Chapter 1

Mobile Programming

Introduction

- **Mobile Devices** need some type of OS to run its services
- **OS**:- a software platform on top of which other apps can run on
- **Modern OS** combines a **PC** with **other mobile specific features** like ***Bluetooth, GPS, speech recognition, video camera, infrared*** etc, so mobile OS had to grow to support all these features. This ***results the growth of technology***
- Some of the mobile operating systems are:
  - **iOS**:- Developed by Apple inc. and distributed exclusively for Apple Hardware
  - **Windows Phone**:- A proprietary software Smartphone OS developed by Microsoft
  - **Android**:- Developed by Open Handset Alliance (OHA), Led by Google
  - Other OSs are: **Symbian, BlackBerry, webOS**

What is Android?

- An open source mobile OS developed by Google, based on the **Linux kernel**
- **It** is designed primarily for **touch screen mobile devices** such as smart phones, tablets, mp4 players, TVs etc.
- It is a **software stack** for mobile devices that includes an **operating system, middleware and key applications**
- **Android SDK** provides the tools and **APIs** necessary to develop applications

History of Android

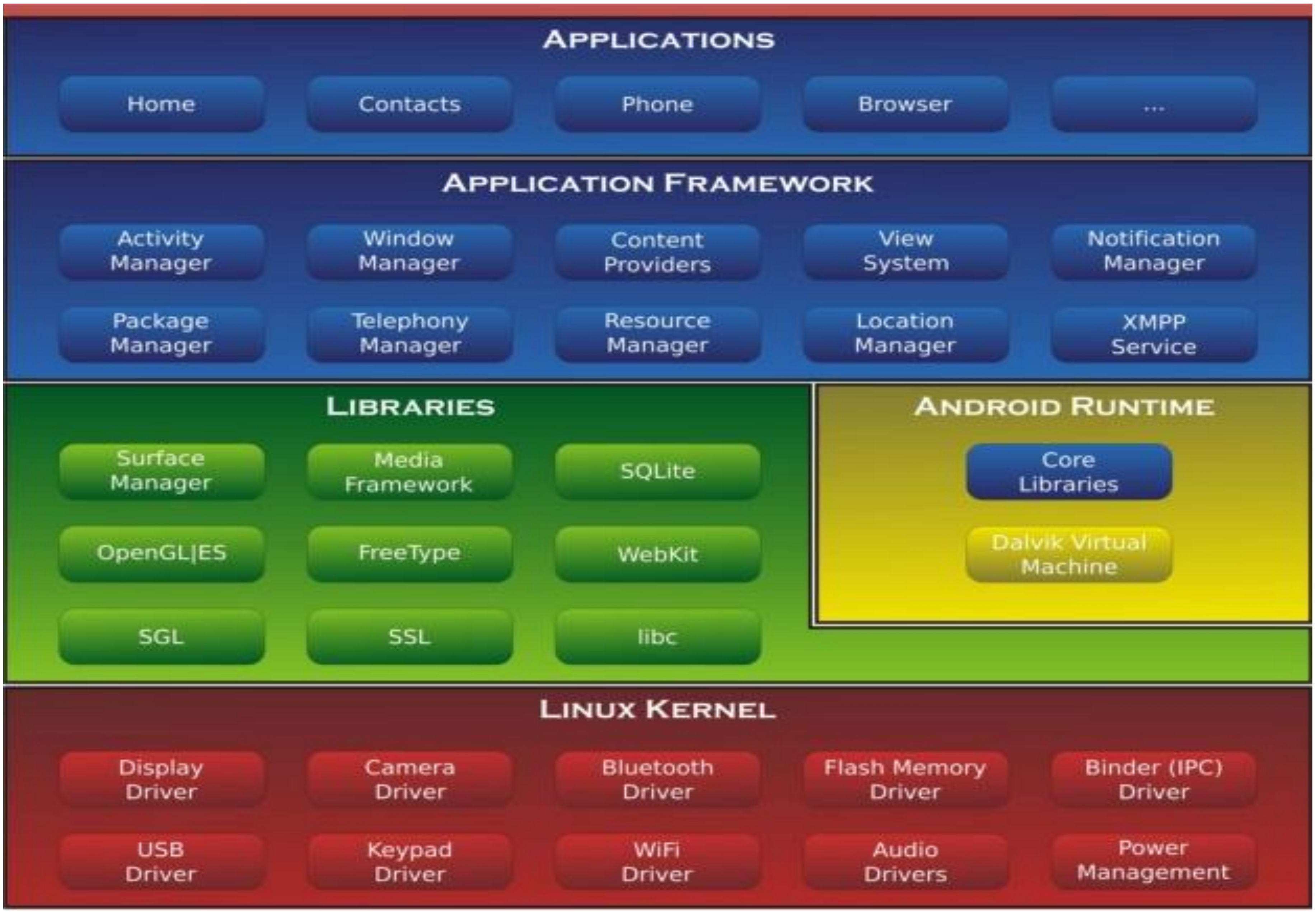
- Initially, **Andy Rubin** founded Android Incorporation in Palo Alto, California, United States in **October, 2003**.
- In **17th August 2005**, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
- Originally intended for **camera** but shifted to **smart phones** later because of **low market for camera only**.
- **Android** is the **nick name of Andy Rubin** given by coworkers because of his love to robots.
- Google formed **Open Handset Alliance** (OHA) on 5th November 2007.
- **OHA** is a group of 84 technology and mobile companies who have come together to accelerate innovation.
- **HTC Dream** was the first Commercial Smartphone running Android (T-Mobile 2008)

Android Versions

Date	Version	Code name	API Level
September 23, 2008	1.0	No codename	1
February 9, 2009	1.1	Petit Four	2
April 27, 2009	1.5	Cupcake	3
September 15, 2009	1.6	Donut	4
October 26, 2009	2.0 - 2.1	Éclair	5 - 7
May 20, 2010	2.2 – 2.2.3	Froyo	8
December 6, 2010	2.3 – 2.3.7	Gingerbread	9 - 10
February 22, 2011	3.0 – 3.2.6	Honeycomb	11 - 13
October 18, 2011	4.0 – 4.0.4	Ice Cream Sandwich	14 - 15
July 9, 2012	4.1 - 4.3	Jelly Bean	16 - 18
October 31, 2013	4.4 - 4.4.4	KitKat	19-20
November 12, 2014	5.0 - 5.1.1	Lollipop	21-22
October 5, 2015	6.0 - 6.0.1	Marshmallow	23
August 22, 2016	7.0 -7.1.2	Nougat	24-25
August 21, 2017	8.0	Oreo	26-27
August 6, 2018	9.0	Pie	28

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

Android Architecture



Linux Kernel

- Is the OS.
- It is responsible for:
  - Memory Management:- allocate and free memory for files
  - Power Management:- Provide power for various devices such as Bluetooth, camera etc.
  - Resource Management:- provide resource for each process and making it versatile.
  - Driver Management:- handle installations of various drivers
- Go to "About phone" or "About tablet" in Android's settings to find out the kernel version

Middleware

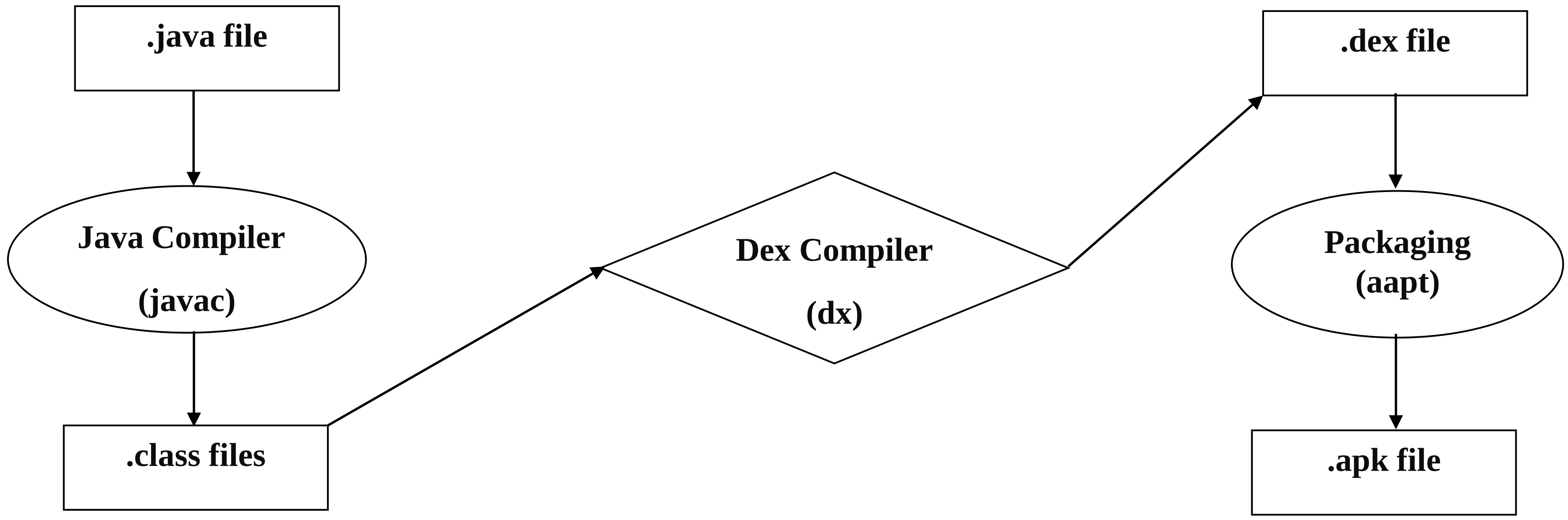
- **Libraries:-**
  - Used for interacting with **low level media components** (programs written in some other languages like C, C++, Assembly language which are specific to hardware and operating system **or native application**).
  - Example: **WebKit** library is responsible for browser support, **SQLite** is for database, **FreeType** for font support, **Media** for playing and recording audio and video formats.
- **Application 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

- **Android Runtimes:-**
  - includes **Dalvik Virtual Machine(DVM)** and **Core Libraries** 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**.

Applications

- is the **top most layer of Android Architecture**.
- All applications using **android framework *uses android runtime and libraries***, while **android runtime and native libraries** are using ***Linux Kernel***.

Dalvik Virtual Machine(DVM)



- Android applications can be developed using **languages such as C++, C#, Java** etc. But the official language is **Java**.
- **JDK** compiles a **Java Code** and Produces a **.class** file (byte code)
- In android **Dex** compiler takes **.class** file and produce **.dex** file, **DVM** takes **.dex** file and generate a **machine code**.
- **DVM** was used as **runtime in Android v 2.2 – 5.0** and then replaced by **Android Run Time**. ART was introduced in the **4.4 release Kitkat**

Why did Google switched from DVM to ART?

- **DVM**
  - based on JIT compilation i.e. that each **time you run an app, the part of the code required for its execution** is going to be translated (**compiled**) to machine code at that moment
  - Requires smaller memory but more CPU overhead
- **ART**
  - compiles the intermediate language, Dalvik bytecode, into a system independent binary **during installation (once),** thus removing the lag that we see when we open an app on our device
  - **Less CPU usage** and in turn results **less battery drain**
  - **Memory** is much **cheaper now**

Android Application Components

- **Activities:** An activity in Android represents a **single screen with which a user can interact with**. An application can have one or more activities but run independently.
- **Services:** Long running background process without any need of user interaction is known as Service.
- **Broadcast Receiver:** handle **communication between Android OS and applications**. Example:-battery is low notification, the sign of earphone as soon as you plug a headset
- **Content Provider:** content providers are used to share data between the applications. In android the data cannot be shared directly between the two applications.
- **Additional Components**: ***Fragments,Views*** , ***Layouts***, ***Intent, Resources, Manifest*** etc

Getting Started

- In order to write android application we need:
  - Java Development Kit (JDK) and JRE (Java Runtime Environment)
  - IDE (Android Studio or Eclipse or Netbeans)
    - Android Studio includes SDK (Software Development Kit)
    - Android Emulator

Building your first app

- File>New>Android Application Project
- Fill Application Name and package name
- Select the Phone in Target SDK and proceed
- Select Activity and Finish

Running Application

- You can run your application on Android Emulator or Real Device
- To run the application via Android Emulator
  - First create an Emulator in AVD manager and Select Run>Run
- To run the application via a real device
  - Go to ***Settings>General>About Phone***.
  - Scroll and select Software ***Information>Build Number.***
  - Quickly tap on "Build Number" seven times and you will see the message "You are now a developer".
  - Connect your mobile device and Select Run>Run

Android Project Structure

- Android Studio creates the necessary structure for all your files and makes them visible in the project window on the left side of the IDE
- **App:**
  - is the default module name for your project. You can have one or more modules.
  - Module is a **collection of source files and build settings**. It provides **essential files and some code templates**
- **Manifest.xml**:
  - describes the **fundamental characteristics of an app** and each of its **components**.
  - It works as an **interface between your android OS and your application.**

- **Android Java:**
  - contains the Java source code files separated by package names and JUnit test code.
  - All the Activity classes are stored inside java folder
- **Android Resource:**
  - contains all non-code resources, such as xml layouts, UI strings, Styles, Colors, bitmap images etc

Chapter 2

Android Layout and UI widgets

Views and View Groups

- **View:**
  - **a**nything which **occupies a rectangular area** on the screen
  - is **responsible for drawing and event handling**
  - every UI (Button, EditText, CheckBox etc) element is subclass of view (**android.view.View)**
- **ViewGroup:**
  - is a subclass of View (*android.view.View)*
  - provides **invisible container** that hold other **views** or other **ViewGroups** and **define their layout** properties
  - It is the **base class for layouts and views containers**.
  - It contains commonly used subclasses like **RelativeLayout, LinearLayout, ListView, GridView** etc

Layouts and its types

- **Layouts** are subclasses of **ViewGroup** which **specifies how a view should be arranged** inside the **viewgroups**
- The standard Layouts are:
  - **LinearLayout**: arranges its children in a single **column** or a single row.
  - **RelativeLayout**: the positions of the children can be described in relation to each other or to the parent.
  - **FrameLayout**: It is used to block out an area on the screen to display a single item.
  - **TableLayout**: arranges its children into rows and columns.

Linear Layout

- is a view group that aligns all children in a single direction, vertically or horizontally
- Layout direction is specified: ***android:orientation=”horizontal”*** ***android:orientation=”vertically”***
- ***Example:***

```
<LinearLayout . . .
   android:orientation="horizontal">

     ...
   <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
   ...
</LinearLayout>
```

RelativeLayout

- lays out elements based on their relationships with one another, and with the parent container.
- These properties will layout elements relative to the parent:
  - ***android:layout_alignParentBottom***-places the element on the bottom of the container.
  - ***android:layout_alignParentLeft-***places the element on the left side of the container.
  - ***android:layout_alignParentRight-***places the element on the right side of the container.
  - ***android:layout_alignParentTop***-places the element on the top of the container.
  - ***android:layout_centerHorizontal***-centers the elements horizontally within its parent container.
  - ***android:layout_centerInParent***- centers the elements both horizontally and vertically within its parent container.
  - ***android:layout_centerVertical***- centers the elements vertically within its parent container.

- **Relative to other element**: Given the element's Id: "*@id/xxxxx*"
  - ***android:layout_above***- places an element above the specified element.
  - ***android:layout_below***- places an element below the specified element.
  - ***android:layout_toLeftOf***- places an element to the left of the specified element.
  - ***android:layout_toRightOf***- places an element to the right of the specified element.
  - ***android:layout_alignBaseline***- aligns baseline of the new element with baseline of the specified element.
  - ***android:layout_alignBottom***- aligns bottom of the new element in with bottom of the specified element.
  - ***android:layout_alignLeft***- aligns left edge of the new element with left edge of the specified element.
  - ***android:layout_alignRight-*** aligns right edge of the new element with right edge of the specified element.
  - ***android:layout_alignTop***- places top of the new element in alignment with the top of the specified element.

- Example

```
<RelativeLayout ...>
  ...
  <TextView
    ...
    android:id="@+id/textview"
    android:layout_alignParentTop="true"
    android:layout_centerInParent="true"/>
  <Button
    ...
    android:id="@+id/button1"
    android:layout_below="@+id/textview"
    android:layout_alignParentStart="true"/>
    ...
</RelativeLayout>
```

Table Layout

- organizes content into rows and columns

Example:

```
<TableLayout …
  <TableRow>
        <TextView   ..      android:text="@string/hello_world" />
   </TableRow>
  <TableRow>
    <TextView  ..     android:text="@string/your_name" />
     <EditText   …  android:inputType="text”  android:text="@string/nametxt" />
  </TableRow>
  <TableRow>
     <Button      …      android:text="@string/submit" />
     <Button  …     android:text="@string/cancel" />
  </TableRow>

</TableLayout>
```

Frame Layout

- designed to **display a single item** at a time
- You can have **multiple elements within a FrameLayout** but each **element will be positioned based on the top left of the screen**.
- Elements that **overlap will be displayed overlapping**
- useful when **elements are hidden and displayed programmatically**
- use the attribute ***android:visibility*** in the XML to hide specific elements.
  - **Visibility values are: visible, invisible** (does not display but still takes up space in the layout) **and gone** (does not display and does not take space in the layout)**.**
- You can even call ***setVisibility*** method from the code

- **Example**:

```
<FrameLayout …>
   …
  <Button
     …
    android:id="@+id/button"
    android:text="@string/button"/>
  <TextView
    …
    android:Text="@string/largetext"
    android:id="@+id/textview"
    android:layout_gravity="center"
    android:textIsSelectable="true"/>
   …
</FrameLayout>
```

Layout Attributes

- Each layout has a set of attributes that define its visual properties.
- Some of the common attributes are:
  - ***android:id*** - uniquely identifies the view.
  - ***android:layout_width or layout_height*** – defines width and height of the layout
  - ***android:layout_marginTop , Bottom,Lefft, Right-*** defines extra space arroung the layout.
  - ***android:layout_gravity***- specifies how child Views are positioned.
  - ***android:layout_weight***- specifies how much of the extra space in the layout should be allocated to the View.
  - ***android:layout_x or y*** - This specifies the x or y -coordinate of the layout.
  - ***android:paddingLeft, Right, Top, Bottom*** - defines the padding filled for the layout.

Width and Height Measurements

- dp (Density-independent Pixels),
- sp (Scale-independent Pixels),
- pt (Points which is 1/72 of an inch),
- px (Pixels),
- mm (Millimeters) and finally in (inches)
- In most cases width and height of a layout are specified in
  - ***android:layout_width=wrap_content*** - tells your view to size itself to the dimensions required by its content.
  - ***android:layout_width=fill_parent*** - tells your view to become as big as its parent view.
- Gravity plays a great role in positioning the view object. The values of android:layout_gravity property are:
  - Top, bottom, right, left, center, start, end, center_vertical, center_horizontal, fill_vertical, fill_horizontal etc

Android UI widgets

- input controls which are used to build a user interface (views)
- It includes Button, TextView, EditView, ImageView, CheckBox, RadioButton, ProgressBar, Spinner, TimePicker etc.

Button

- A GUI component that is sensible to taps (clicks) by the user
- represented by the Android class ***android.widget.button***
- It can be created in two types:
  - Button with Text
  - Button with Images
- Button instance can be inserted into GUI either through
  - .xml file or
  - programmatically in .java file.

- **In xml file**

```
<Button
       android:id="@+id/btn1 "
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Display"
/>
```

- **Programmatically in java file**

```
public class MainActivity extends Activity {
   @Override
  protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
      Button button=new Button(this);//creating instance
       button.setText("Display");//setting a text
  RelativeLayout relativeLayout=(RelativeLayout) findViewById(R.id.rootlayout); //
   creating a viewGroup layout and find a layout
relativeLayout.addView(button);//add button to the layout
  } }
```

Adding a click event to the button

- Use *android:onClick* attribute to specify the event type in the xml file for the button and set the method as its value
- ***Example***: *android:onClick="btnOnclick()"*
- Go to the java button and define the method. Example:

```
protected void btnClick(View view)
{
   Toast.makeText(getBaseContext(), "Button
   Clicked!",Toast.LENGTH_SHORT).show();
}
```

- The above method will be invoked when the user clicks on the button and will display "Button Clicked!" message upon click

Toast

- Is a notification message that pop up for a specific duration
- Uses ***android.widget.Toast*** class which is the subclass of ***java.lang.Object*** class
- Example:

```
Toast toast=Toast.makeText(getApplicationContext(), “Button
   Clicked!”,Toast.LENGTH_SHORT);
toast.setGravity(Gravity.AXIS_PULL_AFTER,0,0);
toast.show();
```

- The method takes three parameters:
  - an application context, the text message and the duration(Toast.LENGTH_SHORT-display the message for 2second and Toast.LENGHT_LONG- for 3.5 seconds) for the toast.
- setGravity() is used for positioning the toast. The standard toast appears at **bottom of the screen, centered horizontally**

ImageButton

- is a button with an image on it
- is similar with the button component except that its value is an Image than a text
- Example

```
<ImageButton
      …
      android:src="@drawable/the_image_button_icon"
    />
```

- Creating image button programatically

```
Button button=new Button(this);
 button.setImageSource(R.drawable.the_image_button_icon);
```

TextView

- is used to display text to the user (label)
- Example

```
<TextView
    android:id="@+id/txtview "
    android:layout_width="fill_parent "
    android:layout_height="wrap_content"
    android:text="Hello World"
  />
```

EditText

- allows users to edit its text content. It is also known as a text field
- It can be either single line or multi-line.
- Touching a text field places the cursor and automatically displays the keyboard.
- In addition to **typing**, text fields allow for a variety of activities such as text **selection** (cut, copy, paste) and **data look-up** via auto-completion.

- **Example:**

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

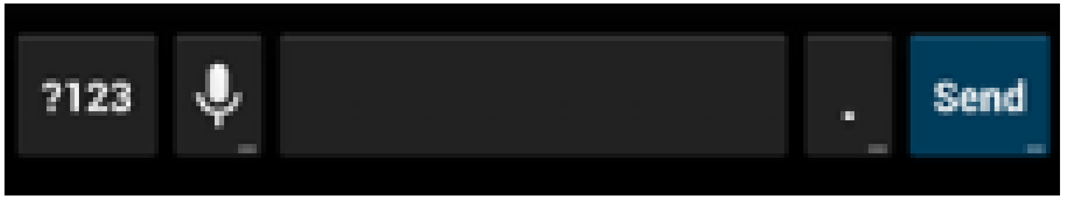
- The common input types
  - ***text***- displays a normal text keyboard
  - ***textEmailAddress***- displays a normal text keyboard with the @ character
  - ***textUri-*** displays a normal text keyboard with the / character
  - ***number-*** displays a basic number keypad
  - ***phone-*** displays a phone style keypad

- The inputType property also allows you to specify keyboard behavior:
  - ***textCapSentences***-displays normal text keyboard that capitalizes the first letter for each new sentence.
  - ***textCapWords***- displays normal text keyboard that capitalizes every word.
  - ***textAutoCorrect***- displays normal text keyboard that corrects commonly misspelled words
  - ***textPassword***- displays normal text keyboard but the characters entered turn into dots.
  - ***textMultiLine***- displays normal text keyboard that allow users to input long strings of text that include line breaks.(carriage return)
- *android:imeOptions* attribute allows you to specify an action for the EditText. Such as Search or Send.
- Example: *android:imeOptions="actionSend"*

Listening to keyboard action

- listen for a specific action event(for example actionSend) using an *TextView.onEditorActionListener*
- *TextView.onEditorActionListener* interface provides a callback method called *onEditorAction()* that indicates the action type invoked with an action ID such as IME_ACTION_SEND or IME_ACTION_SEARCH
- Example:

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



AutoCompleteTextView

- is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing (in dropdown form)
- Create AutoCompleteTextView in the xml

```
<AutoCompleteTextView
    android:layout_width="match_parent "
    android:layout_height="wrap_content"
    android:id="@+id/course_list"
/>
```

- Create the list in res/values/string.xml

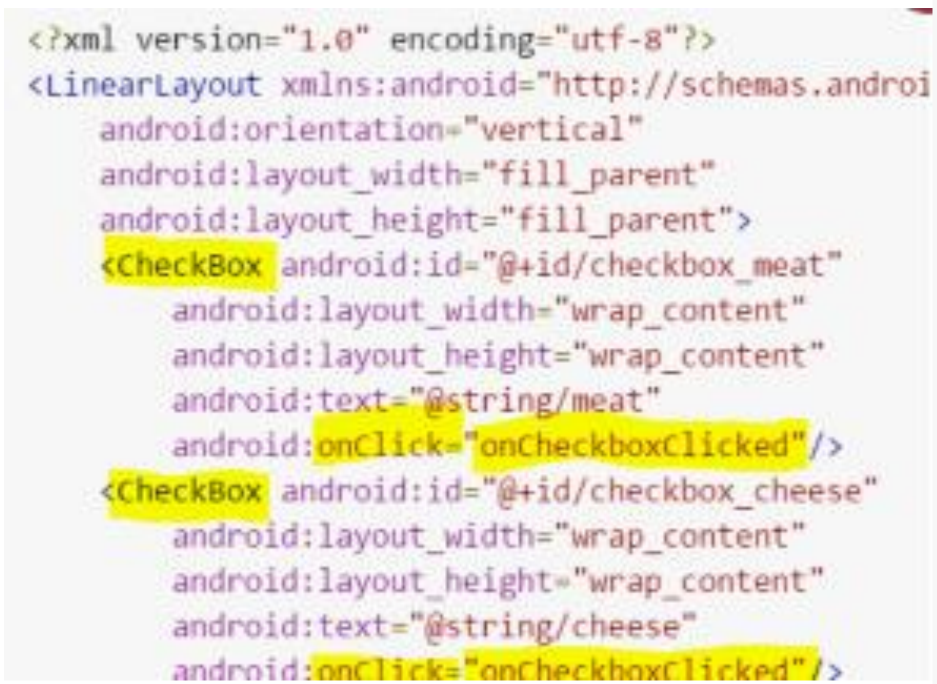
```
<resources>
    <string-array name="course_array">
        <item>OOP in Java</item>
        <item>Data Structures in Java</item>
        <item>Advanced Programming using Java</item>
    </string-array>
</resources>
```

- Get the reference of AutoCompleteTextView in java class

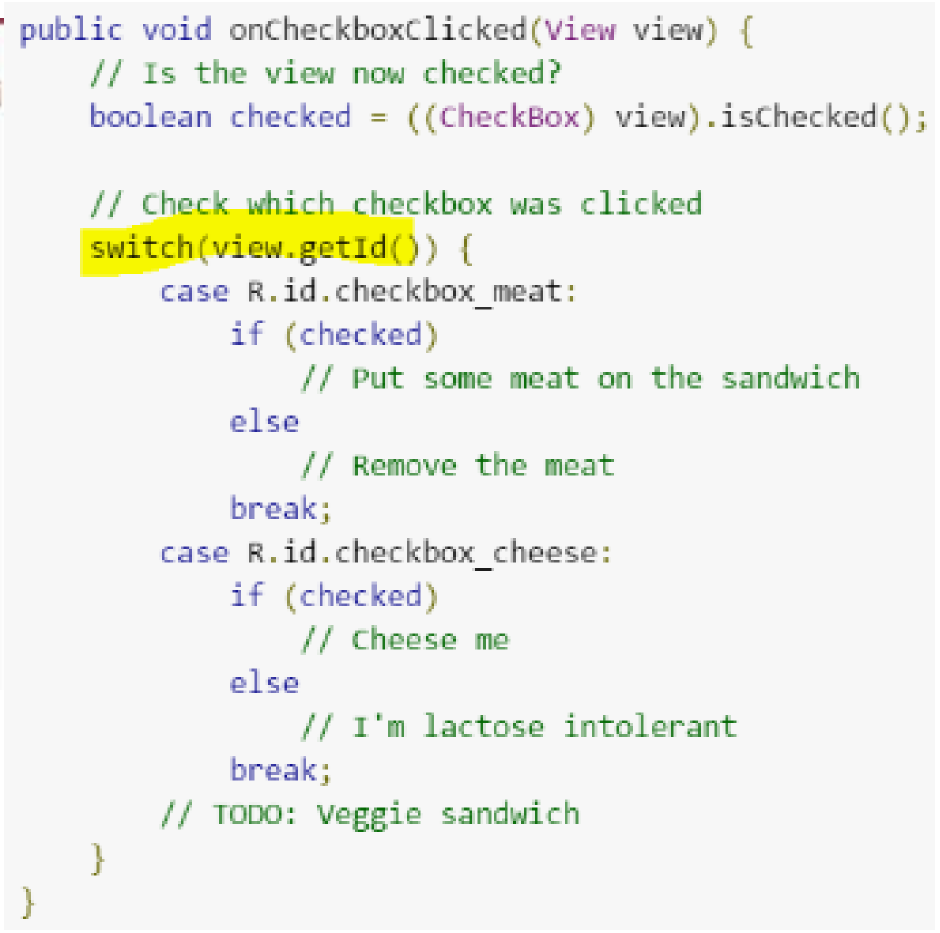
```
AutoCompleteTextView textView=(
  AutoCompleteTextView)findViewById(R.id.course_list);
String[]
  courses=getResources().getStringArray(R.array.courses_
  array);
ArrayAdapter<String> adapter=new
  ArrayAdapter,String>(this.android.R.layout.simple_list
  _item_1,courses);
textView.setAdapter(adapter);
```

CheckBox

- Allows the user to select one or more options from a set
- Typically you should present each checkbox option in a vertical list

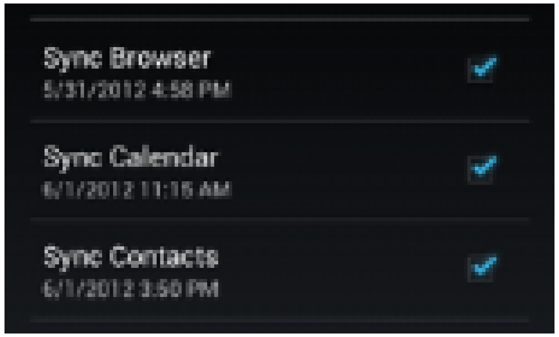


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.androi
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        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"/>
```



```
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
    }
}
```

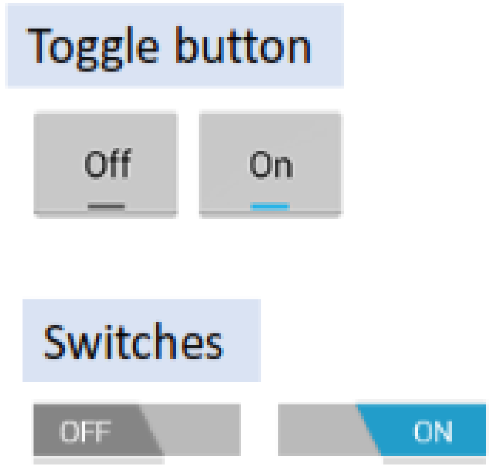


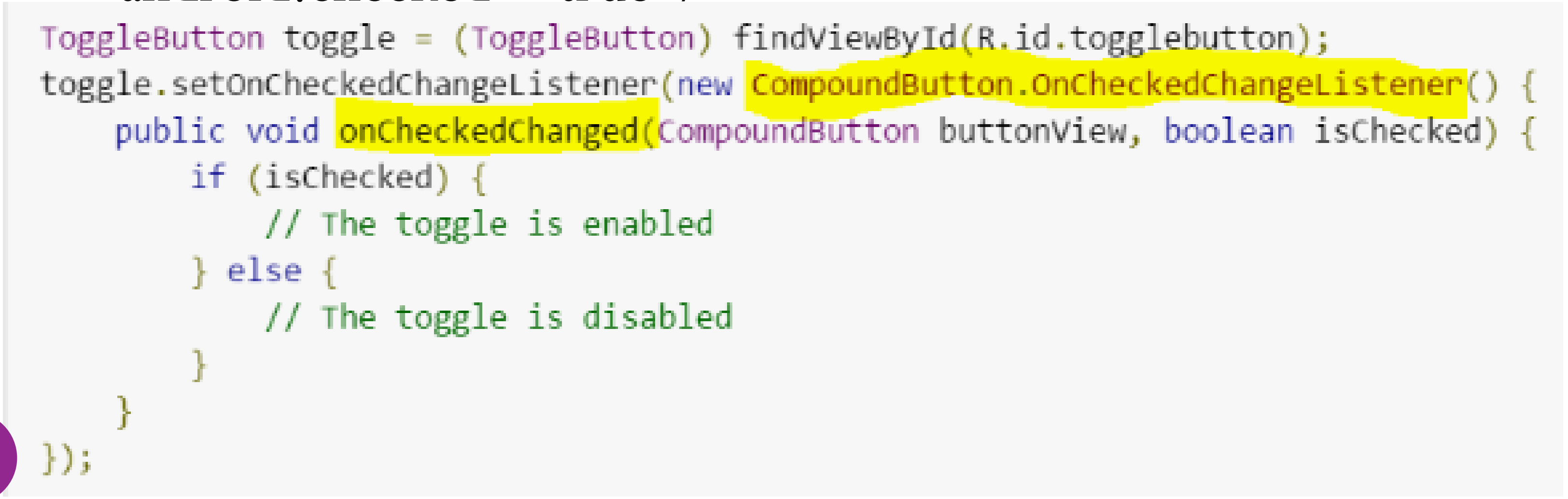
Sync Browser
5/31/2012 4:58 PM
Sync Calendar
6/1/2012 11:15 AM
Sync Contacts
6/1/2012 3:50 PM

ToggleButton

- Allows the user to change a setting between two states
- From API 14, you may use a ***Switch, that*** provides a slider control

```
<ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content“
        android:textOff=“Off"
        android:textOn=“On"
        android:checked="true"/>
```

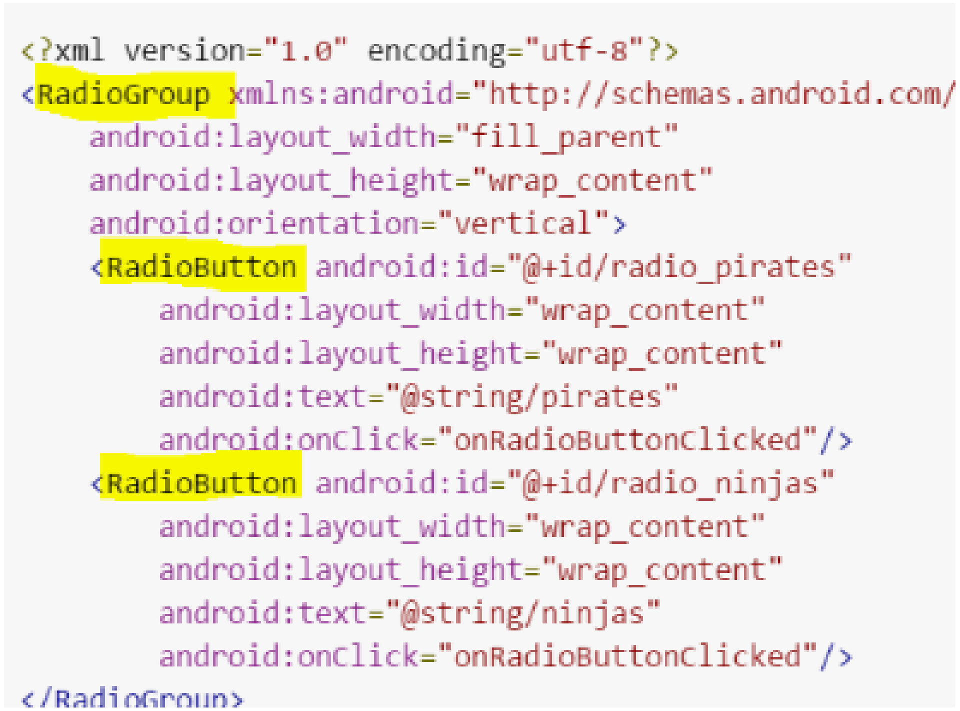




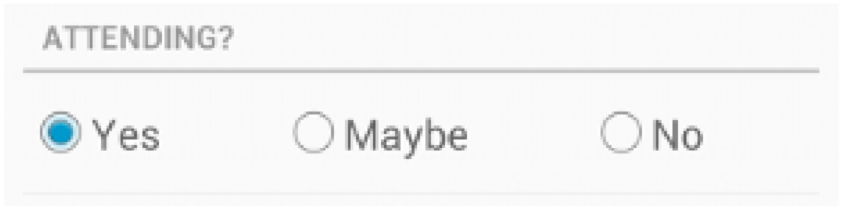
```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
  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

RadioButton

- Allows the user to select one option from a set
- RadioGroup ensures that only one radio button can be selected at a time



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



- The method ***onRadioButtonClicked*** handles the click event

Android Event Handling

- Events are a useful way to collect data about a user's interaction with interactive components of Applications
- Android framework maintains an event queue as first-in, first-out (FIFO) basis
- There are three concepts related to Android Event Management –
- **Event Listeners** – an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners Registration** – is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** – When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Some of the event listeners and event handlers are:

Event Handler	Event Listener & Description
onClick()	OnClickListener()-called when the user either clicks or touches or focuses upon any widget like button, text, image etc
onLongClick()	OnLongClickListener()-Pressing or touching UI element for one or more seconds.
onFocusChange()	OnFocusChangeListener()- called when the widget looses its focus
onKey()	OnFocusChangeListener()-called when the user is focused on the item and presses a key.
onTouch()	OnTouchListener() -called when the user presses the key, releases the key, or any movement gesture on the screen.
onMenuItemClick()	OnMenuItemClickListener()- called when the user selects a menu item.
onCreateContextMe[illegible])	onCreateContextMenuItemListener()-is called when the context menu is being built(as the result of a sustained "long click)

Event Listener Registration

1. **Using an Anonymous Inner Class**

- You have to create an anonymous implementation of the listener
- useful if each class is applied to a single control only
- you have advantage to pass arguments to event handler
- No reference is needed to call to Activity.

2. **Using the activity implements listener interface**

- your Activity class implements the Listener interface and you put the handler method in the main Activity and then you call ***setOnClickListener(this)***
- fine if your application has only a single control of that Listener type
- works poorly for multiple controls

3. **Using layout file activity_main.xml**

- put your event handlers in Activity class without implementing a Listener interface or call to any listener method
- use the layout file (activity_main.xml) to specify the handler method via the ***android:onClick*** attribute for click event.
- event handler method must have a void return type and take a View as an argument
- does not allow you to pass arguments to Listener
- Difficult to manage (you have to look the layout file)

1. Using an Anonymous Inner Class

Activity_main.xml

```
    <Button
    android:id="@+id/button_s”
android:layout_height="wrap_content”
android:layout_width="match_parent‟
android:text="@string/button_small"/>
    <TextView
    android:id="@+id/text_id”
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:capitalize="characters”
android:text="@string/hello_world" />
```

```
protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
 //--- find the button---
        Button sButton = (Button)
findViewById(R.id.button_s);
 // -- register click event with the button ---
        sButton.setOnClickListener(new
View.OnClickListener() {
  public void onClick(View v) {
 // --- find the text view --
               TextView txtView = (TextView)
findViewById(R.id.text_id);
   // -- change text size --
               txtView.setTextSize(14);
           }
        });

       }
```

String.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">EventDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
   <string name="button_small">Small Font</string>
</resources>
```

2. Using the activity implements listener interface

```
public class MainActivity extends Activity implements OnClickListener {
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  //--- find the button---
Button sButton = (Button) findViewById(R.id.button_s);
// -- register click event with the button ---
        sButton.setOnClickListener(this);
}
//--- Implement the OnClickListener callback
    public void onClick(View v) {
       if(v.getId() == R.id.button_s)
       {
// --- find the text view --
 TextView txtView = (TextView) findViewById(R.id.text_id);
 // -- change text size --
    txtView.setTextSize(14);
            return;
       }
    }
```

3. Using layout file activity_main.xml

Activity_main.xml

```
    <Button
    android:id="@+id/button_s”
android:layout_height="wrap_content”
android:layout_width="match_parent“
android:text="@string/button_small“
android:onClick="doSmall"/>

 <TextView
    android:id="@+id/text_id”
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:capitalize="characters”
android:text="@string/hello_world" />
```

```
protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
    }
//--- Implement the event handler for
the button.
public void doSmall(View v)  {
 // --- find the text view –
TextView txtView = (TextView)
findViewById(R.id.text_id);
// -- change text size --
 txtView.setTextSize(14);
       return;
   }
```

Styles and themes

- Android Style works very similar way like CSS does
- A style can specify properties such as height, padding, font color, font size, background color, and much more
- You can specify these attributes in Layout file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
. . .
  android:orientation="vertical" >
  <TextView
  android:id="@+id/text_id"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:capitalize="characters"
  android:textColor="#00FF00"
  android:typeface="monospace"
  android:text="@string/hello_world" />
</LinearLayout>
```

- But this way we need to define style attributes for every attribute separately which is not good for **source code maintenance**

Defining Style

- Open res/values/style.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:capitalize">characters</item>
    <item name="android:typeface">monospace</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textColor">#00FF00</item>/>
  </style>
</resources>
```

Using Style

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout   . . . >
  <TextView     . . .
  style="@style/CustomFontStyle"
/>
 </LinearLayout>
```

Chapter 3

Android Activities, Fragments, Intents and Services

Activity

- represents a single screen with a user interface (UI) and it will act as an entry point for users to interact with an app
- Android apps can contain multiple screens and each screen of our application is an extension of Activity class **(MainActivity class)**
- Example: Contacts App
  - List of contacts
  - Add new contacts
  - Search contacts
- in android there is a minimal dependency between the activities in an app.
- To uses the activities in application
  - we need to register those activities information in our app's manifest file (**AndroidMainfest.xml**) and
  - need to manage activity life cycle properly

```
<?xml version="1.0" encoding="utf-8"?>
  <manifest .....>
    <application .....>
      <activity android:name=".MainActivity" >
        ...
      </activity>

    ...
</application>
  </manifest>
```

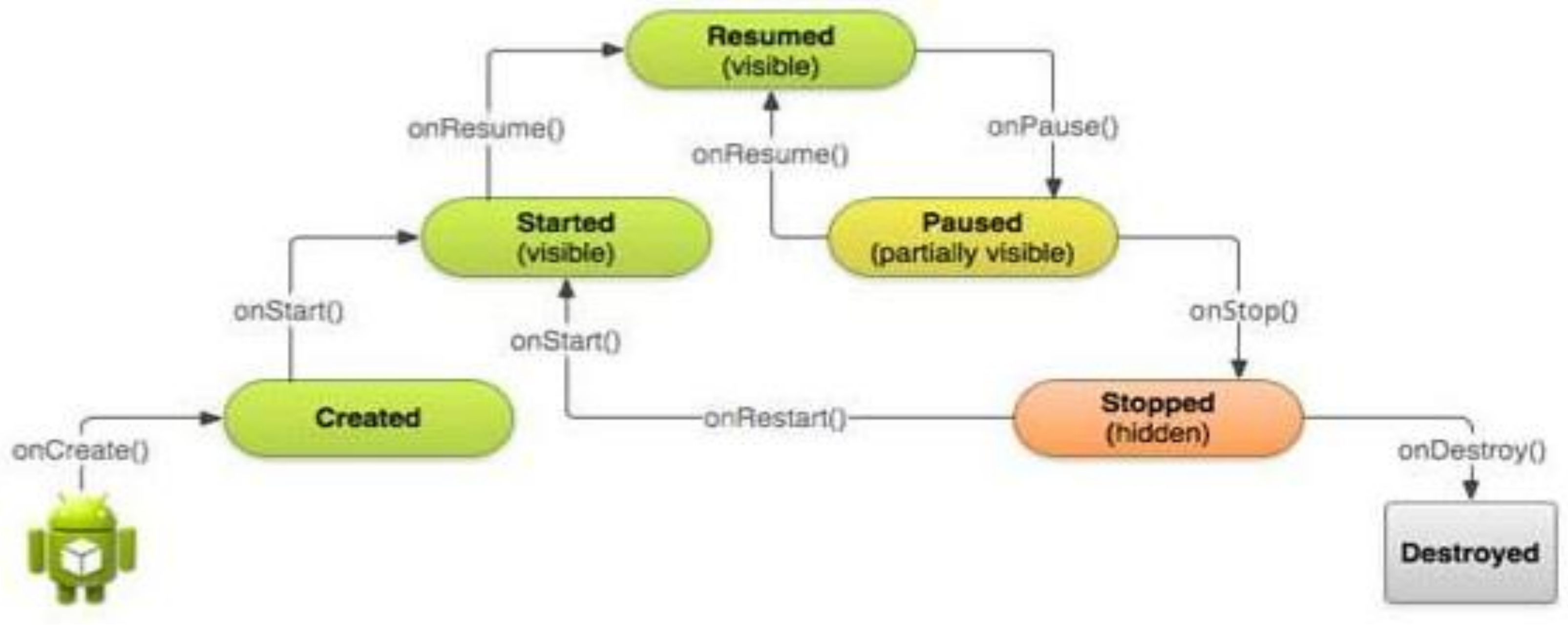
- activity attribute **android:name** will represent the name of class and we can also add multiple attributes like **icon**, **label**, **theme**, **permissions**, etc. to an activity element

- activities can be implemented as a subclass of **Activity** class

```
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Life Cycles of an Activity

- **onCreate()** — Called when the activity is first created
- **onStart()** — Called when the activity becomes visible to the user
- **onResume()** — Called when the activity starts interacting with the user onCreate()
- **onPause()** — Called when the current activity is being paused and the previous activity is being resumed
- **onStop()** — Called when the activity is no longer visible to the user
- **onDestroy()** — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- **onRestart()** — Called when the activity has been stopped and is restarting again



- Use the **onCreate()** method to **create and instantiate the objects** that you will be using in your application.
- Use the **onResume()** method to **start any services or code that needs to run** while your activity is in the **foreground**.
- Use the **onPause()** method **to stop any services or code that does not need to run** when your activity is **not in the foreground**.
- Use the **onDestroy()** method to **free up resources before your activity is destroyed**.

Fragments

- In a small-screen device (such as a smartphone), an **activity typically fills the entire screen** but in a large-screen device, such as on a tablet, it is **somewhat out of place**
- all the views in an activity must be arranged to **make full use of the increased space,** a better approach is to use "**mini-activities**", each containing its own set of views.
- In Android 3.0 and later, these **mini-activities** are known as **fragments.**
- Fragments are always embedded in an **activity**

Fragments

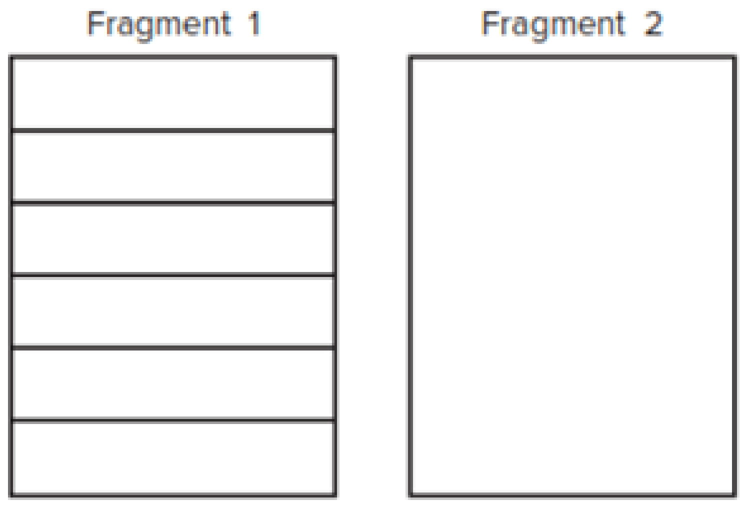


Figure 3.2.1 Fragments in an activity

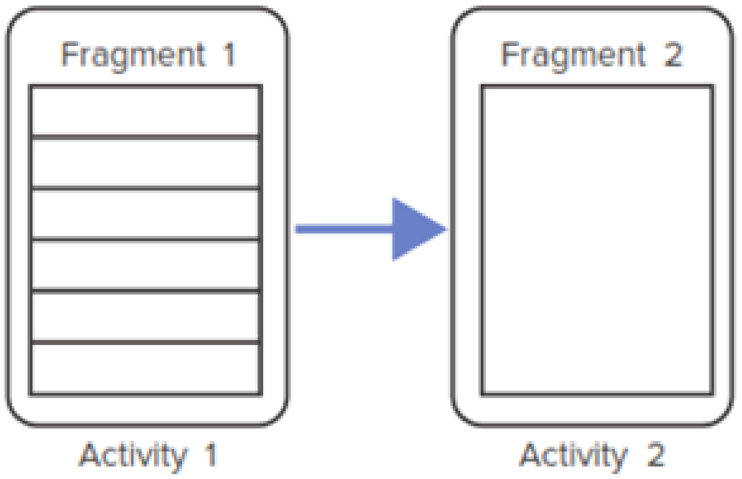


Figure 3.2.2 Fragments in different activities

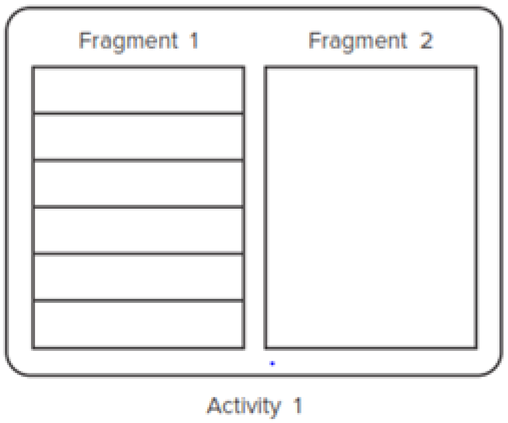


Figure 3.2.3 Fragments in landscape mode

Life Cycle of Fragments

Method	Description
onAttach()	It is called when the fragment has been associated with an activity.
onCreate()	It is used to initialize the fragment.
onCreateView()	It is used to create a view hierarchy associated with the fragment.
onActivityCreated()	It is called when the fragment activity has been created and the fragment view hierarchy instantiated.
onStart()	It is used to make the fragment visible.
onResume()	It is used to make the fragment visible in an activity.
onPause()	It is called when fragment is no longer visible and it indicates that the user is leaving the fragment.
onStop()	It is called to stop the fragment using onStop() method.
onDestoryView()	The view hierarchy which associated with the fragment is being removed after executing this method.
onDestroy()	It is called to perform a final clean up of the fragments state.
nDetach()	It is called immediately after the fragment disassociated from the activity.

Intents

- Android uses Intent for communicating between the components (such as activities, services, broadcast receivers and content providers) of an Application and also from one application to another application.
- It helps you to redirect your activity to another activity on occurrence of any event
- For example **startActivity()** you can perform this task.

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

  - foreground activity is getting redirected to another activity i.e. SecondActivity.java.
  - getApplicationContext() returns the context for your foreground activity

Types of Intents

- **Explicit Intent:**
  - Explicit Intents are used to connect the application internally.
  - In Explicit we use the name of component which will be affected by Intent
  - Explicit Intent works internally within an application to perform navigation and data transfer. Example:

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

  - Here **SecondActivity** is the JAVA class name where the activity will now be navigated.

Implicit Intent:

- In Implicit Intents we do need to specify the name of the component.
- We just specify the Action which has to be performed and further this action is handled by the component of another application. The basic example of implicit Intent is to open any web page.

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

- Unlike Explicit Intent you do not use any class name to pass through Intent().

Benefits of Intents

- **For activity**: Intent object helps to start a new activity and passing data to the second activity
- **For Services**: Services work in background, Intents could be used to start a Service
- **For Broadcast Receivers:** Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.
- **For Android Applications:** Whenever you need to navigate to another activity of your app or you need to send some information to next activity then we can always prefer to Intents for doing so.

Services

- **Android service** is a component that is *used to perform long running operations on the background* such as playing music, handle network transactions, interacting with content providers etc.
- It doesn't have any UI (user interface)
- service can be bounded by a component to perform interactivity and inter process communication (IPC).

Life Cycle of Android Service

1) **Started Service (foreground or background)**

   - A service is started when component (like activity) calls **startService()** method, now it runs in the background indefinitely.
   - It runs in the background even if the application that started the service is closed.
   - It is stopped by **stopService()** method.
   - The service can stop itself by calling the **stopSelf()** method.

2) **Bound Service**

   - A service is bound when another component (e.g. client) calls **bindService()** method.
   - The client can unbind the service by calling the **unbindService()** method. The service cannot be stopped until all clients unbind the service.

Chapter 4

Android Menu

Menu

- **Menu** is a part of user interface (UI) component which is used to handle some common functionality around the application
- useful for displaying additional options that are not directly visible on the main UI of an application
- Menu can be defined in separate XML file and use that file in our activities or fragments based on our requirements.
- **Define android menu**
  - create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML file to build the menu
  - Use menu, item and group(Optional) xml tags
  - You can also create sub menu

```
<item>
      <menu><item></item></menu>
</item>
```

  - Use **MenuInflater.inflate()** to load the menu from an activity and define an event handler to inform what to perform when the menu is selected

- Creating Menu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/inbox“     android:icon="@drawable/ic_inbox"
      android:title="@string/inbox" />
   <item android:id="@+id/compose"     android:icon="@drawable/ic_compose"
      android:title="@string/compose"     android:showAsAction="ifRoom" />
   <item android:id="@+id/outbox“       android:icon="@drawable/ic_outbox"
      android:title="@string/outbox" />
</menu>
```

- Creating Sub menu

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

Types of Android Menu

- Options Menu
- Context Menu
- Popup Menu

Option Menu

- useful to implement actions that have a global impact on the app, such as Settings, Search, etc

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

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

- Handling android option menu click event

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.inbox:
            // do something
            return true;
        case R.id.compose:
            // do something
            return true;
        default:
            return super. onOptionsItemSelected(item);
    }
}
```

Context Menu

- a floating menu that appears when the user performs a long click on an element
- more like the menu which displayed on right click in Windows or Linux
- It does not support any item shortcuts and item icons

```
<?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" >
   <Button
      android:id="@+id/btnShow"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Long press me"
      android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button btn = (Button) findViewById(R.id.btnShow);
    registerForContextMenu(btn);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextM
enuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Context Menu");
    menu.add(0, v.getId(), 0, "Inbox");//groupId,ItemId,order,Title
    menu.add(0, v.getId(), 0, "Outbox");
}
```

- Handling ContextMenu click event

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    if (item.getTitle() == "Inbox") {       // do your coding    }
    else {       return  false;    }
    return true;
}
```

Popup Menu

- displays a list of items in a vertical list that's anchored to the view that invoked the menu
- It does not support any item shortcuts and item icons
- Popup menu is available with API level 11 (Android 3.0) and higher versions

```
<?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" >

   <Button
      android:id="@+id/btnShow"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Show Popup Menu"
      android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/inbox_item"
        android:title="Inbox" />
    <item android:id="@+id/compose_item"
        android:title="Compose" />
    <item android:id="@+id/outbox_item"
        android:title="Outbox" />
    <item android:id="@+id/spam_item"
        android:title="Spam" />
    <item android:id="@+id/logout_item"
        android:title="Logout" />
   </menu>
```

```
public class MainActivity extends AppCompatActivity implements PopupMenu.OnM
enuItemClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button) findViewById(R.id.btnShow);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(MainActivity.this, v);
                popup.setOnMenuItemClickListener(MainActivity.this);
                popup.inflate(R.menu.popup_menu);
                popup.show();
            }
        });
    }
@Override
    public boolean onMenuItemClick(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).s
how();
        switch (item.getItemId()) {
            case R.id.inbox_item:
                // do your code
                return true;
            case R.id.compose_item:
                // do your code
                return true;
            case R.id.outbox_item:
                // do your code
                return true;
            case R.id.spam_item:
                // do your code
                return true;
            case R.id.logout_item:
                // do your code
                return true;
            default:
                return false;
        }    } }
```

Quiz

- What is android service? List some android services.
- What is android menu?
- Discuss the differences between option, context and popup menus.

Chapter 5

Android Data Storage

Android Data Storage

- **Android provides several options to save application data:**
  - **Shared Preference:**
    - store private primitive data in key-value pairs
  - **Internal Storage:**
    - store private data on device memory
  - **External Storage:**
    - store public data on shared external storage
  - **SQLite Database:**
    - store a structured data in a private database
  - **Network Connection:**
    - store data on the web with your network server

Shared Preference

- used to save and retrieve the primitive data types (integer, float, Boolean, string, long) data in the form of key-value pair from a file within an apps file structure. For example:-username-abc, password – abc123
- Shared Preferences can also be used to **manage session**. Session is a way of making some data available throughout an application.
- **Shared Preferences** object will point to a file that contains a key-value pairs and provides a simple read and write methods to save and retrieve the key-value pairs from a file.
- **Shared Preferences file**
  - is managed by an android framework
  - can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured
  - Data is stored in XML file under **data/data/package-name/shared-prefs** folder

Accessing Shared Preference

- **SharedPreference** object provides two methods
  - **getSharedPreferences**( ):-useful to get the values from multiple shared preference files by passing the name as parameter to identify the file. It can be called from any Context in our app. For example

SharedPreferences sp=getSharedPreferences("filename1",Context.MODE_PRIVATE);

  - **getPreferences**():-for **activity level preferences** and each activity will have it's own preference file and by default this method retrieves a default shared preference file that belongs to the activity. For example

SharedPreferences sp = getPreferences(Context.MODE_PRIVATE);

Write to Shared Preference

- We need an **editor** to edit and save the changes in **SharedPreferences** object. Example

```
SharedPreferences sp = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.putString("keyname",stringValue);
editor.putInt("keyname",integerValue);
editor.commit();
```

Read from Shared Preference

```
SharedPreferences sp = getPreferences(Context.MODE_PRIVATE);
sp.getString("keyname",null);
sp.getInt("keyname",0);
```

Delete from Shared Preference

```
SharedPreferences sp = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.remove("keyname");
editor.commit();
```

Clearing from Shared Preference

- We can clear all the data from Shared Preferences file

```
SharedPreferences sp=getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sp.edit();
editor.clear();
editor.commit();
```

Internal Storage

- useful to store or retrieve data to/from the device's internal memory using **FileOutputStream/FileInputStream** object
- These files are **private** to your application and the files are removed when you **uninstall** the application.
- **To write a file to internal storage**
  - Call **openFileOutput(String filename, int mode).** The modes can be Context.MODE_PRIVATE, Context.MODE_APPEND
  - Write to the file with **write()** method
  - Close the stream with **close()** method

```
String FILENAME = "user_details";
String name = "abebe";
FileOutputStream fstream = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fstream.write(name.getBytes());
fstream.close();
```

- **To read a file to internal storage**
  - Call **openFileInput(String filename)**.
  - Read bytes from the file with **read()** method. (since we read data byte by byte we have to append each character using **append()** method)
  - Close the stream with **close()** method

```
String FILENAME = "user_details";
FileInputStream fstream = openFileInput(FILENAME);
StringBuffer sbuffer = new StringBuffer();
int i;
while ((i = fstream.read())!= -1){
    sbuffer.append((char)i);
}
fstream.close();
```

External Storage

- **To write data to external storage**
  - Add permissions(such as READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE) to read or write files on the external storage.

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

  - Check External Storage Availability:

```
public Boolean isExternalStorageWritable(){
 String state=Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state) )  return true;  }
return false;   }
public Boolean isExternalStorageReadable(){
 String state=Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state) || Environment.MEDIA_MOUNTED_RE
AD_ONLY.equals(state) )  return true;  }
return false;
 }
```

- Get the file directory by calling
  - **getExternalStorageFilesDir()-** create files specific to your application and removed when your application is uninstalled

File folder=getExternalStorageFilesDir("MyFolder");

Files myfile=new File(folder,"mydata.txt");

  - **getExternalStoragePublicDirectory()-** not specific to your application and not removed when your application is uninstalled. It includes directories like DIRECTORY_MUSIC, DIRECTORY_PICTURES,DIRECTORY_RINGTONES, DIRECTORY_DOWNLOADS

File folder=Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);

Files myfile=new File(folder,"mydata.txt");

- Use FileOutputStream object to write data:

FileOutputStream fstream = new FileOutputStream(myfile);

fstream.write(name.getBytes());
fstream.close();

- **To read data from external storage**
  - Get the file directory by calling
    - **getExternalStorageFilesDir() or**
    - **getExternalStoragePublicDirectory()**
  - Call openFileInputStream method by passing the file name as argument
  - Read bytes from the file using read() method
  - Close the stream using close() method

```
String FILENAME = “mydata.txt";
File folder = Environment.getExternalStoragePublicDirectory(Environment.DIR
ECTORY_DOWNLOADS);
File myfile = new File(folder, FILENAME);
FileInputStream fstream = new FileInputStream(myfile);
StringBuffer sbuffer = new StringBuffer();
int i;
while ((i = fstream.read())!= -1){
   sbuffer.append((char)i);
}
fstream.close();
```

SQLite

- open source light weight RDBMS used to perform database operations
- **Advantage**
  - Serverless which means it is simple to set up and zero configuration is required
  - File based system makes it very portable (android stores our database in a private disk space that's associated to our application and the data is secure)
  - Great for development and testing
- **Disadvantage**
  - Does not provide network access (i.e. accessing it from another machine)
  - Not built for large-scale applications
  - No user management
- **android.database.sqlite** contains all the required API's to use SQLite database

Creating Database and Table

- by using **SQLiteOpenHelper** class we can easily create required database and tables for our application
- To use **SQLiteOpenHelper**, override the **onCreate()** and **onUpgrade()** methods (See the example)

Insert data into SQLite Database

- we can insert data into SQLite database by passing **ContentValues** to **insert()** method

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
//Create a new map of values, where column names are the keys
ContentValues cValues = new ContentValues();
cValues.put(COLUMN_NAME, name);
cValues.put(COLUMN_EMAIL, email);
// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(TABLE_Users,null, cValues);
```

Retrieving data from SQLite Database

- we can read the data from SQLite database using **query()** method

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
Cursor cursor = db.query(TABLE_Users, new String[]{COLUMN_NAME, COLUMN_E
MAIL}, COLUMN_EMAIL+ "=?",new String[]{String.valueOf(userid)},null,null, null);
//tblName, columns, selection, selectionargs, groupBy, having, OrderBy
```

Update data from SQLite Database

- we can update the data in SQLite database using **update()** method in android applications

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
ContentValues cVals = new ContentValues();
cVals.put(COLUMN_NAME, name);
int count = db.update(TABLE_Users, cVals, COLUMN_EMAIL+" = ?",new String[]{Str
ing.valueOf(id)});
```

Deleting data from SQLite Database

- we can delete data from SQLite database using **delete()** method

```
//Get the Data Repository in write mode
SQLiteDatabase db = this.getWritableDatabase();
db.delete(TABLE_Users, COLUMN_EMAIL+" = ?",new String[]{String.valueOf(userid)});
```

Web Service with PHP/Java & MySQL

- External Databases are also the data repository for android apps for example apps that provide weather information, exchange rates, world news etc
- MySQL is one of the database tool. Connection between an android app and mysql can be done through
  - Web Services
  - JDBC without web services

Web Services

- A **Web Service is a Consumer_Machine-to-Provider_Machine** collaboration schema that operates over a computer network.
- The data exchanges occur independently of the ***OS, browser, platform, and programming languages*** used by the **provider** and the **client**.

- Web Services expect the computer network to support standard Web protocols such as XML, HTTP, HTTPS, FTP, and SMTP.

Advantages of Using the Web Service Architecture

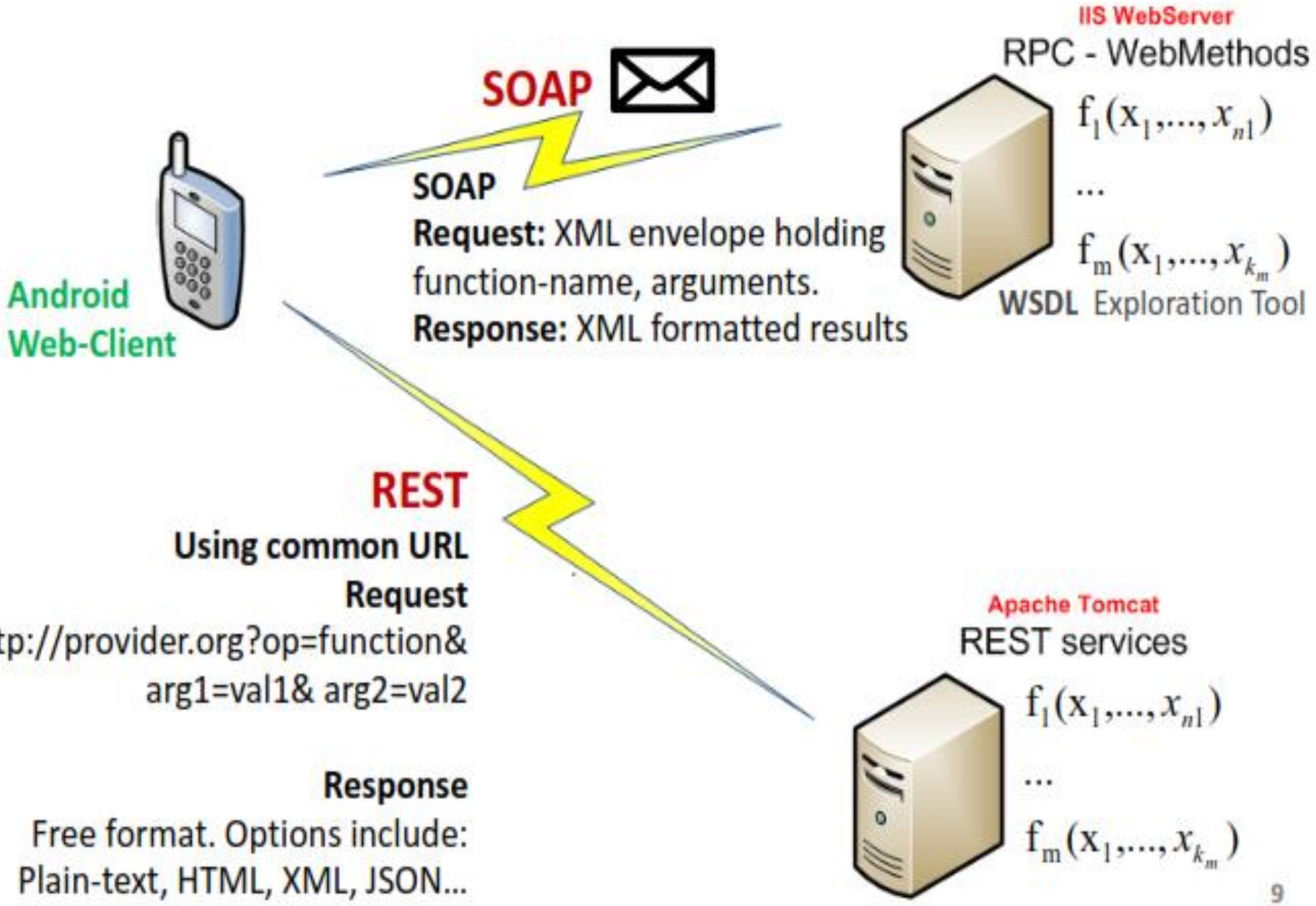
- **invoked functions are *implemented once (in the server) and called many times (by the remote users)*.**
- Elimination of redundant code,
- Ability to transparently make changes on the server to update a particular service function without clients having to be informed.
- Reduced maintenance and production costs.

There are two widely used forms of invoking and using Web Services

- **Representational State Transfer (REST)**
  - Closely tie to the **HTTP protocol** by associating its operation to the common methods: GET, POST, PUT, DELETE for HTTP/HTTPS.
  - has a **simple invocation mode and little overhead**.
  - Service calls rely on a **URL** which may also carry **arguments**. Sender & receiver must have an understanding of how they pass data items from one another. As an example: **Google Maps API uses the REST model**.
- **Remote Procedure Call (RPC).**
  - Remote services are seen as coherent collections of discoverable functions (or method calls) stored and exposed by EndPoint providers.
  - Some implementations of this category include: Simple Object Access Protocol (SOAP), Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) and Sun Microsystems's Java/Remote Method Invocation (RMI).

REST Vs SOAP

- **REST users refer to their remote services through a conventional URL** that commonly includes the location of the (stateless) **server**, the **service name**, the **function to be executed** and the **parameters needed by the function to operate** (if any). Data is transported using HTTP/HTTPS.
- **SOAP requires some scripting effort to create an XML envelop in which** data travels. An additional **overhead on the receiving end is needed to extract data from** the XML **envelope**. SOAP accepts a variety of transport mechanisms, among them HTTP, HTTPS, FTP, SMTP, etc.
- SOAP uses **WSDL (Web Service Description Language) for exposing** the format and operation of the services.



SOAP

SOAP
Request: XML envelope holding function-name, arguments.
Response: XML formatted results

REST

Using common URL
Request
http://provider.org?op=function& arg1=val1& arg2=val2

Response
Free format. Options include: Plain-text, HTML, XML, JSON...

Connecting Android App with MySQL using RESTful web service

- **Download and install** XAMPP(or WAMP)
- Check whether the server work properly or not
  - Create a folder called test-android under C:\xampp\htdocs\ folder
  - Test the server by running http://localhost/test-android/
- Open the phpMyAdmin by visiting http://localhost/phpMyAdmin and create a database and table for your app
- Create connection and php files to handle database operation
- Create android project, Create the activities and create **JSONParser** for data exchange format
- Add the following to the application build gradle

```
android{  useLibrary 'org.apache.http.legacy’  }
dependencies{   compile "org.apache.httpcomponents:httpcore:4.3.2“ }
```

- Allow remote connection permission in AndroidManifest.xml file.

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

- Use http://10.0.2.2/app-folder/phpfile or use an IP address if you want to access the database from your device

Chapter 6

Android Telephony

Telephone Manager

- **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc.

```
//Get the instance of TelephonyManager
    TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SER
VICE);
 //Calling the methods of TelephonyManager the returns the information
     String IMEINumber=tm. getIMEINumber();
     String subscriberID=tm.getDeviceId();
     String phoneID=tm.getLine1Number();
     String SIMSerialNumber=tm.getSimSerialNumber();
     String networkCountryISO=tm.getNetworkCountryIso();
     String SIMCountryISO=tm.getSimCountryIso();
     String softwareVersion=tm.getDeviceSoftwareVersion();
     String voiceMailNumber=tm.getVoiceMailNumber();
```

- Give READ_PHONE_STATE permission in android manifest file

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

Make a phone call

- making a phone call from our android applications is done easily by invoking built-in phone calls app using Intents action (**ACTION_CALL**).

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:" + txtPhone.getText().toString()));
startActivity(callIntent);
```

- we need to add a **CALL_PHONE** permission in our android manifest.

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

Sending SMS

- we can send SMS from our android application in two ways either by
  - using **SMSManager** api or

```
SmsManager smgr = SmsManager.getDefault();
smgr.sendTextMessage(MobileNumber,null,Message,null,null);
```

  - Intents **ACTION_VIEW** action

```
Intent sInt = new Intent(Intent.ACTION_VIEW);
sInt.putExtra("address", new String[]{txtMobile.getText().toString()});
sInt.putExtra("sms_body",txtMessage.getText().toString());
sInt.setType("vnd.android-dir/mms-sms");
```

- Make sure to allow **SEND_SMS** permission android manifest file

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

Sending Email

- we can easily send an email from our android application using **existing email clients** such as **GMAIL**, **Outlook**, etc
- The **Intent** object in android with proper action (**ACTION_SEND**) and **data** will help us to **launch the available email clients to send** an email in our application

```
Intent it = new Intent(Intent.ACTION_SEND);
it.putExtra(Intent.EXTRA_EMAIL, new String[]{"abc@gmail.com "});
it.putExtra(Intent.EXTRA_SUBJECT, "test");
it.putExtra(Intent.EXTRA_TEXT, "Hello this is to inform you that…."); 
it.setType("message/rfc822");
```

- **it** - Our local implicit intent
- **ACTION_SEND** - It's an activity action which specifies that we are sending some data.
- **putExtra** - add extra information to our Intent. Example: recipient, subject and message
  - EXTRA_EMAIL - It's an array of email addresses
  - EXTRA_SUBJECT - The subject of the email that we want to send
  - EXTRA_TEXT - The body of the email
- **setType** - set the MIME type of data to be sent. Example"**message/rfc822**" and other MIME types are "**text/plain**" and "**image/jpg**".

- We need to add **MIME type** and **Permission** in our android manifest file code like as shown below

```
<manifest …>
  <application
    …
        <activity android:name=".MainActivity">
          <intent-filter>
             <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
             <action android:name="android.intent.action.SEND"/>
             <category android:name="android.intent.category.DEFAULT"/>
             <data android:mimeType="message/rfc822"/>
          </intent-filter>
       </activity>
     </application>
  </manifest>
```

- **action** - we use this property to define that the activity can perform **SEND** action.
- **category** - we included the **DEFAULT** category for this activity to be able to receive implicit intents.
- **data** - the type of data the activity can send.

Chapter 7

Location in Android

Location Based Services

- What makes mobile different is we can build advanced services that only **mobile device with sensing can deliver**
  - **location-based search** for say supermarkets, cafe, cinema, users etc
  - Examples DarkSky(Weather App), MapMyFitness, Glympse (Tracking App), Uber etc
- **Android has two basic ways to determine a user's location.**
  - **built-in location APIs –**since its introduction
  - **Google Play Services –** new and best way
- Google Location Services API provides
  - more powerful, high-level framework that automates tasks such as **location provider choice and power management**.
  - new features such as **activity detection**
- **To use these services**
  - download the Google Play Services SDK using the SDK Manager
  - download an emulator (AVD) image that uses the Google APIs

Location Manager

- Android location manager gives location in terms of **longitude and latitude** for the location of the phone.
- Depending on the location provider selected (could be based on GPS, WiFi or Cellular) the accuracy of the location will vary
- A number of services can be built using these simple components:
  - **get the user's current location**
  - **periodically get the user location as the move around**
  - **use proximity alerts when you move in and out of a predefined area**

```
LocationManager locationManager;
String svcName = Context.LOCATION_SERVICE;
locationManager = (LocationManager)getSystemService(svcName);
```

- Modify androidmanifest to get the user's permission to track their location or get a location reading:

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

- Types of location access permission:
  - *ACCESS_COARSE_LOCATION*- support Network providers (cell towers or wifi)
  - *ACCESS_FINE_LOCATION* -support both GPS and Network providers

Location Provider

- Mobile phones can provide location from a set of providers
  - **GPS-** has good accuracy outdoors but is costly in terms of battery consumption
  - **Cellular** - cheap in terms of energy consumption but could provide very rough location information because of the lack of cell tower density but could be great in the city
- Consider the following in selecting a location provider:
  - power consumption
  - longitude/latitude accuracy
  - altitude accuracy
  - speed
  - direction information

- you can specify the location provider explicitly in the code using a number of constants:
  - LocationManager.GPS_PROVIDER
  - LocationManager.NETWORK_PROVIDER
  - LocationManager.PASSIVE_PROVIDER
- But this would be poor programming because the user might turn off GPS. So **let the Android systems match the user's needs to what providers are on offer** by using *Criteria* as shown below. The code states that the user requires:
  - coarse accuracy
  - low power consumption
  - no altitude, bearing or speed

```
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    criteria.setAltitudeRequired(false);    criteria.setBearingRequired(false);
    criteria.setSpeedRequired(false);    criteria.setCostAllowed(true);
String provider = locationManager.getBestProvider(criteria, true);
```

Geocoding

- is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate.
- Geocoder supports two services:
  - forward geocoding: from address to longitude/latitude
  - reverse geocoding: from longitude/latitude to address
    - Where latitude and longitude are points for the search
- The Geocoder class comes with the **Google Maps library**. To use the library you have to import it into the application.
- In addition, the Geocoder class **uses a server** to translate over the Internet so you need to add the following permission to the Manifest:

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

Integrating Google map with android app

- provides facility to integrate Google map in our application
- **Types of Google Maps**
  - **Normal:** displays typical road map**, natural features like river and some features** build by humans.
  - **Hybrid:** displays **satellite photograph** data with typical **road maps. It also displays road and feature labels.**
  - **Satellite:** displays **satellite photograph data, but doesn't display road and feature labels.**
  - **Terrain:** displays **photographic** data. This includes **colors, contour lines and labels and perspective shading.**
  - **None:** displays an empty grid with no tiles loaded.

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Steps to integrate google map in android application

- install **Google Play Services** SDK in our Android Studio
  - To install Google Play Services, open **Android Studio** → Go to **Tools** menu → **Android** → click **SDK Manager**, then new window will open in that select **SDK Tools** tab → Select **Google Play Services** → click **OK.**
- Create an Android project and select Google maps activity.
- Get a Google Map API key
  - Go to your project an open **google_maps_api.xml file in res/values directory.** Copy the link provided in the **google_maps_api.xml** file
  - Paste the console URL in browser and it will take you to **Google API Console, Create new project and press Agree and Continue.**
  - Click on **Create API Key** to create an API key.
- copy the API Key, go back to android studio and paste the API key into the **<string>** element in **google_maps_api.xml** file.

  <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">
  AIzaSyCKPTaBv41DKqr9qxMPWOQAsqp0Q4NHMER</string>
- modify AndroidManifest.xml file by adding user permission like:
  - INTERNET: To determine if we are connected to the internet or not.
  - ACCESS_FINE_LOCATION: to use GPS as content provider (Wifi and mobile too)
  - ACCESS_COARSE_LOCATION: to use Wi-Fi and mobile data as content provider
- Add the following to android build gradle dependencies
  - compile 'com.google.android.gms:play-services-maps:16.1.0'

Chapter 8

Publishing, Distributing, Monetizing, and Promoting Applications

Signing and Publishing android app

- Android applications are distributed as Android package files (.APK).
- To be installed on a device it needs to be signed. This is done before distributing the app or release it for use.
- To sign an app
  - JDK includes the **Keytool** and **Jarsigner** command-line tools which are necessary to create a new keystore/signing certificate
  - Alternatively, you can use the **Generate signed bundle/apk wizard**
- Importance of Signing Certificate
  - It is a means of ***identifying the authenticity of application updates***, and ***applying inter-process security boundaries between installed*** applications.
  - If you lose your certificate, you would need to **create a new listing**, **losing all the reviews, ratings**, and **comments associated** with your previous package, as well as making it impossible to **provide updates to the existing users of your application**
- Follow the steps mentioned on the handout to sign and publish your app

Distribute android app

- One of the advantages of Android's open ecosystem is the freedom to publish and distribute your applications
- The most common and popular distribution channel for android apps is **Google Play**
- But you are also free to distribute your applications using **alternative markets(**including the Amazon App Store, GetJar, Mobogenie, SlideME, F-Droid and carrier-specific stores), **your own website**, **social media**, or **any other distribution channel**
- **When you distribute your app make sure that**
  - **application package name is unique. Because it is** used as unique identifiers for each application
  - The filename of your APK **does not have to be unique because** it will be discarded during the installation process

Monetizing android app

- Android enables you to monetize your applications using whatever mechanism you choose
- If you choose to distribute and monetize your applications using Google Play, three options are available:
  - **Paid applications** — Charge users an upfront fee before they **download** and **install** your application.
  - **Free applications with In-App Billing (IAB)** — Make the download and installation of the application free, but charge within the application for **virtual goods, upgrades, and other value-adds**.(More effective)
  - **Advertising-supported applications** — Distribute the application for free, and monetize it by **displaying advertising**.
- If you want to monetize your app in google play using either of the first two options
  - You should have a **merchant account** and you are **responsible for any legal or taxation obligations** associated with the sale of your application
  - the revenue split between google and the developer is 30/70 respectively.

- If you want to monetize your application using **in-app advertising**
  - You are required to set up advertising within your application:(it may vary asper the ad providers)
    - Create a publisher account.
    - Download and install the associated ads SDK.
    - Update your Fragment or Activity layouts to include an ad banner.
- In many cases, developers have chosen to **offer a paid alternative** (either using up-front payment or IAB) to **allow users to eliminate ad banners from their applications**.

Application Marketing and Promotion Strategy

- 1st step- provide the full set of **high-quality assets for your Google Play listing**
- marketing and promotion strategies vary widely depending on your goals and budget, here are some of the most effective techniques to consider:
  - **Offline cross promotion** —offline presence (such as a stores or branches), or a large media presence (such as within newspapers, magazines, or on TV), these increases awareness and help to ensure users **trust the download**.
  - **Online cross promotion** — if you have a website, promoting your application through direct links to Google Play can be an effective way to drive downloads.
  - **Third-party promotion** — Distributing a promotional video on ***YouTube and leveraging social networks, blogs, press releases, and online review sites*** can help you provide positive word of mouth.
  - **Online advertising** — Online advertising using **in-app ad networks** (such as **AdMob**) or traditional **search-based advertising** (such as **Google AdWords**) can drive significant impressions and downloads for your application.

Application Launch Strategy

- **Ratings and reviews** can have a significant impact on your application's ranking in category lists and within Google Play search results. Poor launch will have impact on it.
- Here are some of the strategies you can use to ensure a successful launch:
  - **Iterate on features not quality** - A poorly implemented but feature-rich application **will receive worse reviews** than a well-polished application that **doesn't do everything**. Ensure each release is of the **same high quality**, **adding new features** as part of each release. Similarly, each **release should be more polished and stable than the last.**
  - **Create high quality Google Play assets -** The **first impression** your application makes is through its appearance in Google Play. Create assets that represent the quality of your application.
  - **Be honest and descriptive** - Disappointed users who find your application is not as it was described are likely to uninstall it, rate it poorly, and leave negative comments.

Promotion with in Google Play

- In addition to the effect of reviews, no of downloads and installs, Google Play editorial team uses several criteria to **highlight high quality applications and categorize them as Featured.**
  - **Featured applications** receiving priority placement in Google Play.
- The criteria mentioned above are **not publicly available**, but some of the general criteria associated with featured application includes:
  - **High quality and innovative** - featured applications in Google Play act as a **showcase for the platform**. So that it should be **useful and innovative**.
  - **A high degree of fit-and-finish -** include all the **requisite promotional assets**, while the applications themselves have **few bugs and a high quality UI**.
  - **Broad device and platform support** - typically support a broad range of device types and platform versions, including both handsets and tablets.
  - **Use of newly released features** - should offer an opportunity for the Google Play team to ***highlight those new features to reviewers and early adopters***.
  - **Consistency with the platform user experience** - provide a convincing user experience that is consistent with the UI and interaction models offered by the Android platform.

Analytics and Referral Tracking

- Mobile application analytics packages are effective tools for you to ***better understand who is using your application and how they are using.*** Example **Google Analytics and Flurry.**
- can help you to
  - discover bugs, prioritize your feature list and make objective decisions on where to focus your development resources
- Using these tools you can track three types of data within your application:
  - **User analytics** -Understand the **geographic locations and language settings, Internet connections speed, screen sizes and resolutions, and the display orientation of your app users.** So that it help you to prioritize **translation efforts, optimize layout and assets for different screen sizes and resolutions**.
  - **Application usage patterns** – during integrating analytics the first step is to record each Activity. This will help you understand the ***way your application is being used, optimize workflows and better understand how well the assumptions you made during design match actual usage.***
  - **Exception tracking** - In addition to printing an error, post each unique exception thrown using your analytics. This will help you to see if there **are particular devices, locations, or usage patterns that lead to particular exceptions.**