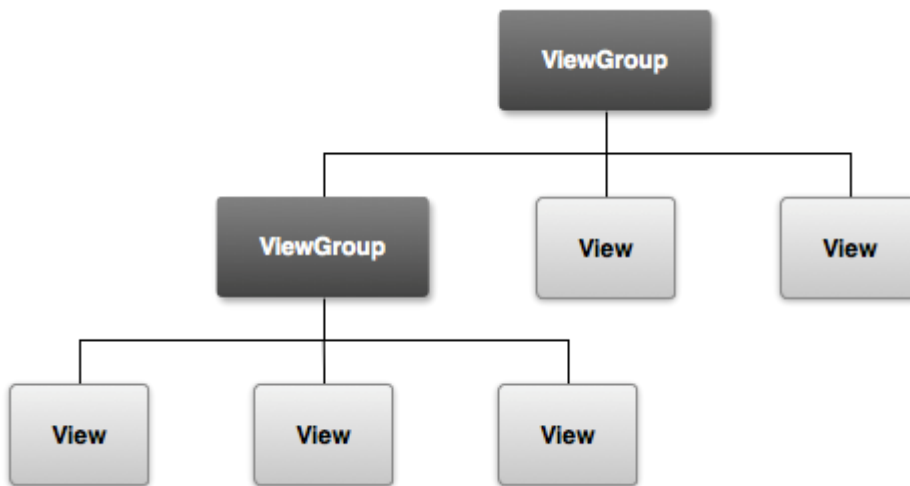
All user interface elements in an Android app are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.

Android provides a collection of both View and ViewGroup subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).



## User Interface Layout

The user interface for each component of your app is defined using a hierarchy of View and ViewGroup objects, as shown in figure 4. Each view group is an invisible container that organizes child views, while the child views may be input controls or other widgets that draw some part of the UI. This hierarchy tree can be as simple or complex as you need it to be (but simplicity is best for performance).



**Figure 4.** Illustration of a view hierarchy, which defines a UI layout.

To declare your layout, you can instantiate View objects in code and start building a tree, but the easiest and most effective way to define your layout is with an XML file. XML offers a human-readable structure for the layout, similar to HTML.

The name of an XML element for a view is respective to the Android class it represents. So a `<TextView>` element creates a TextView widget in your UI, and a `<LinearLayout>` element creates a LinearLayout view group.

For example, a simple vertical layout with a text view and a button looks like this:

```

<?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"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button" />
</LinearLayout>

```

When you load a layout resource in your app, Android initializes each node of the layout into a runtime object you can use to define additional behaviors, query the object state, or modify the layout.

## Layouts

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.

The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your

application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

- You should also try the Hierarchy Viewer tool, for debugging layouts — it reveals layout property values, draws wireframes with padding/margin indicators, and full rendered views while you debug on the emulator or device.
- The layoutopt tool lets you quickly analyze your layouts and hierarchies for inefficiencies or other problems.

The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior. Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile. For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems. As such, this document focuses on teaching you how to declare your layout in XML. If you're interested in instantiating View objects at runtime, refer to the ViewGroup and View class references.

In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods. In fact, the correspondence is often so direct that you can guess what XML attribute corresponds to a class method, or guess what class corresponds to a given XML element. However, note that not all vocabulary is identical. In some cases, there are slight naming differences. For example, the EditText element has a text attribute that corresponds to EditText.setText().

## Write the XML

Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

Each layout file must contain exactly one root element, which must be a View or ViewGroup object. Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout. For example, here's an XML layout that uses a vertical LinearLayout to hold a TextView and a Button:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

After you've declared your layout in XML, save the file with the .xml extension, in your Android project's res/layout/ directory, so it will properly compile. More information about the syntax for a layout XML file is available in the Layout Resources document.

## Load the XML Resource

When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your `Activity.onCreate()` callback implementation. Do so by calling `setContentView()`, passing it the reference to your layout resource in the form of: `R.layout.layout_file_name`. For example, if your XML layout is saved as `main_layout.xml`, you would load it for your Activity like so:

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main_layout);  
}
```

The `onCreate()` callback method in your Activity is called by the Android framework when your Activity is launched (see the discussion about lifecycles, in the Activities document).

## Attributes

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, `TextView` supports the `textSize` attribute), but these attributes are also inherited by any View objects that may extend this class. Some are common to all View objects, because they are inherited from the root View class (like the `id` attribute). And, other attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.

## ID

Any View object may have an integer ID associated with it, to uniquely identify the View within the tree. When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute. This is an XML attribute common to all View objects (defined by the View class) and you will use it very often. The syntax for an ID, inside an XML tag is:

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

The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource. The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file). There are a number of other ID resources that are offered by the Android framework. When referencing an Android resource ID, you do not need the plus-symbol, but must add the android package namespace, like so:

```
android:id="@android:id/empty"
```

With the android package namespace in place, we're now referencing an ID from the android.R resources class, rather than the local resources class. In order to create views and reference them from the application, a common pattern is to:

1. Define a view/widget in the layout file and assign it a unique ID:

```
<Button android:id="@+id/my_button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/my_button_text"/>
```

2. Then create an instance of the view object and capture it from the layout (typically in the onCreate() method):

```
Button myButton = (Button) findViewById(R.id.my_button);
```

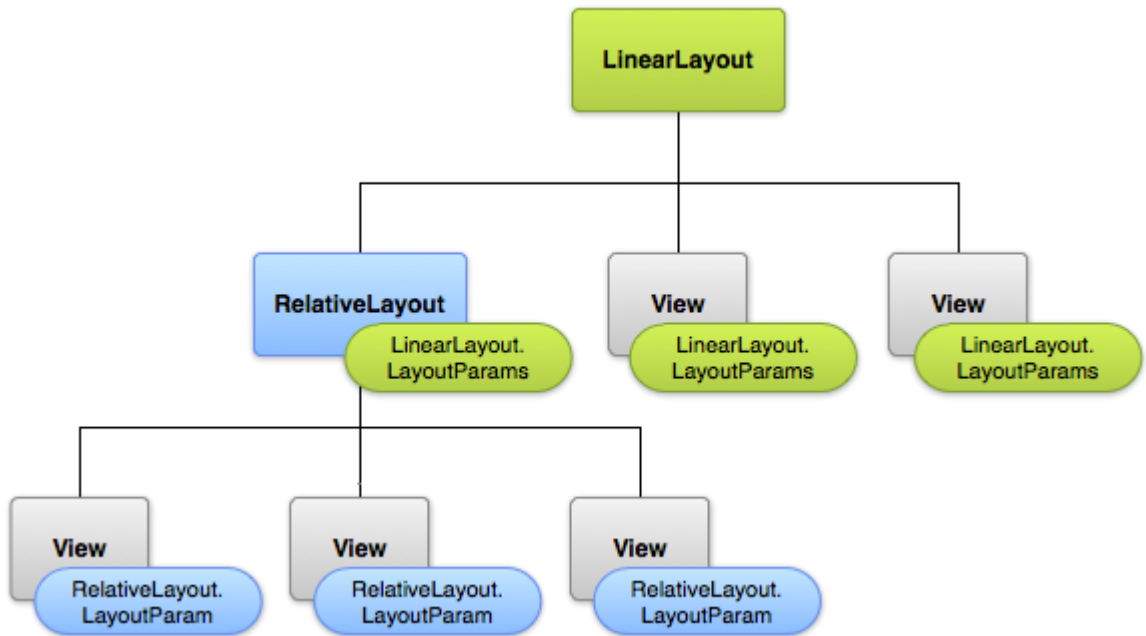
Defining IDs for view objects is important when creating a RelativeLayout. In a relative layout, sibling views can define their layout relative to another sibling view, which is referenced by the unique ID.

An ID need not be unique throughout the entire tree, but it should be unique within the part of the tree you are searching (which may often be the entire tree, so it's best to be completely unique when possible).

## **Layout Parameters**

XML layout attributes named *layout\_something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides.

Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group. As you can see in figure 1, the parent view group defines layout parameters for each child view (including the child view group).



**Figure 4.** Visualization of a view hierarchy with layout parameters associated with each view.

Note that every `LayoutParams` subclass has its own syntax for setting values. Each child element must define `LayoutParams` that are appropriate for its parent, though it may also define different `LayoutParams` for its own children.

All view groups include a width and height (`layout_width` and `layout_height`), and each view is required to define them. Many `LayoutParams` also include optional margins and borders.

You can specify width and height with exact measurements, though you probably won't want to do this often. More often, you will use one of these constants to set the width or height:

- ***wrap\_content*** tells your view to size itself to the dimensions required by its content.



- ***match\_parent*** tells your view to become as big as its parent view group will allow.

In general, specifying a layout width and height using absolute units such as pixels is not recommended. Instead, using relative measurements such as density-independent pixel units (*dp*), ***wrap\_content***, or ***match\_parent***, is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes. The accepted measurement types are defined in the Available Resources document.

### **Layout Position**

The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.

It is possible to retrieve the location of a view by invoking the methods `getLeft()` and `getTop()`. The former returns the left, or X, coordinate of the rectangle representing the view. The latter returns the top, or Y, coordinate of the rectangle representing the view. These methods both return the location of the view relative to its parent. For instance, when `getLeft()` returns 20, that means the view is located 20 pixels to the right of the left edge of its direct parent.

In addition, several convenience methods are offered to avoid unnecessary computations, namely `getRight()` and `getBottom()`. These methods return the coordinates of the right and bottom edges of the rectangle representing the view. For instance, calling `getRight()` is similar to the following computation: `getLeft() + getWidth()`.

## Size, Padding and Margins

The size of a view is expressed with a width and a height. A view actually possess two pairs of width and height values.

The first pair is known as *measured width* and *measured height*. These dimensions define how big a view wants to be within its parent. The measured dimensions can be obtained by calling `getMeasuredWidth()` and `getMeasuredHeight()`.

The second pair is simply known as *width* and *height*, or sometimes *drawing width* and *drawing height*. These dimensions define the actual size of the view on screen, at drawing time and after layout. These values may, but do not have to, be different from the measured width and height. The width and height can be obtained by calling `getWidth()` and `getHeight()`.

To measure its dimensions, a view takes into account its padding. The padding is expressed in pixels for the left, top, right and bottom parts of the view. Padding can be used to offset the content of the view by a specific number of pixels. For instance, a left padding of 2 will push the view's content by 2 pixels to the right of the left edge. Padding can be set using the `setPadding(int, int, int, int)` method and queried by calling `getPaddingLeft()`, `getPaddingTop()`, `getPaddingRight()` and `getPaddingBottom()`.

## Common Layouts

Each subclass of the `ViewGroup` class provides a unique way to display the views you nest within it. Below are some of the more common layout types that are built into the Android platform.

**Note:** Although you can nest one or more layouts within another layout to achieve your UI design, you should strive to keep your layout hierarchy as shallow as possible. Your layout draws faster if it has fewer nested layouts (a wide view hierarchy is better than a deep view hierarchy).

### Linear Layout



A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

### Relative Layout



Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

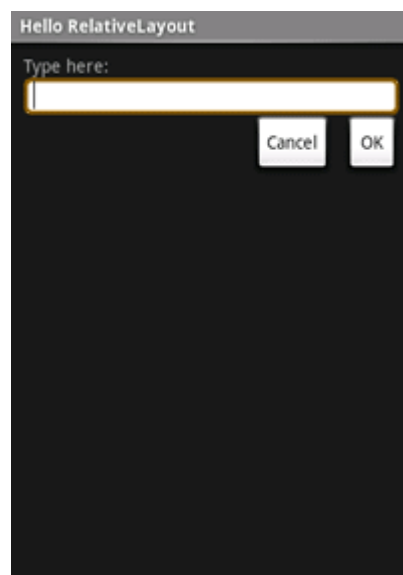
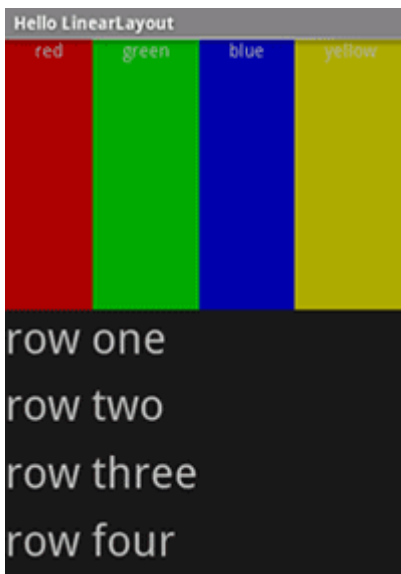
### Web View



Displays web pages.

### Lab exercises

1. Create an app that illustrates that the activity lifecycle method being triggered by various action. Understand when onCreate(),onStart(),onResume event occur.
2. Create a Calculator app that does the function of multiplication, addition, division, subtraction but displays the result in the format of:-Num1 operator num2 = result. Back button on the next activity should get back to the calculator activity again.
3. Create the following given scenario using linear and relative layout concept.



4. Create an app such that when the user click on the given URL typed by the user, it visits the corresponding page.

---

[OBSERVATION SPACE – LAB 2]

[OBSERVATION SPACE – LAB 2]

[OBSERVATION SPACE – LAB 2]

[OBSERVATION SPACE – LAB 2]



[OBSERVATION SPACE – LAB 2]

[OBSERVATION SPACE – LAB 2]

**LAB NO: 3**

**Date:**

## **ACTIVITY AND LAYOUT CONTINUED**

### **Objectives**

- To acquire knowledge on list view, grid view, table view in android application.
- To develop interactive mobile application with multiple pages.

### **Building Layouts with an Adapter**

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses `AdapterView` to populate the layout with views at runtime. A subclass of the `AdapterView` class uses an `Adapter` to bind data to its layout. The `Adapter` behaves as a middleman between the data source and the `AdapterView` layout—the `Adapter` retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the `AdapterView` layout.

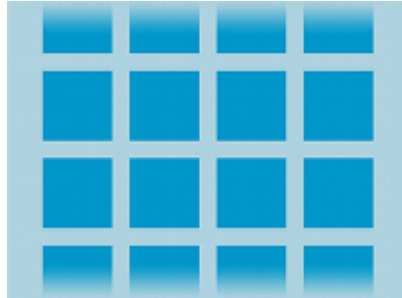
Common layouts backed by an adapter include:

### **List View**



Displays a scrolling single column list.

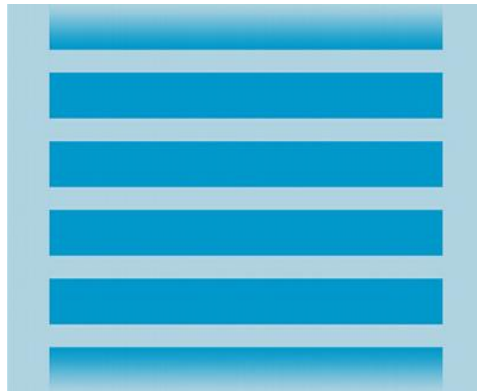
## Grid View



Displays a scrolling grid of columns and rows.

## Details about list view

ListView is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database query and converts each item result into a view that's placed into the list. For an introduction to how you can dynamically insert views using an adapter, read Building Layouts with an Adapter.



## Using a Loader

Using a `CursorLoader` is the standard way to query a `Cursor` as an asynchronous task in order to avoid blocking your app's main thread with the query. When the `CursorLoader` receives the `Cursor` result, the `LoaderCallbacks` receives a callback to `onLoadFinished()`, which is where you update your `Adapter` with the new `Cursor` and the list view then displays the results.

Although the `CursorLoader` APIs were first introduced in Android 3.0 (API level 11), they are also available in the Support Library so that your app may use them while supporting devices running Android 1.6 or higher.

## Example

The following example uses `ListActivity`, which is an activity that includes a `ListView` as its only layout element by default. It performs a query to the `Contacts Provider` for a list of names and phone numbers. The activity implements the `LoaderCallbacks` interface in order to use a `CursorLoader` that dynamically loads the data for the list view.

```
public class ListViewLoader extends ListActivity
    implements LoaderManager.LoaderCallbacks<Cursor> {

    // This is the Adapter being used to display the list's data
    SimpleCursorAdapter mAdapter;

    // These are the Contacts rows that we will retrieve
    static final String[] PROJECTION = new String[]
    {ContactsContract.Data._ID,
        ContactsContract.Data.DISPLAY_NAME};

    // This is the select criteria
    static final String SELECTION = "(" +
        ContactsContract.Data.DISPLAY_NAME + " NOTNULL) AND (" +
        ContactsContract.Data.DISPLAY_NAME + " != ' ' )";
```

```

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

    // Create a progress bar to display while the list loads
    ProgressBar progressBar = new ProgressBar(this);
    progressBar.setLayoutParams(new
LayoutParams(LayoutParams.WRAP_CONTENT,
    LayoutParams.WRAP_CONTENT, Gravity.CENTER));
    progressBar.setIndeterminate(true);
    getListView().setEmptyView(progressBar);

    // Must add the progress bar to the root of the layout
    ViewGroup root = (ViewGroup) findViewById(android.R.id.content);
    root.addView(progressBar);

    // For the cursor adapter, specify which columns go into which views
    String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME};
    int[] toViews = {android.R.id.text1}; // The TextView in
simple_list_item_1

    // Create an empty adapter we will use to display the loaded data.
    // We pass null for the cursor, then update it in onLoadFinished()
    mAdapter = new SimpleCursorAdapter(this,
        android.R.layout.simple_list_item_1, null,
        fromColumns, toViews, 0);
    setListAdapter(mAdapter);

    // Prepare the loader. Either re-connect with an existing one,
    // or start a new one.
    getLoaderManager().initLoader(0, null, this);
}

// Called when a new Loader needs to be created
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // Now create and return a CursorLoader that will take care of
    // creating a Cursor for the data being displayed.
    return new CursorLoader(this, ContactsContract.Data.CONTENT_URI,
        PROJECTION, SELECTION, null, null);
}

```

```

        // Called when a previously created loader has finished loading
        public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
            // Swap the new cursor in. (The framework will take care of closing
the
            // old cursor once we return.)
            mAdapter.swapCursor(data);
        }

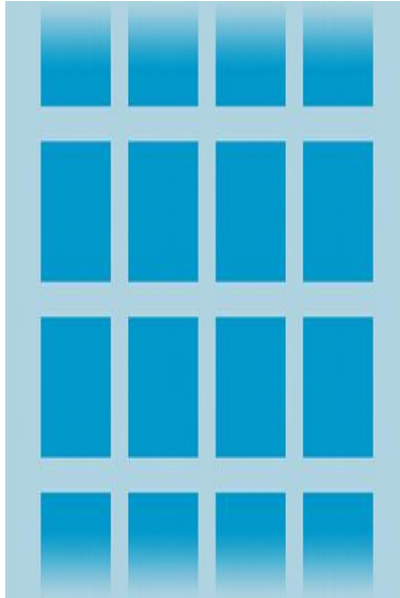
        // Called when a previously created loader is reset, making the data
unavailable
        public void onLoaderReset(Loader<Cursor> loader) {
            // This is called when the last Cursor provided to onLoadFinished()
            // above is about to be closed. We need to make sure we are no
            // longer using it.
            mAdapter.swapCursor(null);
        }

        @Override
        public void onItemClick(ListView l, View v, int position, long id) {
            // Do something when a list item is clicked
        }
    }
}

```

## Grid View

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a ListAdapter. For an introduction to how you can dynamically insert views using an adapter, read Building Layouts with an Adapter.



### Example

In this tutorial, you'll create a grid of image thumbnails. When an item is selected, a toast message will display the position of the image.

1. Start a new project named HelloGridView.
2. Find some photos you'd like to use, or download these sample images. Save the image files into the project's res/drawable/ directory.
3. Open the res/layout/main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
```



```

        android:numColumns="auto_fit"
        android:verticalSpacing="10dp"
        android:horizontalSpacing="10dp"
        android:stretchMode="columnWidth"
        android:gravity="center"
    />

```

1. Open HelloGridView.java and insert the following code for the onCreate() method:

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v,
            int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position,
                Toast.LENGTH_SHORT).show();
        }
    });
}

```

After the main.xml layout is set for the content view, the GridView is captured from the layout with findViewById(int). The setAdapter() method then sets a custom adapter (ImageAdapter) as the source for all items to be displayed in the grid. The ImageAdapter is created in the next step.

To do something when an item in the grid is clicked, the.setOnItemClickListener() method is passed a new AdapterView.OnItemClickListener. This anonymous instance defines the onItemClick() callback method to show a Toast that displays the index position

(zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

2. Create a new class called ImageAdapter that extends BaseAdapter:

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {
            // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
}
```

```

    }

    // references to our images
    private Integer[] mThumbIds = {
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7
    };
}

```

First, this implements some required methods inherited from `BaseAdapter`. The constructor and `getCount()` are self-explanatory. Normally, `getItem(int)` should return the actual object at the specified position in the adapter, but it's ignored for this example. Likewise, `getItemId(int)` should return the row id of the item, but it's not needed here.

The first method necessary is `getView()`. This method creates a new `View` for each image added to the `ImageAdapter`. When this is called, a `View` is passed in, which is normally a recycled object (at least after this has been called once), so there's a check to see if the object is null. If it *is* null, an `ImageView` is instantiated and configured with desired properties for the image presentation:

- a. `setLayoutParams(ViewGroup.LayoutParams)` sets the height and width for the `View`—this ensures that, no matter the size of the drawable, each image is resized and cropped to fit in these dimensions, as appropriate.

- b. `setScaleType(ImageView.ScaleType)` declares that images should be cropped toward the center (if necessary).
- c. `setPadding(int, int, int, int)` defines the padding for all sides. (Note that, if the images have different aspect-ratios, then less padding will cause more cropping of the image if it does not match the dimensions given to the `ImageView`.)

If the `View` passed to `getView()` is *not* null, then the local `ImageView` is initialized with the recycled `View` object.

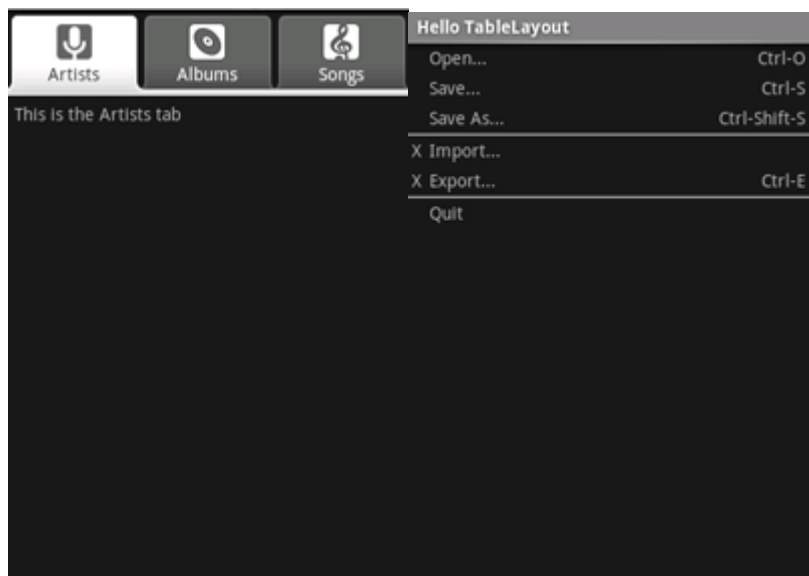
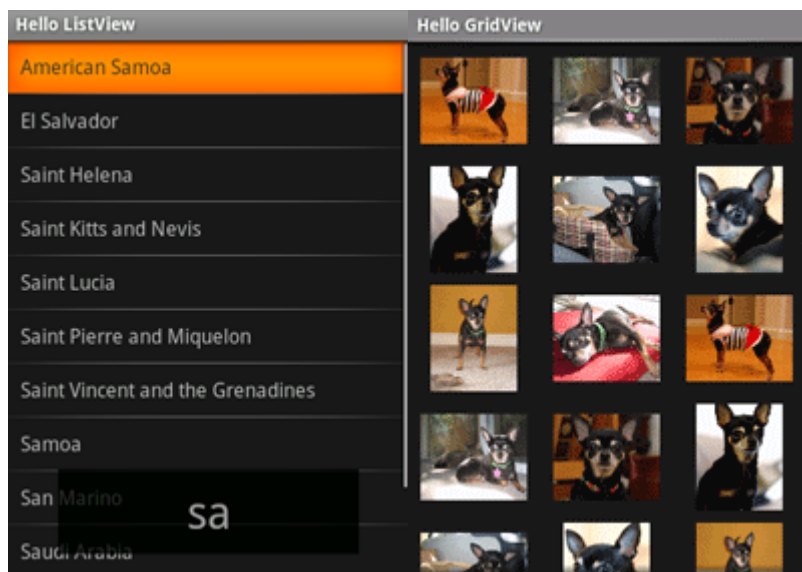
At the end of the `getView()` method, the position integer passed into the method is used to select an image from the `mThumbIds` array, which is set as the image resource for the `ImageView`.

All that's left is to define the `mThumbIds` array of drawable resources.

3. Run the application.

### Lab exercises

1. Using given scenario develop an application to perform following layout operations
  - List view
  - Grid view
  - Tab layout
  - Table layout



2. Write a program to list some grocery items and print the item selected from the list.

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]



[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

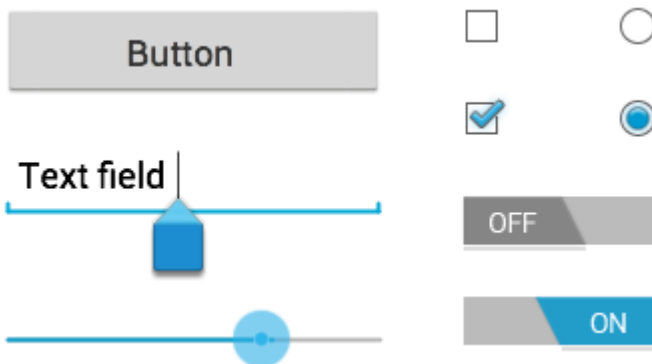
**LAB NO: 4****Date:****INPUT CONTROLS IN ANDROID****Objectives**

- To learn the usage of interactive components in application.
- To learn the interactive mobile application development using variety of control inputs.

**Input controls**

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons, and many more.

Adding an input control to your UI is as simple as adding an XML element to your XML layout. For example, here's a layout with a text field and button:



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

## Buttons

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.



Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

- With text, using the Button class:

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

- With an icon, using the ImageButton class:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

- With text and an icon, using the Button class with the android:drawableLeft attribute:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

## Responding to Click Events

When the user clicks a button, the Button object receives an on-click event.

To define the click event handler for a button, add the android:onClick attribute to the <Button> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

For example, here's a layout with a button using android:onClick:

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

```
android:text="@string/button_send"  
android:onClick="sendMessage" />
```

Within the Activity that hosts this layout, the following method handles the click event:

```
/** Called when the user touches the button */  
public void sendMessage(View view) {  
    // Do something in response to button click  
}
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a View as its only parameter (this will be the View that was clicked)

### Using an OnClickListener

You can also declare the click event handler programmatically rather than in an XML layout. This might be necessary if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.

To declare the event handler programmatically, create an `View.OnClickListener` object and assign it to the button by calling `setOnClickListener(View.OnClickListener)`. For example:

```
Button button = (Button) findViewById(R.id.button_send);  
button.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        // Do something in response to button click  
    }  
});
```

## Styling Your Button

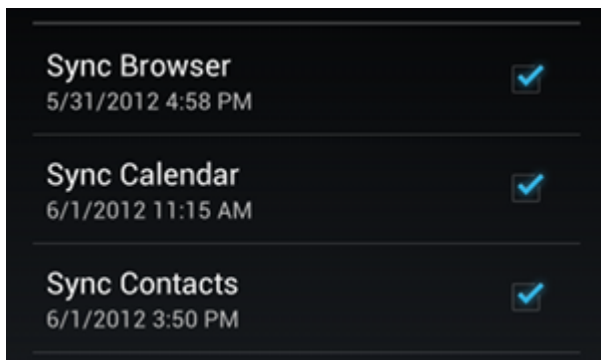
The appearance of your button (background image and font) may vary from one device to another, because devices by different manufacturers often have different default styles for input controls.

You can control exactly how your controls are styled using a theme that you apply to your entire application. For instance, to ensure that all devices running Android 4.0 and higher use the Holo theme in your app, declare `android:theme="@android:style/Theme.Holo"` in your manifest's `<application>` element. Also read the blog post, [Holo Everywhere](#) for information about using the Holo theme while supporting older devices.

To customize individual buttons with a different background, specify the `android:background` attribute with a drawable or color resource. Alternatively, you can apply a style for the button, which works in a manner similar to HTML styles to define multiple style properties such as the background, font, size, and others. For more information about applying styles, see [Styles and Themes](#).

## Check box

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.





To create each checkbox option, create a `CheckBox` in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

## Responding to Click Events

When the user selects a checkbox, the `CheckBox` object receives an on-click event. To define the click event handler for a checkbox, add the `android:onClick` attribute to the `<CheckBox>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

For example, here are a couple `CheckBox` objects in a list:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
```

```
        android:onClick="onCheckboxClicked"/>
    </LinearLayout>
```

Within the Activity that hosts this layout, the following method handles the click event for both checkboxes:

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

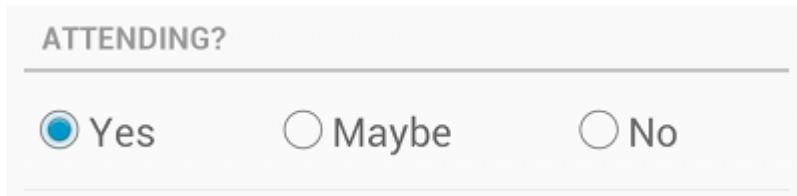
    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich    } }
    }
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. Specifically, the method must:

- Be public
- Return void
- Define a View as its only parameter (this will be the View that was clicked)

### Radio Button

Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a spinner instead.



ATTENDING?

☒ Yes      ☐ Maybe      ☐ No

To create each radio button option, create a `RadioButton` in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a `RadioGroup`. By grouping them together, the system ensures that only one radio button can be selected at a time.

## Responding to Click Events

When the user selects one of the radio buttons, the corresponding `RadioButton` object receives an on-click event. To define the click event handler for a button, add the `android:onClick` attribute to the `<RadioButton>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

For example, here are a couple `RadioButton` objects:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

**Note:** The `RadioGroup` is a subclass of `LinearLayout` that has a vertical orientation by default. Within the Activity that hosts this layout, the following method handles the click event for both radio buttons:

```
public void onRadioButtonClicked(View view) {  
    // Is the button now checked?  
    boolean checked = ((RadioButton) view).isChecked();  
  
    // Check which radio button was clicked  
    switch(view.getId()) {  
        case R.id.radio_pirates:  
            if (checked)  
                // Pirates are the best  
                break;  
        case R.id.radio_ninjas:  
            if (checked)  
                // Ninjas rule  
                break;  
    }  
}
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above.

Specifically, the method must:

- Be public
- Return void

### **Toggle button**

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the `ToggleButton` object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a `Switch` object. `SwitchCompat` is a version of the `Switch` widget which runs on devices back to API 7.

If you need to change a button's state yourself, you can use the `CompoundButton.setChecked()` or `CompoundButton.toggle()` methods.



### *Toggle buttons*



### *Switches (in Android 4.0+)*

***Responding to Button Presses***

To detect when the user activates the button or switch, create an `CompoundButton.OnCheckedChangeListener` object and assign it to the button by calling `setOnCheckedChangeListener()`. For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
    {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

**Toast message**

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.



If user response to a status message is required, consider instead using a Notification.

### The Basics

First, instantiate a Toast object with one of the `makeText()` methods. This method takes three parameters: the application Context, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with `show()`, as shown in the following example:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

This example demonstrates everything you need for most toast notifications. You should rarely need anything else. You may, however, want to position the toast differently or



even use your own layout instead of a simple text message. The following sections describe how you can do these things.

You can also chain your methods and avoid holding on to the Toast object, like this:

```
Toast.makeText(context, text, duration).show();
```

### Positioning your Toast

A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the `setGravity(int, int, int)` method. This accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

For example, if you decide that the toast should appear in the top-left corner, you can set the gravity like this:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

If you want to nudge the position to the right, increase the value of the second parameter. To nudge it down, increase the value of the last parameter.

### Lab exercises

1. Create a simple app that list all the android versions with their icons using list fragment.
2. Create a “Test App” which contains View containing Button, Toggle button. On click of each of this button a custom toast message should come having different images as their content.

3. Write an app such that a View Contains button labelled different versions of Android. On click of each button toast message should get displayed along with the icon of corresponding android version as well as the name of the version.
4. Create a View where in different image should appear for different states of Toggle Button labelled as “Current Profile” which has two states “Silent” and “Ringing “and correspondingly Toast message should display which state is on .Corresponding to each mode on click of button “Action” it should change to corresponding mode.
5. Create a “Food Ordering App” which lists food items with check boxes. Once the user checks /unchecks the item and click on the submit button display the items ordered along with cost of each item and total cost as a Toast message. Once the user clicks on the submit button he/she should not be allowed to change the order i.e. he should not be allowed to change the state of the items checked.

---

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

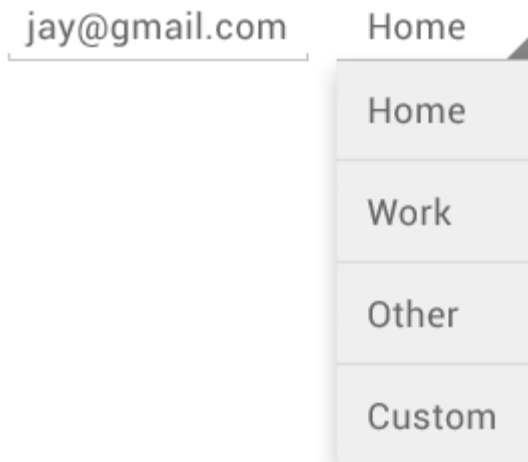


**LAB NO.: 5****Date:****INPUT CONTROLS-SPINNERS, PICKERS****Objectives**

- To introduce the concept of spinners and pickers
- To develop an interactive mobile UI for mobile phones

**Spinners**

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.



You can add a spinner to your layout with the Spinner object. You should usually do so in your XML layout with a `<Spinner>` element. For example:

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

To populate the spinner with a list of choices, you then need to specify a `SpinnerAdapter` in your Activity or Fragment source code.

### Populate the Spinner with User Choices

The choices you provide for the spinner can come from any source, but must be provided through an `SpinnerAdapter`, such as an `ArrayAdapter` if the choices are available in an array or a `CursorAdapter` if the choices are available from a database query. For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

With an array such as this one, you can use the following code in your Activity or Fragment to supply the spinner with the array using an instance of `ArrayAdapter`:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
```

```

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
;
// Apply the adapter to the spinner
spinner.setAdapter(adapter);

```

The `createFromResource()` method allows you to create an `ArrayAdapter` from the string array. The third argument for this method is a layout resource that defines how the selected choice appears in the spinner control. The `simple_spinner_item` layout is provided by the platform and is the default layout you should use unless you'd like to define your own layout for the spinner's appearance.

You should then call `setDropDownViewResource(int)` to specify the layout the adapter should use to display the list of spinner choices (`simple_spinner_dropdown_item` is another standard layout defined by the platform).

Call `setAdapter()` to apply the adapter to your `Spinner`.

## Responding to User Selections

When the user selects an item from the drop-down, the `Spinner` object receives an on-item-selected event. To define the selection event handler for a spinner, implement the `AdapterView.OnItemSelectedListener` interface and the corresponding `onItemSelected()` callback method. For example, here's an implementation of the interface in an `Activity`:

```

public class SpinnerActivity extends Activity implements OnItemSelectedListener {
    ...

    public void onItemSelected(AdapterView<?> parent, View view,
        int pos, long id) {

```

```
// An item was selected. You can retrieve the selected item using
// parent.getItemAtPosition(pos)
}

public void onNothingSelected(AdapterView<?> parent) {
    // Another interface callback
}
}
```

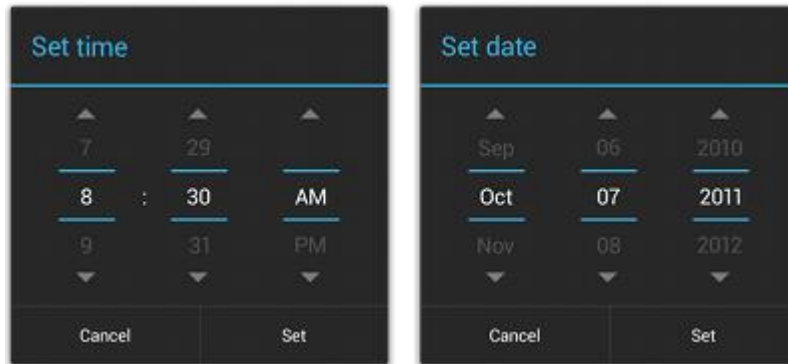
The `AdapterView.OnItemSelectedListener` requires the `onItemSelected()` and `onNothingSelected()` callback methods.

Then you need to specify the interface implementation by calling `setOnItemSelectedListener()`:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

## Pickers

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.



We recommend that you use `DialogFragment` to host each time or date picker. The `DialogFragment` manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.

Although `DialogFragment` was first added to the platform in Android 3.0 (API level 11), if your app supports versions of Android older than 3.0—even as low as Android 1.6—you can use the `DialogFragment` class that's available in the support library for backward compatibility.

**Note:** The code samples below show how to create dialogs for a time picker and date picker using the support library APIs for `DialogFragment`. If your app's `minSdkVersion` is 11 or higher, you can instead use the platform version of `DialogFragment`.

### Creating a Time Picker

To display a `TimePickerDialog` using `DialogFragment`, you need to define a fragment class that extends `DialogFragment` and return a `TimePickerDialog` from the fragment's `onCreateDialog()` method.

**Note:** If your app supports versions of Android older than 3.0, be sure you've set up your Android project with the support library as described in [Setting Up a Project to Use a Library](#).

Extending `DialogFragment` for a time picker

To define a `DialogFragment` for a `TimePickerDialog`, you must:

- Define the `onCreateDialog()` method to return an instance of `TimePickerDialog`
- Implement the `TimePickerDialog.OnTimeSetListener` interface to receive a callback when the user sets the time.

Here's an example:

```
public static class TimePickerFragment extends DialogFragment
    implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
            DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

See the TimePickerDialog class for information about the constructor arguments.

Now all you need is an event that adds an instance of this fragment to your activity.

### Showing the time picker

Once you've defined a DialogFragment like the one shown above, you can display the time picker by creating an instance of the DialogFragment and calling show().

For example, here's a button that, when clicked, calls a method to show the dialog:

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

When the user clicks this button, the system calls the following method:

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

This method calls show() on a new instance of the DialogFragment defined above. The show() method requires an instance of FragmentManager and a unique tag name for the fragment.

### Creating a Date Picker

Creating a DatePickerDialog is just like creating a TimePickerDialog. The only difference is the dialog you create for the fragment.

To display a DatePickerDialog using DialogFragment, you need to define a fragment class that extends DialogFragment and return a DatePickerDialog from the fragment's onCreateDialog() method.

Extending DialogFragment for a date picker

To define a DialogFragment for a DatePickerDialog, you must:

- Define the onCreateDialog() method to return an instance of DatePickerDialog
- Implement the DatePickerDialog.OnDateSetListener interface to receive a callback when the user sets the date.

Here's an example:

```
public static class DatePickerFragment extends DialogFragment
    implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user
    }
}
```

See the DatePickerDialog class for information about the constructor arguments.

Now all you need is an event that adds an instance of this fragment to your activity.

Showing the date picker



Once you've defined a DialogFragment like the one shown above, you can display the date picker by creating an instance of the DialogFragment and calling show().

For example, here's a button that, when clicked, calls a method to show the dialog:

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

When the user clicks this button, the system calls the following method:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

This method calls show() on a new instance of the DialogFragment defined above. The show() method requires an instance of FragmentManager and a unique tag name for the fragment.

### Lab exercise

1. Create an App ,”Vehicle Parking Registration”, where in the user selects from a range of different vehicles type such as car, bike etc. from a spinner list ,and enters the vehicle no and RC no into a text area .Once the user click on the submit button ,details entered by user should display in other View and ask for user confirmation or give an edit option. Once the details are confirmed from the user, the details should be displayed in a toast message with a specific serial no. confirming allotment of parking.

2. Create a ticket booking app, where the user selects destination and source from spinner and date of journey from the date picker. Using toggle button provide an option to select to –fro ticket or for just one-way journey. Provide “Submit”, “Clear” button. On click of “Submit ” button all the entered details should be displayed in next form in proper format. “Clear ” button should clear all the fields and reset the date picker to current time of the system.
  
3. Create a ticket booking app, where the user selects destination and source from spinner and date of journey from the date picker, suitable time of journey from time picker. Provide “Submit”, “Clear” button. On click of “Submit ” button all the entered details should be displayed in next form in proper format along with available trains in those time duration. “Clear ” button should clear all the fields and reset the date picker to current time of the system. Using toggle button provide option to either book general ticket or tatkal. If the option states “tatkal ” is on ,then the user should be allowed to click on the submit button only after 1100 hours. Do all the validations required

---

[OBSERVATION SPACE – LAB 5]

**[OBSERVATION SPACE – LAB 5]**

**[OBSERVATION SPACE – LAB 5]**

**[OBSERVATION SPACE – LAB 5]**

**LAB NO. 6**

**Date:**

**INTRODUCTION TO MENU**

**Objectives**

- To understand how menu works with android and how to make it runnable in web browser
- To learn how to create options menu and an app bar using android
- To learn how to create context menu using android
- To learn how to create a pop up menu in android

**What is a Menu?**

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This guide shows how to create the three fundamental types of menus or action presentations on all versions of Android:

**Options menu and app bar**

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

## Context menu and contextual action mode

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

## Popup menu

A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should not directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

## Defining a Menu in XML

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

To define the menu, create an XML file inside your project's `res/menu/` directory and build the menu with the following elements:

<menu>	Defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item> and <group> elements
<item>	Creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu

<group>	An optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.
---------	--

Here's an example menu named game\_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
  <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

The <item> element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

android:id	A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.
android:icon	A reference to a drawable to use as the item's icon.
android:title	A reference to a string to use as the item's title.
android:showAsAction	Specifies when and how this item should appear as an action item in the app bar.

You can add a submenu to an item in any menu (except a submenu) by adding a <menu> element as the child of an <item>. Submenus are useful when your application has a lot of functions that can be organized into topics, like items in a PC application's menu bar (File, Edit, View, etc.). For example:

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



```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/file"
        android:title="@string/file" >
    <!-- "file" submenu -->
    <menu>
        <item android:id="@+id/create_new"
              android:title="@string/create_new" />
        <item android:id="@+id/open"
              android:title="@string/open" />
    </menu>
  </item>
</menu>
```

To use the menu in your activity, you need to inflate the menu resource (convert the XML resource into a programmable object) using `MenuInflater.inflate()`. In the following sections, you'll see how to inflate a menu for each menu type.

### Creating an Options Menu

The options menu is where you should include actions and other options that are relevant to the current activity context, such as "Search," "Compose email," and "Settings."

Where the items in your options menu appear on the screen depends on the version for which you've developed your application:

- If you've developed your application for Android 2.3.x (API level 10) or lower, the contents of your options menu appear at the bottom of the screen when the user presses the Menu button, as shown in figure 1. When opened, the first visible portion is the icon menu, which holds up to six menu items. If your menu includes more than six items, Android places the sixth item and the rest into the overflow menu, which the user can open by selecting `More`.

You can declare items for the options menu from either your Activity subclass or a Fragment subclass. If both your activity and fragment(s) declare items for the options menu, they are combined in the UI. The activity's items appear first, followed by those of each fragment in the order in which each fragment is added to the activity. If necessary, you can re-order the menu items with the `android:orderInCategory` attribute in each `<item>` you need to move.



Figure 1. Options menu in the Browser, on Android 2.3.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

To specify the options menu for an activity, override `onCreateOptionsMenu()` (fragments provide their own `onCreateOptionsMenu()` callback). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback. Example code given above:

You can also add menu items using `add()` and retrieve items with `findItem()` to revise their properties with `MenuItem` APIs.

If you've developed your application for Android 2.3.x and lower, the system calls `onCreateOptionsMenu()` to create the options menu when the user opens the menu for the first time. If you've developed for Android 3.0 and higher, the system calls `onCreateOptionsMenu()` when starting the activity, in order to show items to the app bar.

### Handling click events

When the user selects an item from the options menu (including action items in the app bar), the system calls your activity's `onOptionsItemSelected()` method. This method passes the `MenuItem` selected. You can identify the item by calling `getItemId()`, which returns the unique ID for the menu item (defined by the `android:id` attribute in the menu resource or with an integer given to the `add()` method). You can match this ID against known menu items to perform the appropriate action. For example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

When you successfully handle a menu item, return `true`. If you don't handle the menu item, you should call the superclass implementation of `onOptionsItemSelected()` (the default implementation returns `false`).

If your activity includes fragments, the system first calls `onOptionsItemSelected()` for the activity then for each fragment (in the order each fragment was added) until one returns `true` or all fragments have been called.

### **Changing menu items at runtime**

After the system calls `onCreateOptionsMenu()`, it retains an instance of the `Menu` you populate and will not call `onCreateOptionsMenu()` again unless the menu is invalidated for some reason. However, you should use `onCreateOptionsMenu()` only to create the initial menu state and not to make changes during the activity lifecycle.

If you want to modify the options menu based on events that occur during the activity lifecycle, you can do so in the `onPrepareOptionsMenu()` method. This method passes you the `Menu` object as it currently exists so you can modify it, such as add, remove, or disable items. (Fragments also provide an `onPrepareOptionsMenu()` callback.)

On Android 2.3.x and lower, the system calls `onPrepareOptionsMenu()` each time the user opens the options menu (presses the `Menu` button).

On Android 3.0 and higher, the options menu is considered to always be open when menu items are presented in the app bar. When an event occurs and you want to perform a menu update, you must call `invalidateOptionsMenu()` to request that the system call `onPrepareOptionsMenu()`.

### **Lab exercises**

1. Create a home page of “XYZ Institute” where in the menu based on Option Menu is used with the following Requirements

a. Create a simple option menus where in once you click the “menu options”, the option items should get displayed which are “Courses”, “Admission”, “Faculty”. On click of each item corresponding content should get displayed. Eg. Courses should display all the course details offered by the institute. Admission should display all the details regarding admission procedure. Faculty should display the details photo of each faculty.

b) This option menu uses images/icons instead of textual content. This textual content should represent “Contact US”, “About Us”, “Homepage”. Once the user clicks on the corresponding icons it should display corresponding content.

**[OBSERVATION SPACE – LAB 6]**

**[OBSERVATION SPACE – LAB 6]**

**[OBSERVATION SPACE – LAB 6]**

**[OBSERVATION SPACE – LAB 6]**



**[OBSERVATION SPACE – LAB 6]**

**[OBSERVATION SPACE – LAB 6]**

**[OBSERVATION SPACE – LAB 6]**

**LAB NO. 7**

**Date:**

**CREATING CONTEXTUAL AND POP-UP MENUS**

**Objectives**

- To demonstrate significant experience with the contextual menu's
- To create suitable application using pop menus.

**Creating Contextual Menus**

A contextual menu offers actions that affect a specific item or context frame in the UI. You can provide a context menu for any view, but they are most often used for items in a ListView, GridView, or other view collections in which the user can perform direct actions on each item.

There are two ways to provide contextual actions:

- In a floating context menu. A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.
- In the contextual action mode. This mode is a system implementation of `ActionMode` that displays a contextual action bar at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it).

**Creating a floating context menu**

To provide a floating context menu:

1. Register the View to which the context menu should be associated by calling `registerForContextMenu()` and pass it the View.

If your activity uses a ListView or GridView and you want each item to provide the same context menu, register all items for a context menu by passing the ListView or GridView to `registerForContextMenu()`.

2. Implement the `onCreateContextMenu()` method in your Activity or Fragment.

When the registered view receives a long-click event, the system calls your `onCreateContextMenu()` method. This is where you define the menu items, usually by inflating a menu resource. For example:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                               ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

1. `MenuInflater` allows you to inflate the context menu from a menu resource. The callback method parameters include the `View` that the user selected and a `ContextMenu.ContextMenuInfo` object that provides additional information about the item selected. If your activity has several views that each provide a different context menu, you might use these parameters to determine which context menu to inflate.

2. Implement `onContextItemSelected()`.

When the user selects a menu item, the system calls this method so you can perform the appropriate action. For example:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

The `getItemId()` method queries the ID for the selected menu item, which you should assign to each menu item in XML using the `android:id` attribute, as shown in the section about Defining a Menu in XML.

When you successfully handle a menu item, return `true`. If you don't handle the menu item, you should pass the menu item to the superclass implementation. If your activity includes fragments, the activity receives this callback first. By calling the superclass when unhandled, the system passes the event to the respective callback method in each fragment, one at a time (in the order each fragment was added) until `true` or `false` is returned. (The default implementation for `Activity` and `android.app.Fragment` return `false`, so you should always call the superclass when unhandled.)

### Using the contextual action mode

The contextual action mode is a system implementation of `ActionMode` that focuses user interaction toward performing contextual actions. When a user enables this mode by selecting an item, a contextual action bar appears at the top of the screen to present actions the user can perform on the currently selected item(s). While this mode is enabled, the user can select multiple items (if you allow it), deselect items, and continue to navigate within the activity (as much as you're willing to allow). The action mode is disabled and the contextual action bar disappears when the user deselects all items, presses the `BACK` button, or selects the `Done` action on the left side of the bar.

For views that provide contextual actions, you should usually invoke the contextual action mode upon one of two events (or both):

- The user performs a long-click on the view.
- The user selects a checkbox or similar UI component within the view.

How your application invokes the contextual action mode and defines the behavior for each action depends on your design. There are basically two designs:

- For contextual actions on individual, arbitrary views.
- For batch contextual actions on groups of items in a `ListView` or `GridView` (allowing the user to select multiple items and perform an action on them all).

The following sections describe the setup required for each scenario.

Enabling the contextual action mode for individual views

If you want to invoke the contextual action mode only when the user selects specific views, you should:

1. Implement the `ActionMode.Callback` interface. In its callback methods, you can specify the actions for the contextual action bar, respond to click events on action items, and handle other lifecycle events for the action mode.
2. Call `startActionMode()` when you want to show the bar (such as when the user long-clicks the view)

For example:

1. Implement the `ActionMode.Callback` interface:

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {

    // Called when the action mode is created; startActionMode() was called
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate a menu resource providing context menu items
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    // Called each time the action mode is shown. Always called after
    onCreateActionMode, but
    // may be called multiple times if the mode is invalidated.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }

    // Called when the user selects a contextual menu item
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
```

```

        case R.id.menu_share:
            shareCurrentItem();
            mode.finish(); // Action picked, so close the CAB
            return true;
        default:
            return false;
    }
}

// Called when the user exits the action mode
@Override
public void onDestroyActionMode(ActionMode mode) {
    mActionMode = null;
}
};

```

1. Notice that these event callbacks are almost exactly the same as the callbacks for the options menu, except each of these also pass the `ActionMode` object associated with the event. You can use `ActionMode` APIs to make various changes to the CAB, such as revise the title and subtitle with `setTitle()` and `setSubtitle()` (useful to indicate how many items are selected).

Also notice that the above sample sets the `mActionMode` variable null when the action mode is destroyed. In the next step, you'll see how it's initialized and how saving the member variable in your activity or fragment can be useful.

2. Call `startActionMode()` to enable the contextual action mode when appropriate, such as in response to a long-click on a `View`:

```

someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Called when the user long-clicks on someView
    public boolean onLongClick(View view) {
        if (mActionMode != null) {
            return false;
        }

        // Start the CAB using the ActionMode.Callback defined above

```

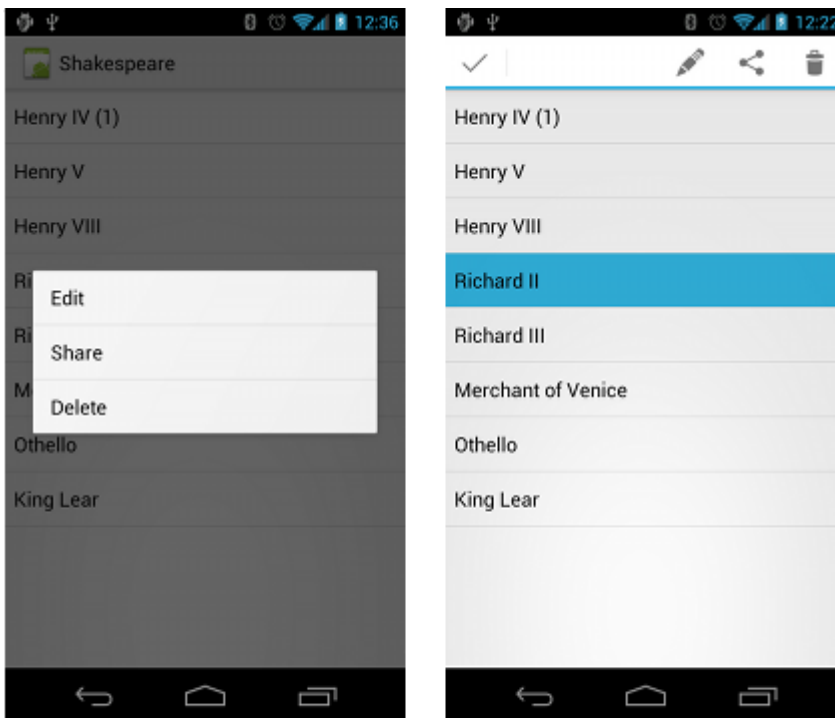


```

        mActionMode = getActivity().startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});

```

When you call `startActionMode()`, the system returns the `ActionMode` created. By saving this in a member variable, you can make changes to the contextual action bar in response to other events. In the above sample, the `ActionMode` is used to ensure that the `ActionMode` instance is not recreated if it's already active, by checking whether the member is null before starting the action mode.

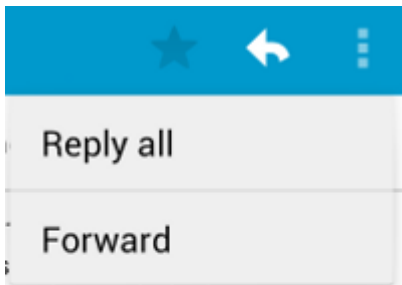


**Figure 3.** Screenshots of a floating context menu (left) and the contextual action bar (right).

## Creating a Popup Menu

A PopupMenu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise. It's useful for:

- Providing an overflow-style menu for actions that relate to specific content (such as Gmail's email headers, shown in figure 4).
- Providing a second part of a command sentence (such as a button marked "Add" that produces a popup menu with different "Add" options).
- Providing a drop-down similar to Spinner that does not retain a persistent selection.



**Figure 4.** A popup menu in the Gmail app, anchored to the overflow button at the top-right.

If you define your menu in XML, here's how you can show the popup menu:

1. Instantiate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.
2. Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu().
3. Call PopupMenu.show().

For example, here's a button with the android:onClick attribute that shows a popup menu:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

The activity can then show the popup menu like this:

```
public void showPopup(View v) {  
    PopupMenu popup = new PopupMenu(this, v);  
    MenuInflater inflater = popup.getMenuInflater();  
    inflater.inflate(R.menu.actions, popup.getMenu());  
    popup.show();  
}
```

In API level 14 and higher, you can combine the two lines that inflate the menu with `PopupMenu.inflate()`.

The menu is dismissed when the user selects an item or touches outside the menu area. You can listen for the dismiss event using `PopupMenu.OnDismissListener`

Handling click events

To perform an action when the user selects a menu item, you must implement the `PopupMenu.OnMenuItemClickListener` interface and register it with your `PopupMenu` by calling `setOnMenuItemClickListener()`. When the user selects an item, the system calls the `onMenuItemClick()` callback in your interface.

For example:

```
public void showMenu(View v) {  
    PopupMenu popup = new PopupMenu(this, v);  
  
    // This activity implements OnMenuItemClickListener  
    popup.setOnMenuItemClickListener(this);  
    popup.inflate(R.menu.actions);  
    popup.show();  
}  
  
@Override  
public boolean onMenuItemClick(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.archive:  
            archive(item);  
            return true;  
    }  
}
```

```
case R.id.delete:
    delete(item);
    return true;
default:
    return false;
}
}
```

### Lab exercises

1: Write a Program which displays your contact details in your phone book. Once you have listed the contact details, on long press, each contact should display the options based on following

- a) whether they are saved in sim/phone
- b) call, message option
- c) show whether they have WhatsApp/Gmail detail saved .

Once any of these sub option is clicked say msg, automatically the corresponding contact should be put into senders message option with the empty message content should be displayed

2. Create a Simple form with a button “Pop-Up Menu”. Once its clicked display the options “One”, “Two”, “Three” as the menu items. Once each item is selected display the corresponding item as selected in a Toast.

3. Create a View with the name “My Menu ” which contains an image as an icon. On click of that icon it shows a submenu as “Image -1”, “Image -2” . On click of each item that particular image should be displayed with the corresponding content in the Toast.

4. Create view which contains Textual content which gives description about “De-Monitization”. Add a filter option with submenu’s as “keywords”-to be given by the user that can be searched in the content, “Sort”-sorting the entire document.

---

**[OBSERVATION SPACE – LAB 7]**

**[OBSERVATION SPACE – LAB 7]**

**[OBSERVATION SPACE – LAB 7]**

**[OBSERVATION SPACE – LAB 7]**



**[OBSERVATION SPACE – LAB 7]**

**LAB NO. 8**

**Date:**

**ANDROID SQLITE & SHARED PREFERENCES**

**Objectives**

- To apply and manipulate several core SQL Queries through SQLite in Android
- To learn and understand SQLite Browser.
- To apply Shared Preference for storing data of an application.

**1. Android SQLite**

SQLite is an open-source relational database i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

It is embedded in android by default. So, there is no need to perform any database setup or administration task.

Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC, ODBC e.t.c

**Database - Package**

The main package is android.database.sqlite that contains the classes to manage your own databases

**Database - Creation**

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object. Its syntax is given below

<pre>SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);</pre>
--

Apart from this , there are other functions available in the database package , that does

this job. They are listed below.

Sr.No	Method & Description
1	<b>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)</b> This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	<b>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)</b> It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	<b>openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)</b> It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.
4	<b>openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)</b> This method is similar to above method but it takes the File object as a path rather than a string. It is equivalent to file.getPath()

## SQLiteDatabase class

It contains methods to be performed on sqlite database such as create, update, delete, select etc.

## Methods of SQLiteDatabase class

There are many methods in SQLiteDatabase class. Some of them are as follows:

Method	Description
<code>void execSQL(String sql)</code>	executes the sql query not select query.
<code>long insert(String table, String nullColumnHack, ContentValues values)</code>	inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
<code>int update(String table, ContentValues values, String whereClause, String[] whereArgs)</code>	updates a row.
<code>Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)</code>	returns a cursor over the resultset.

## Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialPoint(Uname VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialPoint VALUES('admin','admin');");
```

This will insert some values into our table in our database.

## Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor

pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatabase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes.

Sr.No	Method & Description
1	getColumnCount() This method return the total number of columns of the table.
2	getColumnIndex(String columnName) This method returns the index number of a column by specifying the name of the column.
3	getColumnName(int columnIndex) This method returns the name of the column by specifying the index of the column
4	getColumnNames() This method returns the array of all the column names of the table.
5	getCount() This method returns the total number of rows in the cursor.
6	getPosition() This method returns the current position of the cursor in the table
7	isClosed() This method returns true if the cursor is closed and return false otherwise.

### SQLite Browser

DB Browser for SQLite is a high quality, visual, open source tool to create, design, and edit database files compatible with SQLite.

It is for users and developers wanting to create databases, search, and edit data. It uses a familiar spreadsheet-like interface, and you don't need to learn complicated SQL commands.

Controls and wizards are available for users to:

- Create and compact database files
- Create, define, modify and delete tables
- Create, define and delete indexes
- Browse, edit, add and delete records
- Search records
- Import and export records as text
- Import and export tables from/to CSV files
- Import and export databases from/to SQL dump files
- Issue SQL queries and inspect the results
- Examine a log of all SQL commands issued by the application.

### **To view Table structure and Data in Eclipse.**

Here are the steps

1. Download the Sqlite Manager jar file [here](#).
2. Add it to your eclipse > dropins Directory.
3. Restart eclipse.
4. Launch the compatible emulator or device
5. Run your application.
6. Go to Window > Open Perspective > DDMS >
7. Choose the running device.
8. Go to File Explorer tab.
9. Select the directory called databases under your application's package.
10. Select the .db file under the database directory.
11. Then click Sqlite manager icon like this .
12. Now you're able to see the .db file.

## 2.Shared Preferences

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of key,value pair.

In order to use shared preferences, you have to call a method `getSharedPreferences()` that returns a `SharedPreferences` instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedPreferences = getSharedPreferences(MyPREFERENCES,
Context.MODE_PRIVATE);
```

The first parameter is the key and the second parameter is the MODE. Apart from private there are other modes available that are listed below :

Sr.No	Mode & description
1.	MODE_APPEND  This will append the new preferences with the already existing preferences
2.	MODE_ENABLE_WRITE_AHEAD_LOGGING  Database open flag. When it is set , it would enable write ahead logging by default
3.	MODE_MULTI_PROCESS  This method will check for modification of preferences even if the sharedpreference instance has already been loaded
4.	MODE_PRIVATE  By setting this mode, the file can only be accessed using calling application
5.	MODE_WORLD_READABLE

	This mode allow other application to read the preferences.
6.	<b>MODE_WORLD_WRITEABLE</b> This mode allow other application to write the preferences

You can save something in the sharedPreferences by using SharedPreferences.Editor class. You will call the edit method of SharedPreferences instance and will receive it in an editor object. Its syntax is :

```
Editor editor = sharedPreferences.edit();
editor.putString("key", "value");
editor.commit();
```

Apart from the putString method , there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows :

Sr.No	Mode & description
1.	<b>apply()</b>  It is an abstract method. It will commit your changes back from editor to the sharedPreferences object you are calling
2.	<b>clear()</b>  It will remove all values from the editor.
3.	<b>remove(String key)</b>  It will remove the value whose key has been passed as a parameter
4.	<b>putLong(String key, long value)</b>  It will save a long value in a preference editor
5.	<b>putInt(String key, int value)</b>  It will save a integer value in a preference editor
6.	<b>putFloat(String key, float value)</b>



	It will save a float value in a preference editor
--	---

### Lab exercises

- Write a program to create “Contact Book” wherein user has to create the following
  - View that saves the name,phone-no ,emailId of a person.
  - To display the saved contact.
  - To edit the details of any contact
- Write a program to add grocery items along with its cost into the database and display the total cost of all the item selected by the user. Items has to be displayed through spinner.
- Create an application “Movie Review” to create a movie review to do the following:
  - User can write a movie review by stating movie name, year,giving points ranging from 1-5 and save details in a database.
  - User can view the movie review for which review are already defined .Movie names has to be displayed using listview,and the selected movie’s details should be displayed in a table.
- Create an app “Student Details”,which does the following
  - Detail Submission:-Student details are entered through view which asks for name,studentid,semester,branch,faculty incharge.
  - View Students:-Students are listed in View ,and using Context Menu option either a student whole details can be viewed or deleted from database.
  - Edit Details:-Student list is displayed based on student ID and required field is edited.
- Make use of SQLite browser to view the database ,tables created.Modify the tables and upload the same into your project and view the modified data through your app.
- Demonstrates the use of the Shared Preferences through an android application which displays a screen with some text fields. Save the value when the application is closed and brought back when it is opened again.

**[OBSERVATION SPACE – LAB 8]**

**[OBSERVATION SPACE – LAB 8]**

**[OBSERVATION SPACE – LAB 8]**

**[OBSERVATION SPACE – LAB 8]**

**[OBSERVATION SPACE – LAB 8]**

**[OBSERVATION SPACE – LAB 8]**

**LAB NO. 9****Date:****SECURITY AND PERMISSIONS****Objectives**

- To ensure that the designed app is more secure.
- To incorporate best practise that will positively impact app security.

**1. Enforce secure communication**

When you safeguard the data that you exchange between your app and other apps, or between your app and a website, you improve your app's stability and protect the data that you send and receive.

- Use implicit intents and non-exported content providers
  - Show an app chooser

If an implicit intent can launch at least two possible apps on a user's device, explicitly show an app chooser. This interaction strategy allows users to transfer sensitive information to an app that they trust. Sample code is given below:

```
Intent intent = new Intent(Intent.ACTION_SEND);
List<ResolveInfo>
possibleActivitiesList = getPackageManager()
    .queryIntentActivities(intent,
PackageManager.MATCH_ALL);

// Verify that an activity in at least two apps on the user's device
// can handle the intent. Otherwise, start the intent only if an app
// on the user's device can handle the intent.
if (possibleActivitiesList.size() > 1) {

    // Create intent to show chooser.
    // Title is something similar to "Share this photo with".
```



```
String title =  
    getResources().getString(R.string.chooser_title);  
    Intent chooser = Intent.createChooser(intent, title);  
    startActivity(chooser);  
} else if (intent.resolveActivity(getPackageManager()) != null)  
{  
    startActivity(intent);  
}
```

b. Apply signature-based permissions

When sharing data between two apps that you control or own, use signature-based permissions. These permissions don't require user confirmation and instead check that the apps accessing the data are signed using the same signing key. Therefore, these permissions offer a more streamlined, secure user experience.

```
<manifest  
xmlns:android="http://schemas.android.com/apk/res/android"  
package="com.example.myapp">  
    <permission android:name="my_custom_permission_name"  
        android:protectionLevel="signature" />  

```

c. Disallow access to your app's content providers

Unless you intend to send data from your app to a different app that you don't own, you should explicitly disallow other developers' apps from accessing the `ContentProvider` objects that your app contains. This setting is particularly important if your app can be installed on devices running Android 4.1.1 (API level 16) or lower, as the `android:exported` attribute of the `<provider>` element is true by default on those versions of Android.

```

<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.myapp">
  <application ... >
    <provider

        android:name="android.support.v4.content.FileProvider"
        android:authorities="com.example.myapp.fileprovider"
        ...
        android:exported="false">
        <!-- Place child elements of <provider> here. -->
    </provider>
    ...
  </application>
</manifest>

```

- ii. Ask for credentials before showing sensitive information

When requesting credentials from users so that they can access sensitive information or premium content in your app, ask for either a PIN/password/pattern or a biometric credential, such as using face recognition or fingerprint recognition.

- iii. Apply network security measures

- a. Use SSL traffic

If your app communicates with a web server that has a certificate issued by a well-known, trusted CA, the HTTPS request is very simple:

```

URL url = new URL("https://www.google.com");
HttpsURLConnection urlConnection = (HttpsURLConnection)
url.openConnection();
urlConnection.connect();

```

```
InputStream in = urlConnection.getInputStream();
```

iv. Use WebView objects carefully

Whenever possible, load only allow listed content in WebView objects. In other words, the WebView objects in your app shouldn't allow users to navigate to sites that are outside of your control.

In addition, you should never enable JavaScript interface support unless you completely control and trust the content in your app's WebView objects.

a. Use HTML message channels

If your app must use JavaScript interface support on devices running Android 6.0 (API level 23) and higher, use HTML message channels instead of communicating between a website and your app, as shown in the following code snippet:

```
WebView myWebView = (WebView)
findViewById(R.id.webview);

// channel[0] and channel[1] represent the two ports.
// They are already entangled with each other and have been
// started.
WebMessagePort[] channel =
myWebView.createWebMessageChannel();

// Create handler for channel[0] to receive messages.
channel[0].setWebMessageCallback(new
WebMessagePort.WebMessageCallback() {
    @Override
    public void onMessage(WebMessagePort port, WebMessage
message) {
        Log.d(TAG, "On port " + port + ", received this message:
" + message);
    }
});
```

```

    }
  });

  // Send a message from channel[1] to channel[0].
  channel[1].postMessage(new WebMessage("My secure
  message"));

```

## 2. Provide the right permissions

Your app should request only the minimum number of permissions necessary to function properly. When possible, your app should relinquish some of these permissions when they're no longer needed.

### i. Use intents to defer permissions

Whenever possible, don't add a permission to your app to complete an action that could be completed in another app. Instead, use an intent to defer the request to a different app that already has the necessary permission.

The following example shows how to use an intent to direct users to a contacts app instead of requesting the READ\_CONTACTS and WRITE\_CONTACTS permissions:

```

// Delegates the responsibility of creating the contact to a contacts app,
// which has already been granted the appropriate WRITE_CONTACTS
// permission.
Intent insertContactIntent = new Intent(Intent.ACTION_INSERT);
insertContactIntent.setType(ContactsContract.Contacts.CONTENT_T
TYPE);

// Make sure that the user has a contacts app installed on their device.
if (insertContactIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(insertContactIntent);
}

```

In addition, if your app needs to perform file-based I/O—such as accessing storage or choosing a file—it doesn't need special permissions because the system can complete the operations on your app's behalf. Better still, after a user selects content at a particular URI, the calling app gets granted permission to the selected resource.

ii. Share data securely across apps

Follow these best practices in order to share your app's content with other apps in a more secure manner:

- Enforce read-only or write-only permissions as needed.
- Provide clients one-time access to data by using the `FLAG_GRANT_READ_URI_PERMISSION` and `FLAG_GRANT_WRITE_URI_PERMISSION` flags.
- When sharing data, use "content://" URIs, not "file://" URIs. Instances of `FileProvider` do this for you.

The following code snippet shows how to use URI permission grant flags and content provider permissions to display an app's PDF file in a separate PDF Viewer app:

```
// Create an Intent to launch a PDF viewer for a file owned by this app.
Intent viewPdfIntent = new Intent(Intent.ACTION_VIEW);
viewPdfIntent.setData(Uri.parse("content://com.example/personal-
info.pdf"));

// This flag gives the started app read access to the file.
viewPdfIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMI
SSION);

// Make sure that the user has a PDF viewer app installed on their device.
if (viewPdfIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(viewPdfIntent);
}
```

}

### **Lab Exercises**

1. Design an android application which uses chooser intent to allow a user to share a photo using multiple apps like Whatsapp, Email ,Bluetooth, Outlook etc.
2. Design an android application where user is searching for an Retail outlet in browser and direction option will defer to use google map.

---

[OBSERVATION SPACE – LAB 9]



[OBSERVATION SPACE – LAB 9]



[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

## **SERVICE,BROADCAST RECEIVER**

### **Objectives**

- To get acquainted with the concept of services and broadcast receiver.
- To realise the concept through android application.

### **1. Services**

#### **What are services?**

A service is a component which runs in the background without direct interaction with the user. As the service has no user interface, it is not bound to the lifecycle of an activity. Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers and the like.

A Service has two states Started and Bound. A service is started when an application component( as an activity), starts it by calling `startService()`. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. 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, get results, and even do so across processes with interprocess communication (IPC).

Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them. Services can also be configured to be restarted if they get terminated by the Android system once sufficient system resources are available again.

It is possible to assign services the same priority as foreground activities. In this case it is required to have a visible notification active for the related service. It is frequently used for services which play videos or music.

### **a. Services and background processing**

By default, a service runs in the same process as the main thread of the application. Therefore, you need to use asynchronous processing in the service to perform resource intensive tasks in the background. A commonly used pattern for a service implementation is to create and run a new Thread in the service to perform the processing in the background and then to terminate the service once it has finished the processing. Services which run in the process of the application are sometimes called local services.

### **b. Platform service and custom services**

The Android platform provides and runs predefined system services and every Android application can use them, given the right permissions. These system services are usually exposed via a specific Manager class. Access to them can be gained via the `getSystemService()` method. The Context class defines several constants for accessing these services.

An Android application can, in addition to consuming the existing Android platform services, define and use new services. Defining your custom services allows you to design responsive applications. You can fetch the application data via it and once the application is started by the user, it can present fresh data to the user.

#### **i. Starting and defining custom services**

Custom services are started from other Android components, i.e., activities, broadcast receivers and other services.

#### **ii. Foreground services**

A foreground service is a service that should have the same priority as an active activity and therefore should not be killed by the Android system, even if the system is low on memory. A foreground service must provide a notification for the status bar, which is placed under the "Ongoing" heading, which means that the notification cannot be dismissed unless the service is either stopped or removed from the foreground.

```

Notification notification = new Notification(R.drawable.icon,
getText(R.string.ticker_text), System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);
notification.setLatestEventInfo(this,getText(R.string.notification_title),get
Text(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);

```

### iii. Defining custom services

#### a. Implementation and declaration

A service needs to be declared in the `AndroidManifest.xml` file and the implementing class must extend the `Service` class or one of its subclasses.

The following code shows an example for a service declaration and its implementation.

```

<service
    android:name="MyService"
    android:icon="@drawable/icon"
    android:label="@string/service_name"
</service>

public class MyService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //TODO do something useful
        return Service.START_NOT_STICKY;
    }
    @Override
    public IBinder onBind(Intent intent) {
        //TODO for communication return IBinder implementation
        return null;
    }
}

```

**b. Start a service**

An Android component (service, receiver, activity) can trigger the execution of a service via the `startService(intent)` method.

```
// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);
// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
context.startService(i);
```

A service can also be started via the `bindService()` method call. This method allows to communicate directly with the service. We discuss that later.

**c. Service start process and execution**

If the `startService(intent)` method is called and the service is not yet running, the service object is created and the `onCreate()` method of the service is called.

Once the service is started, the `onStartCommand(intent)` method in the service is called. It passes in the `Intent` object from the `startService(intent)` call.

If `startService(intent)` is called while the service is running, its `onStartCommand()` is also called. Therefore your service needs to be prepared that `onStartCommand()` can be called several times.

A service is only started once, no matter how often you call the `startService()` method.

**d. Service restart behavior**

In its `onStartCommand()` method call, the service returns an `int` which

defines its restart behavior in case the service gets terminated by the Android platform. You can use the constants, the most common options are described by the following table.

Table 1. Restart options

Option	Description
<code>Service.START_STICKY</code>	Service is restarted if it gets terminated. Intent data passed to the <code>onStartCommand</code> method is null. Used for services which manages their own state and do not depend on the Intent data.
<code>Service.START_NOT_STICKY</code>	Service is not restarted. Used for services which are periodically triggered anyway. The service is only restarted if the runtime has pending <code>startService()</code> calls since the service termination.
<code>Service.START_REDELIVER_INTENT</code>	Similar to <code>Service.START_STICKY</code> but the original Intent is re-delivered to the <code>onStartCommand</code> method.

e. Stopping a service

To stop a service via the `stopService()` method. No matter how frequently you called the `startService(intent)` method, one call to the `stopService()` method stops the service.

A service can terminate itself by calling the `stopSelf()` method. This is typically done if the service finishes its work.

f. IntentServices for one time tasks

You can also extend the `IntentService` class for your service implementation.

The `IntentService` is used to perform a certain task in the background.



Once done, the instance of IntentService terminates itself automatically. An example for its usage would be downloading certain resources from the internet.

The IntentService class offers the onHandleIntent() method which will be asynchronously called by the Android system.

## 2. Broadcast Receivers:

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents –

- Creating the Broadcast Receiver.
  - Registering Broadcast Receiver.
- i. Creating the Broadcast Receiver: A broadcast receiver is implemented as a subclass of BroadcastReceiver class and overriding the onReceive() method where each message is received as a Intent object parameter.

```
public class MyReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(context, "Intent Detected.",  
            Toast.LENGTH_LONG).show();  
    }  
}
```

- ii. Registering Broadcast Receiver: An application listens for specific broadcast intents by registering a broadcast receiver in AndroidManifest.xml file. Consider we are going to register MyReceiver for system generated event ACTION\_BOOT\_COMPLETED which is fired by the system once the

Android system has completed the boot process.

```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <receiver android:name="MyReceiver">
    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED">
    </action>
    </intent-filter>
  </receiver>
</application>
```

Whenever the Android device gets booted, it will be intercepted by BroadcastReceiver. MyReceiver and implemented logic inside onReceive() will be executed.

Follow the following steps to create BroadcastReceiver to intercept custom intent. Once familiar with custom intent, then it is easy to program your application to intercept system generated intents.

Step 1: create an Android application and name it as MyApplication under your package.

Step2 : Modify main activity file MainActivity.java to add broadcastIntent() method

Step 3: Create a new java file called MyReceiver.java under the package to define a BroadcastReceiver.

Step 4: An application can handle one or more custom and system intents without any restrictions. Every intent you want to intercept must be registered in your AndroidManifest.xml file using <receiver.../> tag

Step5: Modify the default content of res/layout/activity\_main.xml file to include a button to broadcast intent.

Step 6 : No need to modify the string file, Android studio take care of string.xml file.

Step 7: Run the application to launch Android emulator and verify the result of the changes done in the application.

## 2.1 Pending Intent

A pending intent is a token that you give to another application. For example, the notification manager, alarm manager or other 3rd party applications). This allows the other application to restore the permissions of your application to execute a predefined piece of code.

To perform a broadcast via a pending intent, get a `PendingIntent` via the `getBroadcast()` method of the `PendingIntent` class. To perform an activity via a pending intent, you receive the activity via `PendingIntent.getActivity()`.

## 3. Communication with services

### i. Options for communication

There are several possibilities for a communication between an activity and a service. The following description discusses the possible approaches.

### ii. Using Intent data

In a simple scenario no direct communication is required. The service receives the intent data from the starting Android component and performs its work. No notification is necessary. For example, in case the service updates a content provider, the activity is notified by the content provider and no extra step in the service is necessary. This approach works for local and services running in their own process.

### iii. Using receiver

You can use broadcasts and registered receivers for the communication. For example, your activity can register a broadcast receiver for an event and the service sends out corresponding events. This is a very typical scenario, in which the service need to signal to the activity that his processing has finished.

This approach works for local and services running in their own process. Android provides the `LocalBroadcastManager` class in the support library v4. This is a helper class to register for and send broadcasts of Intents to local objects within your process. This approach improves security as the broadcast events are only visible within your process and is faster than using standard events.

#### **iv. Activity binding to local service:**

If the service is started in the same process as the activity, the activity can directly bind to the service. This is a relatively simple and efficient way to communicate and recommended for activities which need to have a fast communication layer with the service. This approach works for local services.

#### **v. Handler and ResultReceiver or Messenger**

If the service should be communicating back to the activity, it can receive an object of type `Messenger` via the Intent data it receives from the activity. If the `Messenger` is bound to a `Handler` in the activity, the service can send objects of type `Message` to the activity.

A `Messenger` is parcelable, which means it can be passed to another process and you can use this object to send Messages to the `Handler` in the activity. `Messenger` also provides the method `getBinder()` which allows passing a `Messenger` to the activity. The activity can therefore send Messages to the service. This approach works for local services running in their own process.

### **Lab Exercise:**

1. Develop an android application to use a service to download a file from the internet, based on click of the button from an activity. Once download is complete, the service has to notify the activity via a broadcast receiver that the download is complete.

2. Develop an android application to define a broadcast receiver which listens to telephone state changes. If the phone receives a phone call, then the receiver will be notified and log a message.

[OBSERVATION SPACE – LAB 10]



**LAB NO. 11****Date:****ANDROID FOR CAMERA, BLUETOOTH AND WI-FI****Objectives:**

- 1.Understand and demonstrate to launch the camera on PC
2. Demonstrate to transfer files via bluetooth
- 3.Demonstrate to transfer files via wi-fi

**Introduction:**

In Android, Camera is a hardware device that allows capturing pictures and videos in your applications. Bluetooth is a way to send or receive data between two different devices. Android platform includes support for the Bluetooth framework that allows a device to wirelessly exchange data with other Bluetooth devices. Android allows applications to access to view the access the state of the wireless connections at very low level. Application can access almost all the information of a wifi connection

**PartA:Camera**

You will use `MediaStore.ACTION_IMAGE_CAPTURE` to launch an existing camera application installed on your phone. Its syntax is given below

Intent	intent	=	new
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);			

Now you will use the function *startActivityForResult()* to launch this activity and wait for its result. Its syntax is given below

<code>startActivityForResult(intent,0)</code>
---

This method has been defined in the **activity** class. We are calling it from main activity.

The result can be obtained by overriding the function *onActivityResult*.

Here is an example that shows how to launch the existing camera application to capture an image and display the result in the form of bitmap.To experiment with this example , you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Android studio IDE to create an Android application and name it as Camera under a com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add intent code to launch the Camera.
3	Modify layout XML file res/layout/activity_main.xml
4	Add the Camera permission and 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.Manifest;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Settings;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;

public class MainActivity extends AppCompatActivity {
    public static final int MY_PERMISSIONS_REQUEST_CAMERA = 100;
    public static final String ALLOW_KEY = "ALLOWED";
    public static final String CAMERA_PREF = "camera_pref";

    @Override
```



```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
        if (getFromPref(this, ALLOW_KEY)) {
            showSettingsAlert();
        } else if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.CAMERA)

            != PackageManager.PERMISSION_GRANTED) {

                // Should we show an explanation?
                if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                    Manifest.permission.CAMERA)) {
                    showAlert();
                } else {
                    // No explanation needed, we can request the permission.
                    ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.CAMERA},
                        MY_PERMISSIONS_REQUEST_CAMERA);
                }
            }
        } else {
            openCamera();
        }
    }

    public static void saveToPreferences(Context context, String key, Boolean allowed) {
        SharedPreferences myPrefs = context.getSharedPreferences(CAMERA_PREF,
            Context.MODE_PRIVATE);
        SharedPreferences.Editor prefsEditor = myPrefs.edit();
        prefsEditor.putBoolean(key, allowed);
        prefsEditor.commit();
    }

    public static Boolean getFromPref(Context context, String key) {
        SharedPreferences myPrefs = context.getSharedPreferences(CAMERA_PREF,
            Context.MODE_PRIVATE);
        return (myPrefs.getBoolean(key, false));
    }

```

```

}

private void showAlert() {
    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this).create();
    alertDialog.setTitle("Alert");
    alertDialog.setMessage("App needs to access the Camera.");

    alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "DONT ALLOW",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                finish();
            }
        });

    alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "ALLOW",
        new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                ActivityCompat.requestPermissions(MainActivity.this,
                    new String[]{Manifest.permission.CAMERA},
                    MY_PERMISSIONS_REQUEST_CAMERA);
            }
        });
    alertDialog.show();
}

private void showSettingsAlert() {
    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.this).create();
    alertDialog.setTitle("Alert");
    alertDialog.setMessage("App needs to access the Camera.");
    alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "DONT ALLOW",
        new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                //finish();
            }
        });
}

```

```

    });

    alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "SETTINGS",
        new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
                startInstalledAppDetailsActivity(MainActivity.this);
            }
        });

    alertDialog.show();
}
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[]
grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_CAMERA: {
            for (int i = 0, len = permissions.length; i < len; i++) {
                String permission = permissions[i];

                if (grantResults[i] == PackageManager.PERMISSION_DENIED) {
                    boolean
                        showRationale =
                            ActivityCompat.shouldShowRequestPermissionRationale(
                                this, permission);

                    if (showRationale) {
                        showAlert();
                    } else if (!showRationale) {
                        // user denied flagging NEVER ASK AGAIN
                        // you can either enable some fall back,
                        // disable features of your app
                        // or open another dialog explaining
                        // again the permission and directing to
                        // the app setting
                        saveToPreferences(MainActivity.this, ALLOW_KEY, true);
                    }
                }
            }
        }
    }
}

```

```

    }

    // other 'case' lines to check for other
    // permissions this app might request
    }
}

@Override
protected void onResume() {
    super.onResume();
}

public static void startInstalledAppDetailsActivity(final Activity context) {
    if (context == null) {
        return;
    }

    final Intent i = new Intent();
    i.setAction(Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
    i.addCategory(Intent.CATEGORY_DEFAULT);
    i.setData(Uri.parse("package:" + context.getPackageName()));
    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    i.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
    i.addFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
    context.startActivity(i);
}

private void openCamera() {

    Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
    startActivity(intent);
}
}

```

Following will be the content of **res/layout/activity\_main.xml** file—

```

<?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: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">
</RelativeLayout>

```

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

```

<resources>
    <string name="app_name">My Application</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.sairamkrishna.myapplication" >
    <uses-permission android:name="android.permission.CAMERA" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.sairamkrishna.myapplication.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 your 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 tool bar. Before starting your application, Android studio will display the window as in Figure 11.1 to select an option where you want to run your Android application.

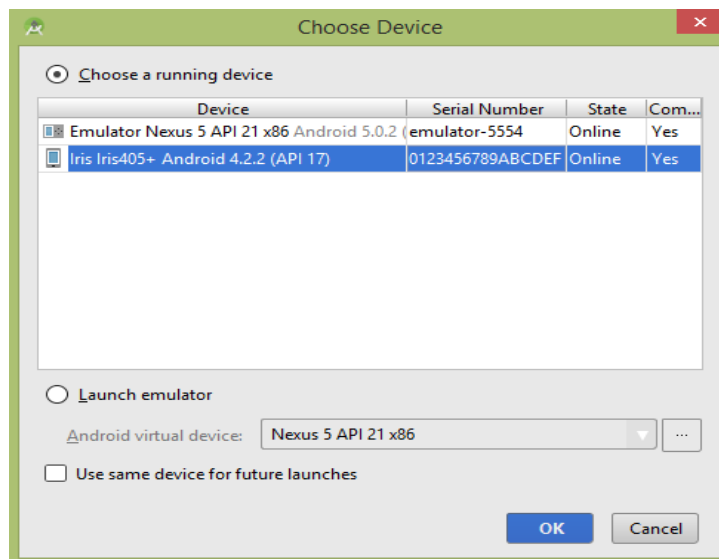


Figure 11.1: Option window to run your Android application

Select your mobile device as an option and then check your mobile device which will open the camera and display the screen.

## PartB: Bluetooth for Android

Android provides Bluetooth API to perform these different operations.

- Scan for other Bluetooth devices
- Get a list of paired devices
- Connect to other devices through service discovery

Android provides `BluetoothAdapter` class to communicate with Bluetooth. Create an object of this class by calling the static method `getDefaultAdapter()`. Its syntax is given below.

```
private BluetoothAdapter BA;  
BA = BluetoothAdapter.getDefaultAdapter();
```

In order to enable the Bluetooth of your device, call the intent with the following Bluetooth constant `ACTION_REQUEST_ENABLE`. Its syntax is.

```
Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);  
startActivityForResult(turnOn, 0);
```

Once you enable the Bluetooth, you can get a list of paired devices by calling `getBondedDevices()` method. It returns a set of Bluetooth devices. Its syntax is.

```
private Set<BluetoothDevice>pairedDevices;  
pairedDevices = BA.getBondedDevices();
```

### Let us learn through an example

To experiment with this example, you need to run this on an actual device. Follow these steps:

1. You will use Android studio to create an Android application with package `com.example.xxxx.myapplication`.
2. Modify `src/MainActivity.java` file to add the code
3. Modify layout XML file `res/layout/activity_main.xml` add any GUI component if required.
4. Modify `AndroidManifest.xml` to add necessary permissions.

5. Run the application and choose a running android device and install the application on it and verify the results.

Here is the content of **src/MainActivity.java**

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

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListView;

import android.widget.Toast;
import java.util.ArrayList;
import java.util.Set;

public class MainActivity extends Activity {
    Button b1,b2,b3,b4;
    private BluetoothAdapter BA;
    private Set<BluetoothDevice>pairedDevices;
    ListView lv;

    @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);

        BA = BluetoothAdapter.getDefaultAdapter();
```



```

    lv = (ListView)findViewById(R.id.listView);
}

public void on(View v){
    if (!BA.isEnabled()) {
        Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(turnOn, 0);
        Toast.makeText(getApplicationContext(), "Turned
on",Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(getApplicationContext(), "Already on",
Toast.LENGTH_LONG).show();
    }
}

public void off(View v){
    BA.disable();
    Toast.makeText(getApplicationContext(), "Turned off"
,Toast.LENGTH_LONG).show();
}

public void visible(View v){
    Intent getVisible = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    startActivityForResult(getVisible, 0);
}

public void list(View v){
    pairedDevices = BA.getBondedDevices();

    ArrayList list = new ArrayList();

    for(BluetoothDevice bt : pairedDevices) list.add(bt.getName());
    Toast.makeText(getApplicationContext(), "Showing Paired
Devices",Toast.LENGTH_SHORT).show();

    final ArrayAdapter adapter = new
ArrayAdapter(this,android.R.layout.simple_list_item_1, list);

```

```

        lv.setAdapter(adapter);
    }
}

```

Here is the content of **activity\_main.xml**

```

<?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: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">

    <TextView android:text="Bluetooth Example"
        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:src="@drawable/abc"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:theme="@style/Base.TextAppearance.AppCompat" />

```

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Turn On"
    android:id="@+id/button"
    android:layout_below="@+id/imageView"
    android:layout_toStartOf="@+id/imageView"
    android:layout_toLeftOf="@+id/imageView"
    android:clickable="true"
    android:onClick="on" />

```

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Get visible"
    android:onClick="visible"
    android:id="@+id/button2"
    android:layout_alignBottom="@+id/button"
    android:layout_centerHorizontal="true" />

```

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="List devices"
    android:onClick="list"
    android:id="@+id/button3"
    android:layout_below="@+id/imageView"
    android:layout_toRightOf="@+id/imageView"
    android:layout_toEndOf="@+id/imageView" />

```

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

```

```

    android:text="turn off"
    android:onClick="off"
    android:id="@+id/button4"
    android:layout_below="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

```

```

<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/button"
    android:layout_alignStart="@+id/button"
    android:layout_below="@+id/textView2" />

```

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Paired devices:"
    android:id="@+id/textView2"
    android:textColor="#ff34ff06"
    android:textSize="25dp"
    android:layout_below="@+id/button4"
    android:layout_alignLeft="@+id/listView"
    android:layout_alignStart="@+id/listView" />

```

```

</RelativeLayout>

```

Here is the content of **Strings.xml**

```

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

```

Here is the content of **AndroidManifest.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sairamkrishna.myapplication" >
    <uses-permission android:name="android.permission.BLUETOOTH"/>

```

```

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

<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 your application. I assume you have connected your actual Android Mobile device with your computer.

1.To run the app from Android studio, open one of your project's activity files and click Run  icon from the tool bar.If your Bluetooth will not be turned on then, it will ask your permission to enable the Bluetooth as shown in Figure 11.2.

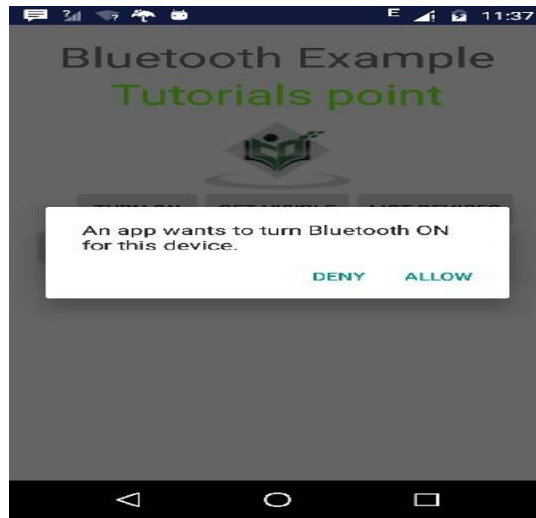


Figure 11.2: Permission window to enable the Bluetooth

2. Now just select the Get Visible button to turn on your visibility. A screen would appear asking your permission to turn on discovery for 120 seconds as shown in Figure 11.3.
3. Now just select the List Devices option. It will list down the paired devices in the list view.
4. Now just select the Turn off button to switch off the Bluetooth. A message would appear when you switch off the bluetooth indicating the successful switching off of Bluetooth as shown in Figure 11.4.

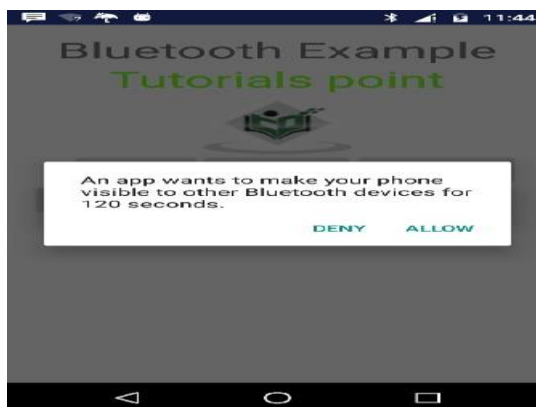


Figure 11.3: On using Visible button to turn on your visibility



Figure 11.4: On using Turn off button to switch off the Bluetooth

## C: Wi-fi for Android

The information that an application can access includes connected network's link speed, IP address, negotiation state, other networks information. Applications can also scan, add, save, terminate and initiate Wi-Fi connections.

Android provides **WifiManager** API to manage all aspects of WIFI connectivity. We can instantiate this class by calling **getSystemService** method. Its syntax is given below –

```
WifiManager mainWifiObj;
mainWifiObj = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

In order to scan a list of wireless networks, you also need to register your BroadcastReceiver. It can be registered using **registerReceiver** method with argument of your receiver class object. Its syntax is given below –

```
class WifiScanReceiver extends BroadcastReceiver {
    public void onReceive(Context c, Intent intent) {
    }
}

WifiScanReceiver wifiReciever = new WifiScanReceiver();
registerReceiver(wifiReciever,
IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
```

new

The wifi scan can be start by calling the **startScan** method of the WifiManager class. This method returns a list of ScanResult objects. You can access any object by calling the **get** method of list. Its syntax is given below –

```
List<ScanResult> wifiScanList = mainWifiObj.getScanResults();
String data = wifiScanList.get(0).toString();
```

Here is an example demonstrating the use of WIFI. It creates a basic application that open your wifi and close your wifi. To experiment with this example, you need to run this on an actual device on which wifi is turned on.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add WebView code.
3	Modify the res/layout/activity_main to add respective XML components
4	Modify the AndroidManifest.xml to add the necessary permissions
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.projectname.myapplication;

import android.net.wifi.WifiManager;
import android.os.Bundle;
```



```

import android.app.Activity;
import android.content.Context;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
    Button enableButton,disableButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        enableButton=(Button)findViewById(R.id.button1);
        disableButton=(Button)findViewById(R.id.button2);

        enableButton.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                WifiManager wifi = (WifiManager)
getSystemService(Context.WIFI_SERVICE);
                wifi.setWifiEnabled(true);
            }
        });

        disableButton.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                WifiManager wifi = (WifiManager)
getSystemService(Context.WIFI_SERVICE);
                wifi.setWifiEnabled(false);
            }
        });
    }
}

```

Following is the modified content of the xml **res/layout/activity\_main.xml**.

```

<?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: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">

```

```

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abc"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />

```

```

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="76dp"
    android:text="Enable Wifi"
    android:layout_centerVertical="true"
    android:layout_alignEnd="@+id/imageView" />

```

```

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Disable Wifi"
    android:layout_marginBottom="93dp"
    android:layout_alignParentBottom="true"
    android:layout_alignStart="@+id/imageView" />

```

```

</RelativeLayout>

```

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


    <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 your application. Connect to 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 will display the window to select an option where you want to run your Android application as in Figure 11.5.

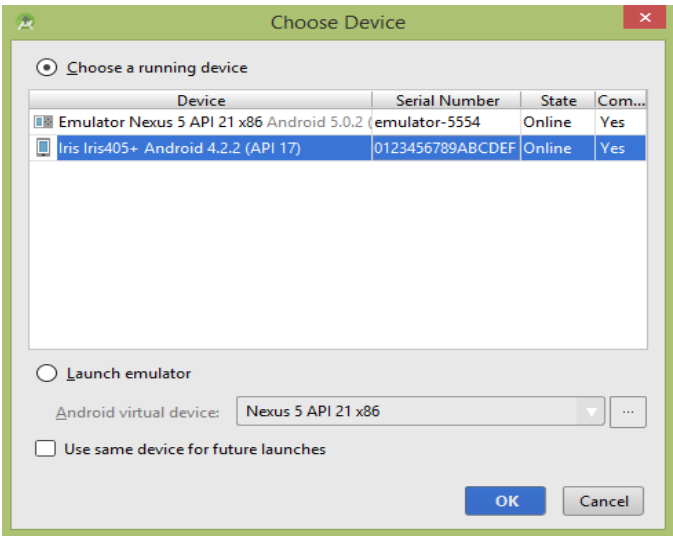


Figure 11.5: Window to select an option where to run your Android application

Select your mobile device as an option and It will shows the window as in Figure 11.6.

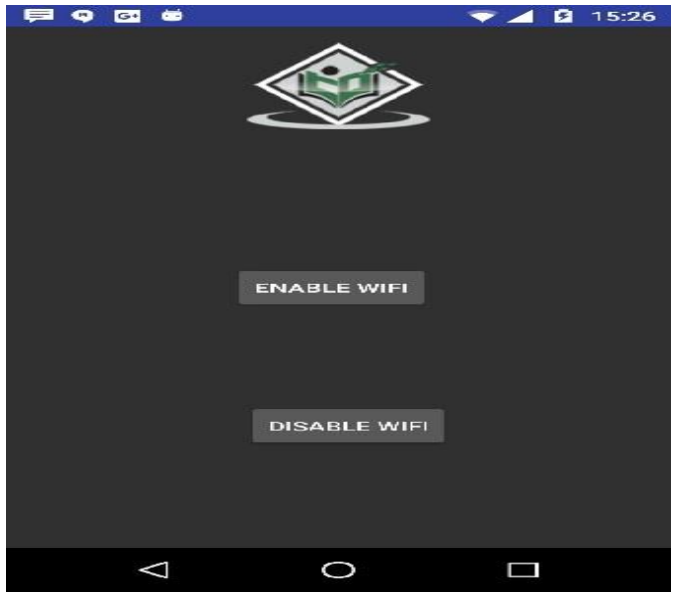


Figure 11. 6: Enable/Disable window

**Lab Exercise:**

- 1 .Develop an android application to transfer picture from mobile phone to your email via Bluetooth
- 2 . Develop an android application to transfer picture from mobile phone to your email via email.

[OBSERVATION SPACE – LAB 11]

**LAB 12**

**Date:**

**PROJECT IMPLEMENTATION & TESTING**

**Objectives**

- To implement a mini project employing all the concepts learnt.

**Lab exercises**

1. Give the implementation details regarding your project.
-

[OBSERVATION SPACE – LAB 12]

[OBSERVATION SPACE – LAB 12]



## REFERENCES

1. Zheng, Pei, and Lionel Ni. Smart phone and next generation mobile computing. Morgan Kaufmann, 2010.
2. Hermes, Daniel. Xamarin Mobile Application Development: Cross-Platform C# and Xamarin. Forms Fundamentals. Apress, 2015.
3. Mark, Dave, and Jeff LaMarche. "Beginning iPhone Development." Exploring the iPhone SDK, APRESS, Berkeley (2009).
4. Lee, Henry, and Eugene Chuvyrov. Beginning Windows Phone App Development. Apress, 2012.