

Mobile Application Development (20IT505 / JO1A)

By`

K. Bhaskara Rao

Asst. Prof.

Dept. of Information Technology

BEC, Bapatla

2024-25



Mobile Application Development

UNIT-I

- **Hello, Android:-**Android: An Open platform for Mobile development, Android SDK Features, Introducing the Development Framework.
- **Getting Started:-**Developing for Android, Developing for Mobile and Embedded devices.

UNIT-II

- **Creating Applications and Activities:-**Components of an Android Application, Introducing the Application Manifest File, The Android Application Lifecycle, A Closer Look at Android Activities, Creating Activities, The Activity Lifecycle, Activity States Android Application class, Android Activities.
- **Building User Interfaces:-** Fundamental Android UI Design, Android User Interface Fundamentals, Introducing Layouts, Introducing Fragments.

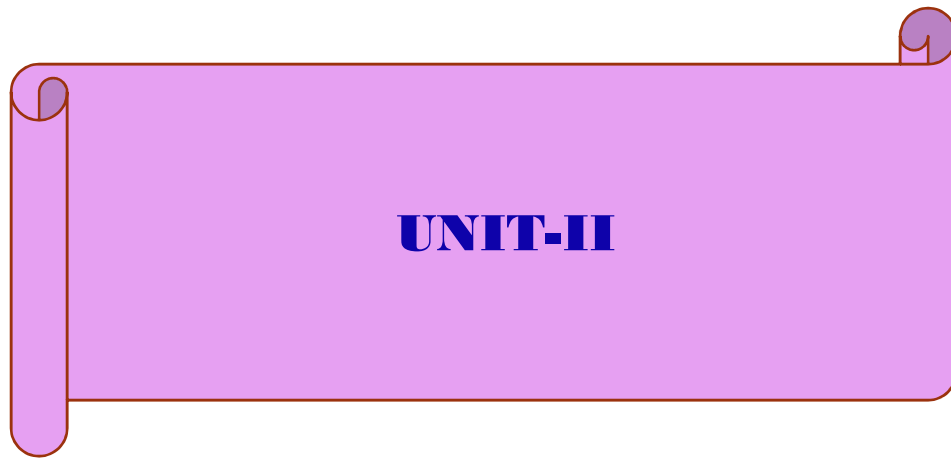
Mobile Application Development

UNIT-III

- **Intents and Broadcast Receivers:-**Introducing Intents, Creating Intent Filters and Broadcast Receivers.
- **Saving State and User Preferences:-**Creating and Saving Shared Preferences, Retrieving Shared Preferences Persisting the Application Instance State.
- **Creating and Using Databases:-** Working with SQLite Databases.

UNIT-IV

- **Content Providers:-** Creating Content Providers, Accessing Content Providers, using Native Android Content Providers.
- **Working in the Background:-** Creating and Controlling Services, Binding Services to Activities.
- **Expanding the User Experience:-** Introducing the Action Bar ,Creating and Using Menus and Action Bar Action Items.



UNIT-II

AndroidManifest.xml file

- The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code.
- The manifest lets you define the structure and metadata (icon, version number and theme) of your application and its components.
- **It does the following:**
 - ✓ It names the Java package
 - ✓ It describes the components of application (content providers, activities, services, broadcast receivers
 - ✓ It determines which processes will host application components.
 - ✓ It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
 - ✓ It also declares the permissions that others are required to have in order to interact with the application's components.
 - ✓ It lists the Instrumentation classes that provide profiling and other information.
 - ✓ It declares the minimum level of the Android API that the application requires
 - ✓ It lists the libraries that the application must be linked against.

AndroidManifest.xml file

- Version code is an integer value that represents the version of the code. It is used internally by Android OS to check for updates. It is incremented by 1 for every release.
- Version name is the string / value of version displayed for end user only.
(Major.Minor.Point) form.

uses-configuration node: specify each combination of input mechanisms are supported by your application.

```
<uses-configuration android:reqTouchScreen="finger"  
android:reqNavigation="trackball"  
android:reqHardKeyboard="true"  
android:reqKeyboardType="qwerty"/>
```

AndroidManifest.xml file

- **uses-feature node:** uses-feature nodes to specify which hardware features your application requires.

```
<uses-feature android:name="android.hardware.nfc" />
```

optional features: Audio, Bluetooth, Camera, Location, Microphone, NFC, Sensors, Telephony, TouchScreen, USB, Wi-Fi.

uses-library node: Used to specify a shared library that this application requires, for ex: the maps API.

```
<uses-library android:name="com.google.android.maps"  
    android:required="false" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworld2"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.helloworld2.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>
```


- 1. Creating Resources**
- 2. Using Resources**
3. Activity Life Cycle
4. Basic Views, Picker Views, List Views
5. Android Layouts

Creating Resources

- It's always good practice to keep non-code resources, such as images and string constants, external to your code.
- Android supports the externalization of resources such as strings, colors, images, animations, themes and menus.
- It supports internationalization and different screen size and resolution, languages.
- Android automatically selects correct resources based on the screen size, language etc..

Resource Types

- **anim folder:** XML files that define **View animations** (like changing size, position, rotation & transparency). They are saved in res/anim/ folder and accessed from the **R.anim** class.
- **Animator** → The **animator** resource directory can contain XML files that define **property animations** (change the property of Views: ex – translateX, TextScaleX) for View objects in your app.
- **Color** → (**colors.xml**) XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
- **Drawable** → The drawable folder holds all the images for your project. Android currently supports BMP, GIF, JPG, PNG, and WebP image formats. (**drawable animation** can be applied on the images of this folder).

Referring drawables:

“@drawable/filename”

- **Font** → The font folder holds any custom font files you wish to use. Android currently supports ttf, otf, ttc, and xml font files.

Referring Fonts in xml file:

“@font/filename”

Resource Types

- **Layout** → The layout folder holds all the XML layout files for your projects.
- **Menu** → XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.
- **Transition**
- **Values** → this folder contains **strings.xml**, **colors.xml**, **styles.xml** etc..

colors.xml → The colors file holds the hex codes for each color you would like to use in your app.

defining color values:

```
<color name="app_background">#FF0000FF</color>
```

usage:

for a view: **android:background="@color/colorPrimary"**

dimens.xml → The dimen file holds all the “measurements” for your layouts. You can create new dimensions by adding a new dimen element, giving it a name, and adding your desired dimension in sp for text and dp for everything else.

ex: `<dimen name="dim1">10px</dimen>`

usage: in xml file, you can refer as: “@dimen/dimename”

- **XML** → This directory houses arbitrary XML files used by Android for various tasks, such as defining [search configurations](#) or external capabilities, such as the use of [Android Auto](#).
- **mipmap** → The mipmap folder holds the icon image for your apps launcher.
- **raw** → The raw folder holds media files such as video and audio.

referring raw resource in java code:

R.raw.rawfilename

values folder

- XML files that contain simple values, such as strings, integers, and colors.
- **arrays.xml** for resource arrays, and accessed from the **R.array** class.
- **integers.xml** for resource integers, and accessed from the **R.integer** class.
- **bools.xml** for resource boolean, and accessed from the **R.bool** class.
- **colors.xml** for color values, and accessed from the **R.color** class.
- **dimens.xml** for dimension values, and accessed from the **R.dimen** class.
- **strings.xml** for string values, and accessed from the **R.string** class.
- **styles.xml** for styles, and accessed from the **R.style** class.
- **plurals.xml** → Similar to strings, [plurals](#) allows you to provide alternative strings when a number is passed into the retrieval function. This handles situations where a string quantifier should change based on the number of items it represents.

Simple Values

- Supported simple values include strings, colors, dimensions, styles, and string or integer arrays.
- All simple values are stored within XML files in the res/values folder.

Strings: `<string name="hello_world">Hello world!</string>`

Dimensions: `<dimen name="dim1">10px</dimen>`

integer-array:

```
<integer-array name="evennos">
  <item >10</item>
  <item >20</item>
  <item >30</item>
  <item >40</item>
</integer-array>
```

Simple Values

string-array:

Ex: `<string-array name="names">`
 `<item >Bapatla</item>`
 `<item >Guntur</item>`
 `<item >Chirala</item>`
`</string-array>`

Simple Values example

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">To Do List</string>
<plurals name="androidPlural">
<item quantity="one">One android</item>
<item quantity="other">%d androids</item>
</plurals>
<color name="app_background">#FF0000FF</color>
<dimen name="default_border">5px</dimen>
<string-array name="string_array">
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
</string-array>
<array name="integer_array">
<item>3</item>
<item>2</item>
<item>1</item>
</array>
</resources>
```


Strings

In strings.xml file : (res/values/strings.xml file)

```
<resources>  
  <string name="app_name">StylesThemesEx</string>  
  <string name="hello_world">Hello world!</string>  
  <string name="Theme1str">Theme1 Text</string>  
</resources>
```

In activity_main.xml file (res/layout/activity_main.xml file)

```
<TextView  
  android:id="@+id/textView1"  
  style="@style/SpecialText"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:text="@string/hello_world" />
```

Styles & Themes

- A **style** is a set of one or more formatting attributes that you can apply as a unit to single elements in your layout XML file(s).
ex: certain text size and color
- A **theme** is a set of one or more formatting attributes that you can apply as a unit to all activities in an application or just a single activity.
ex: colors for the window frame and the panel foreground and background, text sizes and colors for menus.

Defining and using styles

In `styles.xml` file : (res/values/styles.xml file)

```
<resources> <style name="SpecialText"    parent="@style/Text">
<item name="android:textSize">18sp</item>
<item name="android:textColor">#008</item> </style>
</resources>
```

In `activity_main.xml` file :

```
<TextView
    android:id="@+id/textView1"
    style="@style/SpecialText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

<Button

```
    android:id="@+id/button2"  
    android:layout_width="wrap_content"  
    android:layout_height="fill_parent"  
    android:layout_x="12dp"  
    android:layout_y="100dp"  
    android:text="@string/btn2str" />
```

wrap_content: width of the button will be same as that of its content.

fill_parent: height of the button occupies the entire height of the parent (screen).

```
<TextView
    android:id="@+id/tv_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:text="@string/title"
    style="@style/TitleStyle"/>
```

using Resources in java code:

```
getResources().getColor(R.color.colorPrimary);
getResources().getDrawable(R.drawable.kittens);
getDimension(),getFont(), getIntArray(), getInteger(),
getString(), getStringArray(), getValue(), getXml()
```

Defining and using themes

- Defining theme is done in styles.xml file as a style.
- A theme is applicable to an activity whole or all the activities of an App.

In styles.xml file :

```
<resources>  
<style name="CustomTheme">  
<item name="android:windowNoTitle">true</item>  
<item name="menuItemTextColor">?panelTextColor</item>  
<item name="menuItemTextSize">?panelTextSize</item> </style>  
</resources>
```

In AndroidManifest.xml file :

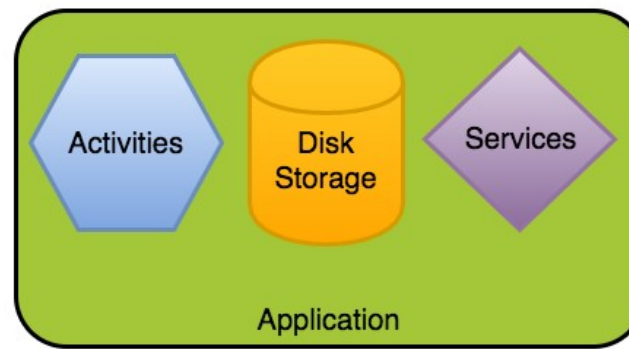
```
<application android:theme="@style/CustomTheme">
```

Android Application class



Android Application Class

- The Application class in Android is the base class within an Android app that contains all other components such as activities and services.
- This class is primarily used for initialization of global state before the first Activity is displayed.
- The Application class is mainly used for some Application level callbacks and for maintaining global Application state.
- The Application class, or any subclass of the Application class, is instantiated before any other class when the process for your application/package is created.
- Extending the Application class with your own implementation enables you to respond to application-level events broadcast by the Android run time (such as low memory conditions.)
- Note that custom Application objects should be used carefully and are often **not needed at all**.



Android Application Class

- Base class for maintaining global application state.
- When your Application implementation is registered in the manifest, it will be instantiated when your application process is created.
- You can provide your own implementation by creating a subclass and specifying the fully-qualified name of this subclass as the "android:name" attribute in your AndroidManifest.xml's <application> tag.
- The Application class, or your subclass of the Application class, is instantiated before any other class when the process for your application/package is created.

Maintaining global application state:

- Sometimes you want to store data, like global variables which need to be accessed from multiple Activities – sometimes everywhere within the application. In this case, the Application object will help you.
- For example, if you want to get the basic authentication data for each http request, you can implement the methods for authentication data in the application object.

After this, you can get the username and password **in any of the activities** like this:

```
MyApplication mApplication = (MyApplication)getApplicationContext();  
String username = mApplication.getUsername();  
String password = mApplication.getPassword();
```

Application Class methods

Methods (Application Level Callbacks)	Description
static String <code>getProcessName()</code>	Returns the name of the current process.
void <code>onConfigurationChanged(Configuration newConfig)</code>	Called by the system when the device configuration changes while your component is running.
void <code>onCreate()</code>	Called when the application is starting, before any activity, service, or receiver objects (excluding content providers) have been created.
void <code>onLowMemory()</code>	This is called when the overall system is running low on memory, and actively running processes should trim their memory usage.
void <code>onTerminate()</code>	This method is for use in emulated process environments. It will never be called on a production Android device, where processes are removed by simply killing them; no user code (including this callback) is executed when doing so.
void <code>onTrimMemory(int level)</code>	Called when the operating system has determined that it is a good time for a process to trim unneeded memory from its process.

Custom Application Class

In many apps, there's no need to work with an application class directly. However, there are a few acceptable uses of a custom application class:

- Specialized tasks that need to run before the creation of your first activity
- Global initialization that needs to be shared across all components (crash reporting, persistence)
- Static methods for easy access to static immutable data such as a shared network client object
- Note that you should never store mutable shared data inside the Application object since that data might disappear or become invalid at any time. Instead, store any mutable shared data using [persistence strategies](#) such as files, SharedPreferences or SQLite.
- However, you should never store mutable instance data inside the Application object because if you assume that your data will stay there, your application will inevitably crash at some point with a NullPointerException.
- **the app won't be restarted from scratch.** Android will create a new Application object and start the activity where the user was before to give the illusion that the application was never killed in the first place.
- If we do want a custom application class, we start by creating a new class which extends `android.app.Application` as follows:

Android Application Class

```
public class MyCustomAppl extends Application{  
    @override  
    public void onCreate(){ }  
    @override  
    public void onConfigurationChanged(){ }  
    @override  
    public void onLowMemory(){ }  
}
```

In the AndroidManifest.xml:

```
<application android:name=".MyCustomAppl"  
    android:icon="@drawable/icon"  
    android:label="@string/app_name" ...>
```

Activities

- Creating Activities**
- Activity Life cycle**
- Activity States**
- Android Activites**

Android Activity

- Each Activity represents a screen that an application can present to your users.
- The more complicated your application, the more screens you are likely to need.
- Typically, this includes at least a “main Activity” i.e the primary interface screen that handles the main UI functionality of your application.

Creating an activity :

- To create a new Activity, extend the Activity class—or one of its subclasses (most commonly the AppCompatActivity). Within your new class you must assign a UI and implement your functionality. The base Activity class presents an empty screen that encapsulates the window display handling.

Example:

```
package com.professionalandroid.apps.helloworld;
import android.app.Activity;
import android.os.Bundle;
public class MyActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}
```

Android Activity - setContentView()

- To assign a UI to an Activity, call setContentView from the onCreate method.
- In this next snippet, an instance of a TextView is used as the Activity's UI:

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    TextView textView = new TextView(this);  
    setContentView(textView);  
}
```

A complex layout can be assigned to an Activity by passing id of the layout to setContentView() method.

Example:

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

Android Activity - using Activity

- To use an Activity in your application, you need to register it in the manifest.
- Add a new activity tag within the application node of the manifest; the activity tag includes attributes for metadata, such as the label, icon, required permissions, and themes used by the Activity.

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

- An Activity without a corresponding activity tag can't be used—attempting to start it will result in a runtime exception:
- Within the activity tag you can add intent-filter nodes that specify the Intents that can be used to start your Activity. Each Intent Filter defines one or more actions and categories that your Activity supports.
- for an Activity to be available from the application launcher, it must include an Intent Filter listening for the MAIN action and the LAUNCHER category,

- ```
<activity android:label="@string/app_name"
```

- ```
  android:name=".MyActivity">
```

- ```
 <intent-filter>
```

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

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



## Android Activity - using Activity

- To use an Activity in your application, you need to register it in the manifest.
- Add a new activity tag within the application node of the manifest; the activity tag includes attributes for metadata, such as the label, icon, required permissions, and themes used by the Activity.

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

- An Activity without a corresponding activity tag can't be used—attempting to start it will result in a runtime exception:
- Within the activity tag you can add intent-filter nodes that specify the Intents that can be used to start your Activity. Each Intent Filter defines one or more actions and categories that your Activity supports.

```
<activity android:label="@string/app_name"
 android:name=".MyActivity">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
</activity>
```

# Activity Life Cycle

## States of an activity :

**Resumed/running state:** In this state, activity is in the foreground and the user can interact with it. Also called “running” state.

**Paused state:** In this state, the activity is partially obscured by another activity. The other activity does not cover full screen.

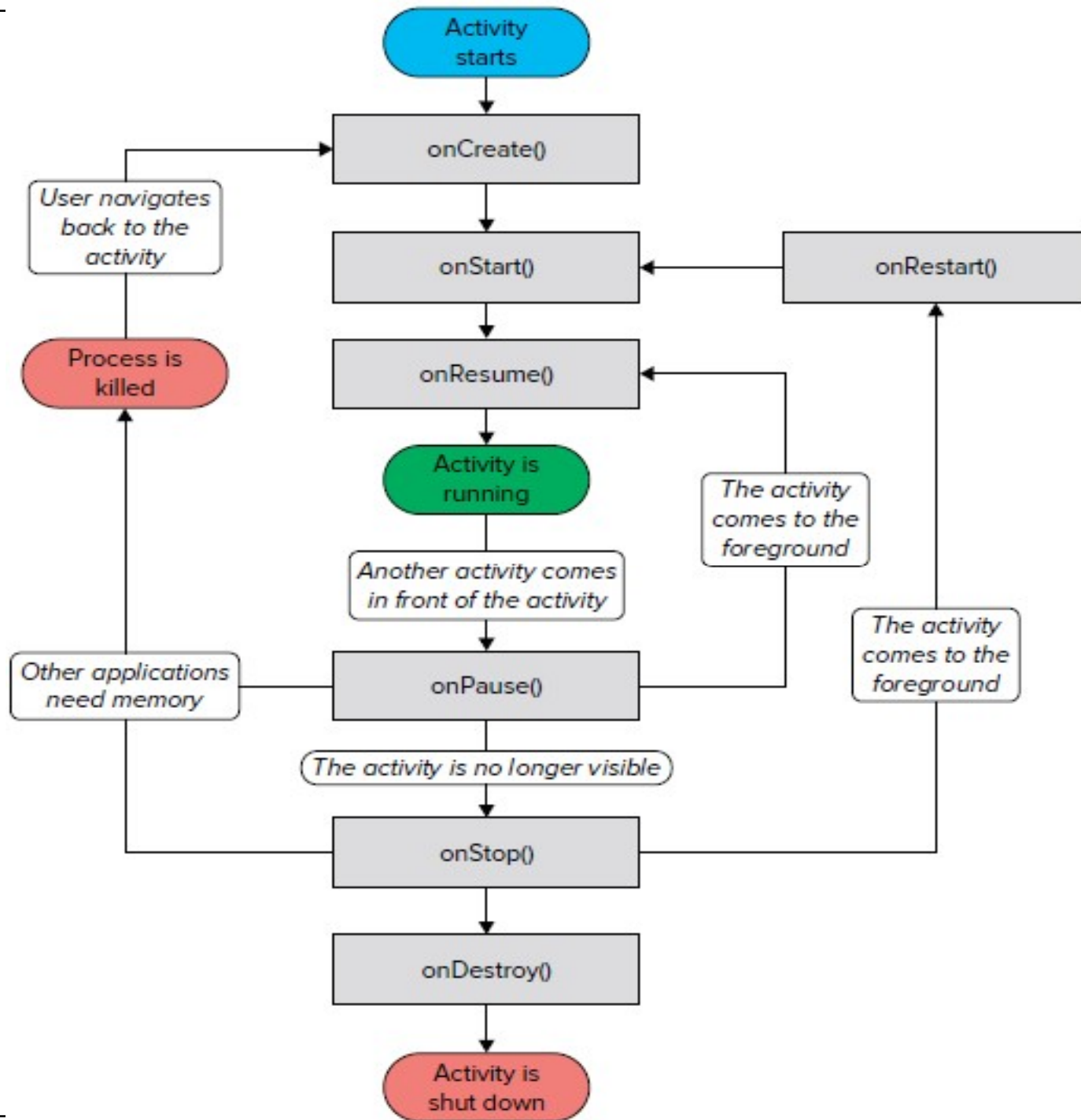
The paused activity does not receive user input and cannot execute any code.

**Stopped state:** In this state, the activity is completely hidden and not visible to the user. Activity information and all of its state ( variables ) is retained, but it cannot execute any code.

## Life cycle methods:

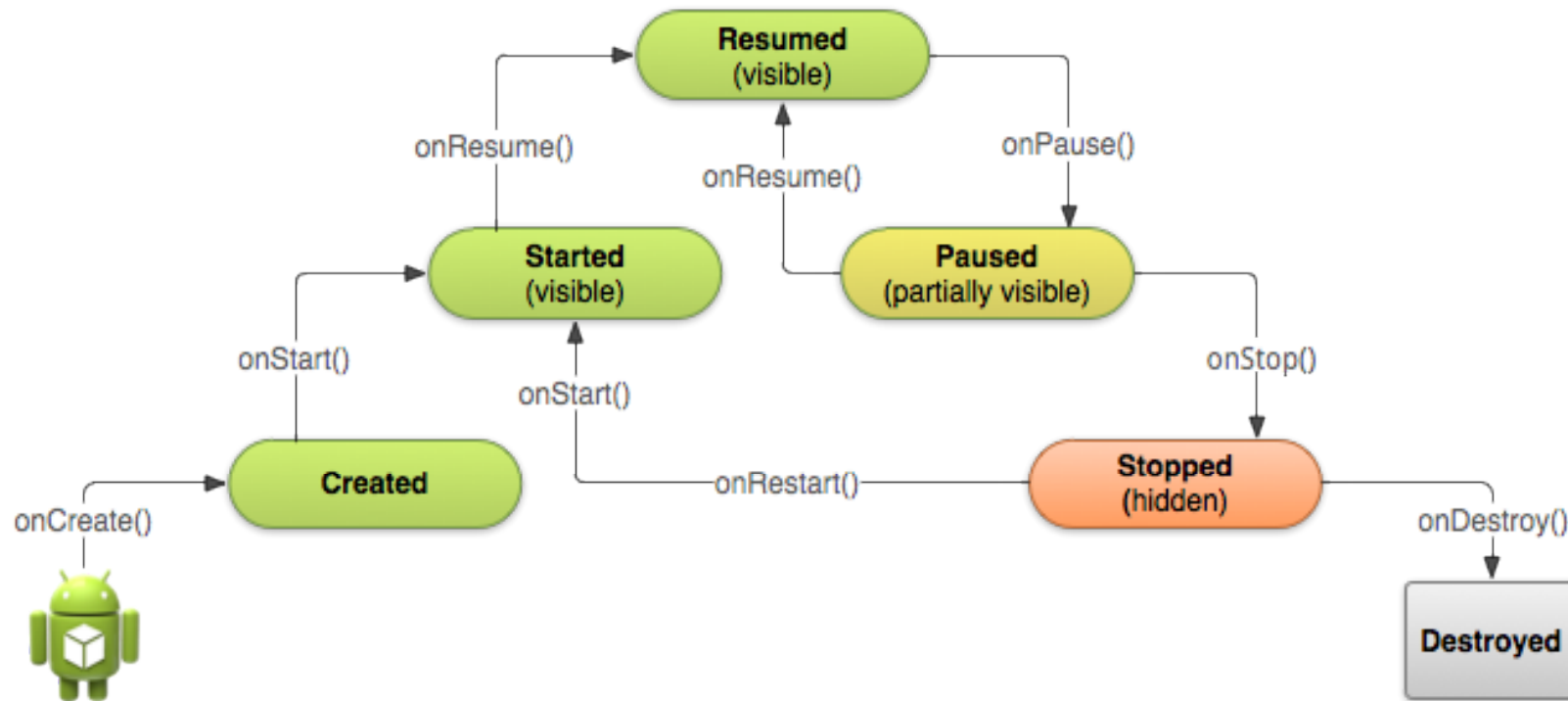
- Use the **onCreate()** method to create and instantiate the objects that you will be using in your application.
- **onStart()** - Called when the activity becomes visible but may not be in the foreground. You can perform tasks like registering listeners.
- Use the **onResume()** method to start any 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.
- **onStop()** - Called when the activity is no longer visible. Release resources and unregister listeners.

34 Use the **onDestroy()** method to free up resources before your activity is destroyed



## Activity Life Cycle

# Activity Life Cycle



## Android Activity State transitions

Case 1: When the activity is first loaded:

`onCreate()` → `onStart()` → `onResume()`

Case 2: When the back button is pressed:

`onPause()` → `onStop()` → `onDestroy()`

Case 3: When the **HOME** button is pressed:

`onPause()` → `onStop()`

Case 2: When the application is started again:

`onRestart()` → `onStart()` → `onResume()`

## scenerios

When HOME buton is pressed :

onPause( )

onSaveInstanceState( )

onStop( )

When BACK button is pressed:

onPause( )

onStop( )

onDestroy( )

## Method call sequence in orientation change

Press Ctrl F12 after entering text in EditText controls.

**Sequence of method calls:**

`onPause()`

`onSaveInstanceState()`

`onStop()`

`onDestroy()`

`onCreate()`

`onRestoreInstanceState()`

`onResume()`



**Basic Views, Picker Views, List Views  
Android Layouts**



# Views & View Groups

## View:

- An activity contains views and view groups.
- A view is a widget that has an appearance on screen.
  - Examples of views are buttons, labels, and text boxes. A view derives from the base class `android.view.View`.

## ViewGroup:

- One or more views can be grouped together into a ViewGroup. A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views.
- Examples of ViewGroups include `LinearLayout` and `FrameLayout`.
- A ViewGroup derives from the base class `android.view.ViewGroup`.

# Views in Android

## 1. Basic Views

- TextView, EditText, Button, ImageButton, CheckBox, ToggleButton, RadioButton, RadioGroup, ProgressBar, AutoCompleteTextView

## 2. Picker Views

- DatePicker, TimePicker

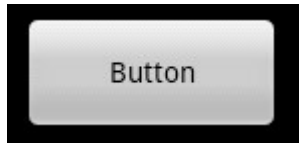
## 3. List Views

## Basic views

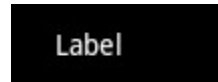
- TextView
- EditText
- Button
- ImageButton
- CheckBox
- ToggleButton
- RadioButton
- RadioGroup
- ProgressBar
- AutoCompleteTextView

# Basic UI Controls

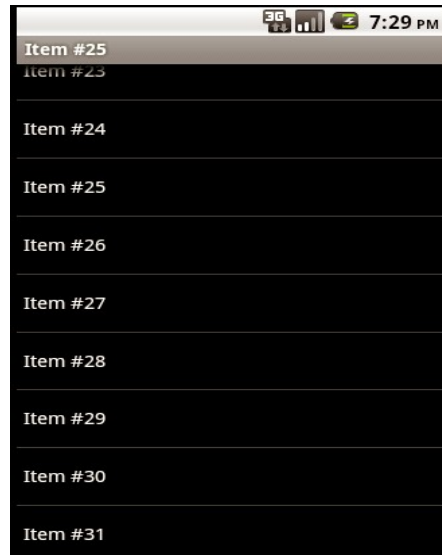
Button:



Label :



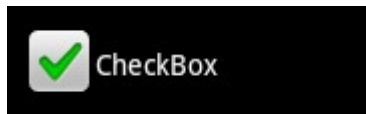
ListView (ListBox)



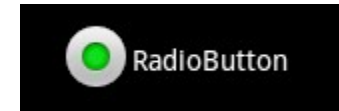
ProgressBar



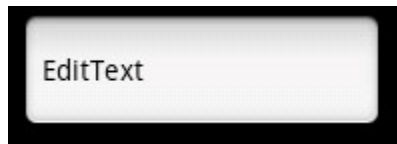
Checkbox:



RadioButton



EditText (TextBox):



ScrollView

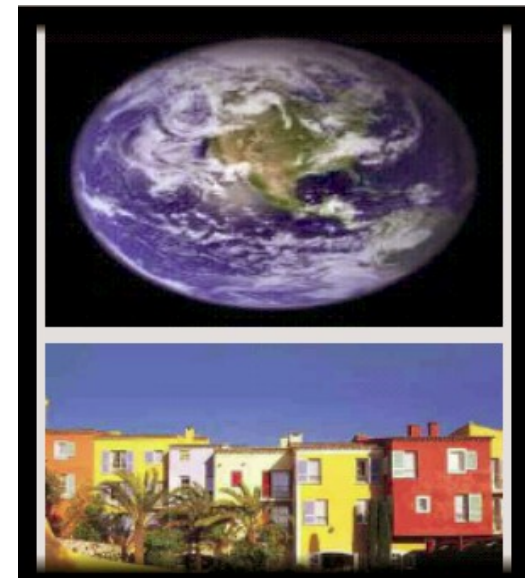


Image View:



Panel



# Basic UI Controls

## SeekBar (TrackBar)



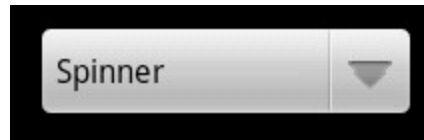
## ToggleButton



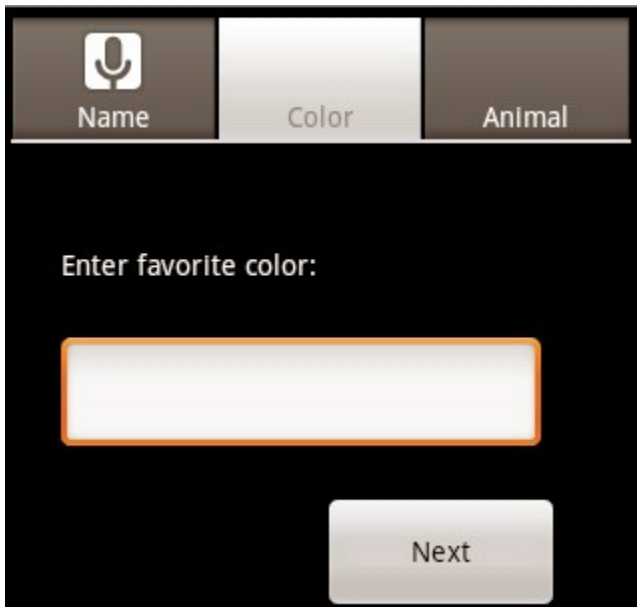
## WebView



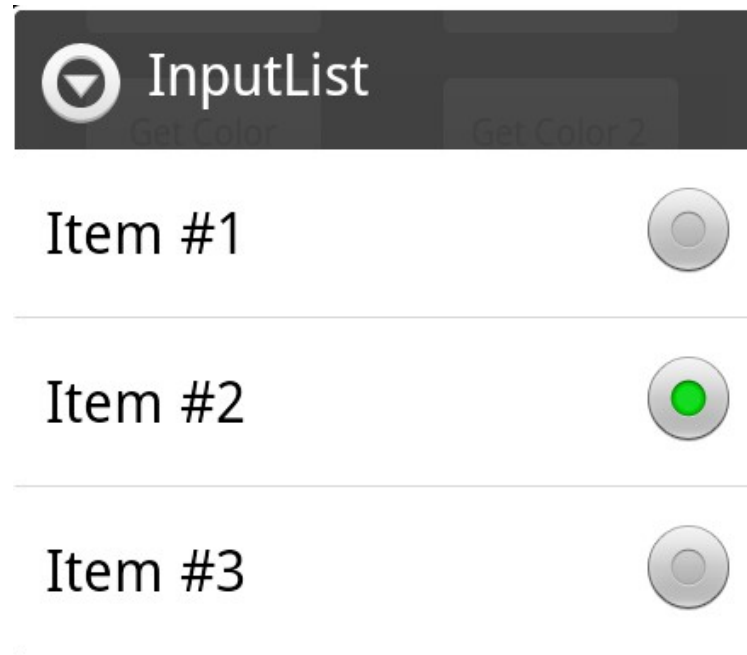
## Spinner (ComboBox)



## TabHost (TabControl)

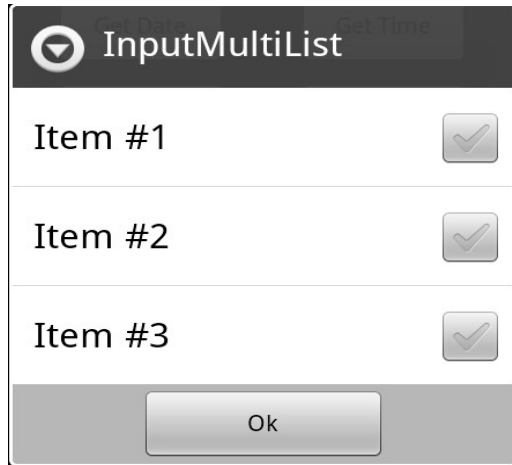


## InputList



# Basic views

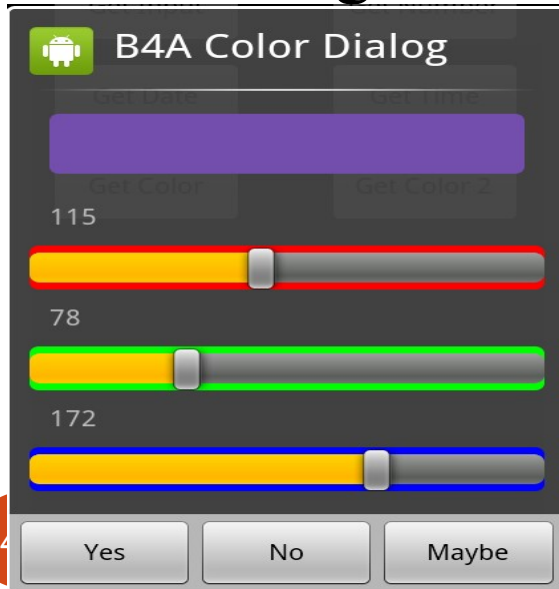
## InputMultiList



## MsgBox



## ColorDialog



## ColorPickerDialog

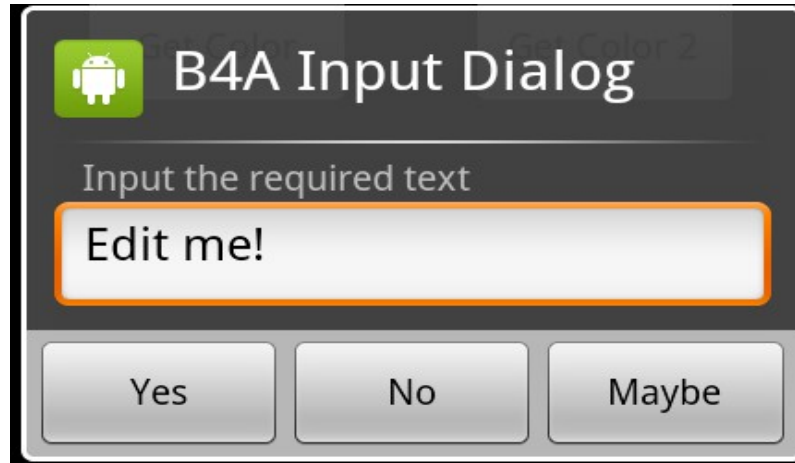


# Basic views

## DateDialog



## InputDialog



## TimeDialog

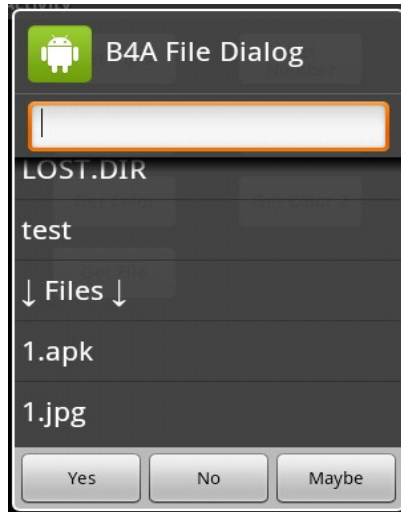


## NumberDialog



# Basic views

## FileDialog



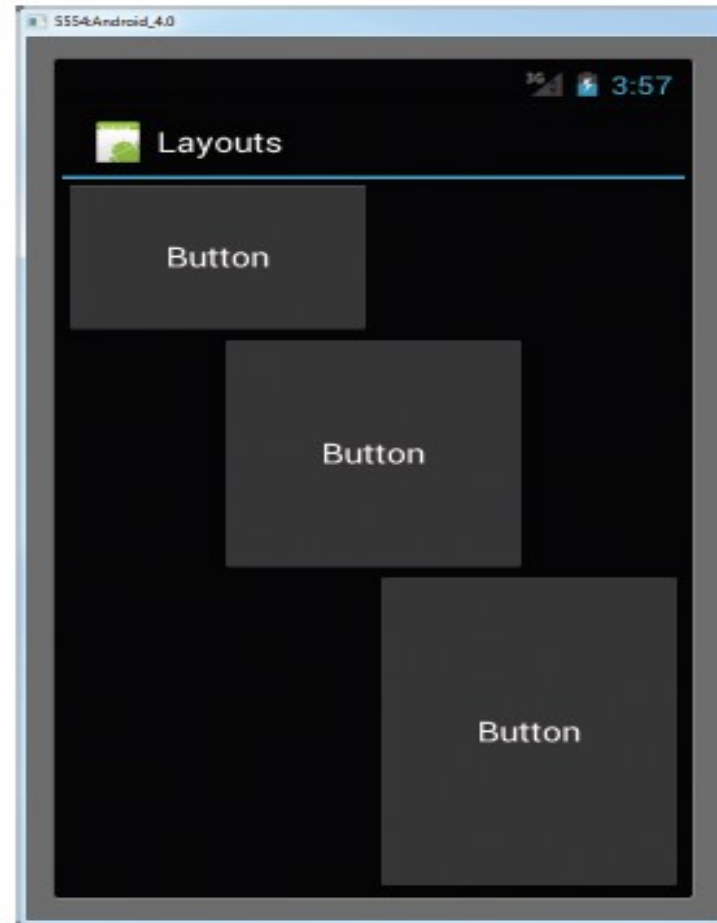


## Attributes in the Views and ViewGroups

- layout\_width** → Specifies the width of the View or ViewGroup
- layout\_height** → Specifies the height of the View or ViewGroup
- Layout\_marginTop** → Specifies extra space on the top side of the View or ViewGroup
- Layout\_marginBottom**
- layout\_marginLeft**
- layout\_marginRight**
- layout\_x** → Specifies the x-coordinate of the View or ViewGroup
- layout\_y** → Specifies the y-coordinate of the View or ViewGroup
- layout\_gravity** → attribute indicates the positions the views should gravitate towards.
- layout\_weight** → attribute specifies the distribution of available space

## layout\_gravity & layout\_weight

```
<Button
android:layout_width="160dp"
android:layout_height="wrap_content"
android:text="Button"
android:layout_gravity="left"
android:layout_weight="1" />
<Button
android:layout_width="160dp"
android:layout_height="fill_parent"
android:text="Button"
android:layout_gravity="center"
android:layout_weight="2" />
<Button
android:layout_width="160dp"
android:layout_height="wrap_content"
android:text="Button"
android:layout_gravity="right"
android:layout_weight="3" />
```



## Button event handling

1. get the control by using its id.

```
Button btnOpen = (Button) findViewById(R.id.btnOpen);
```

2. Register Listener to the control and add handler implementation.

```
btnOpen.setOnClickListener(new View.OnClickListener() {
 public void onClick(View v) {
Toast.makeText(getBaseContext(), "clicked on Open
button", Toast.LENGTH_SHORT).show();
 }
});
```

## Picker Views

- DatePicker → for selecting date.
- TimePicker → for selecting time.

## Picker Views - TimePicker

- In Android, [TimePicker](#) is a widget used for selecting the time of the day in either AM/PM mode or 24 hours mode. The displayed time consist of hours, minutes and clock format.
- If we need to show this view as a Dialog then we have to use a **TimePickerDialog** class.

### Methods:

| Method                                                                                 | Description                                                                                                                                                 |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>setHour(Integer hour)</b>                                                           | set the value for hours.                                                                                                                                    |
| <b>setMinute(Integer minute)</b>                                                       | sets the value for minutes.                                                                                                                                 |
| <i>getHour()</i>                                                                       | we get the value of hours from a <a href="#">timepicker</a>                                                                                                 |
| <i>getMinute()</i>                                                                     | get the value of minutes from a time picker.                                                                                                                |
| <b>setIs24HourView(Boolean is24HourView)</b>                                           | set the mode of the Time picker either 24 hour mode or AM/PM mode. True → indicates 24 hour mode, false → indicates AM/PM mode                              |
| <b>is24HourView()</b>                                                                  | This method returns true if its 24 hour mode or false if AM/PM mode is set.                                                                                 |
| <b>setOnTimeChangeListener((TimePicker.OnTimeChangeListener onTimeChangeListener))</b> | set the callback that indicates the time has been adjusted by the user.<br>onTimeChanged(TimePicker view, int hourOfDay, int minute) is the handler method. |

# Picker Views - TimePicker

## Attributes:

**timePickerMode:** time picker mode is an attribute of time picker used to set the mode either spinner or clock. Default mode is clock but this mode is no longer used after api level 21, so from api level 21 you have to set the mode to spinner.

# Picker Views - DatePicker

- In Android, **DatePicker** is a control that will allow users to select the date by a day, month and year in our application user interface.
- If we use **DatePicker** in our application, it will ensure that the users will select a valid date.

## DatePicker with calendar Mode:

<DatePicker

```
 android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:datePickerMode="calendar"/>
```



## DatePicker with spinner mode:

<DatePicker

```
 android:id="@+id/datePicker1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:datePickerMode="spinner"/>
```



## Picker Views - DatePicker

| Attribute              | Description                                                                      |
|------------------------|----------------------------------------------------------------------------------|
| android:id             | It is used to uniquely identify the control                                      |
| android:datePickerMode | It is used to specify datepicker mode either spinner or calendar                 |
| android:background     | It is used to set the background color for the date picker.                      |
| android:padding        | It is used to set the padding for left, right, top or bottom of the date picker. |



## DatePicker

int getMonth( )

int getDayOfMonth( )

int getYear( )

boolean isEnabled( )

void setEnabled(boolean enabled)

void setFirstDayOfWeek(int day) → sets the first day of week

# AutoCompleteTextView

- **Android AutoCompleteTextView** completes the word based on the reserved words, so no need to write all the characters of the word.
- Android AutoCompleteTextView is a editable text field, it displays a list of suggestions in a drop down menu from which user can select only one suggestion or value.
- Android AutoCompleteTextView is the subclass of EditText class. The MultiAutoCompleteTextView is the subclass of AutoCompleteTextView class.

# Layouts

- **LinearLayout:** Aligns a sequence of child Views in either a vertical or a horizontal line.
- **RelativeLayout:** Define the position of each element within the layout in terms of its parent and the other Views.
- **GridLayout:** The Grid Layout uses an arbitrary grid to position Views. By using row and column spanning, the Space View, and Gravity attributes, you can create complex without resorting to the often complex nesting required to construct UIs using the Relative Layout.
- **FrameLayout:** The FrameLayout is a placeholder on screen that you can use to display a single view. Views that you add to the FrameLayout are always anchored to the top left of the layout.
- **TableLayout:** Groups Views into rows and columns. Use `<TableRow>` to designate a row in the table.
- **ScrollView:** ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display.

# LinearLayout

The LinearLayout arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally.

```
<LinearLayout
```

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

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

```
android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
```

```
>
```

```
<TextView
```

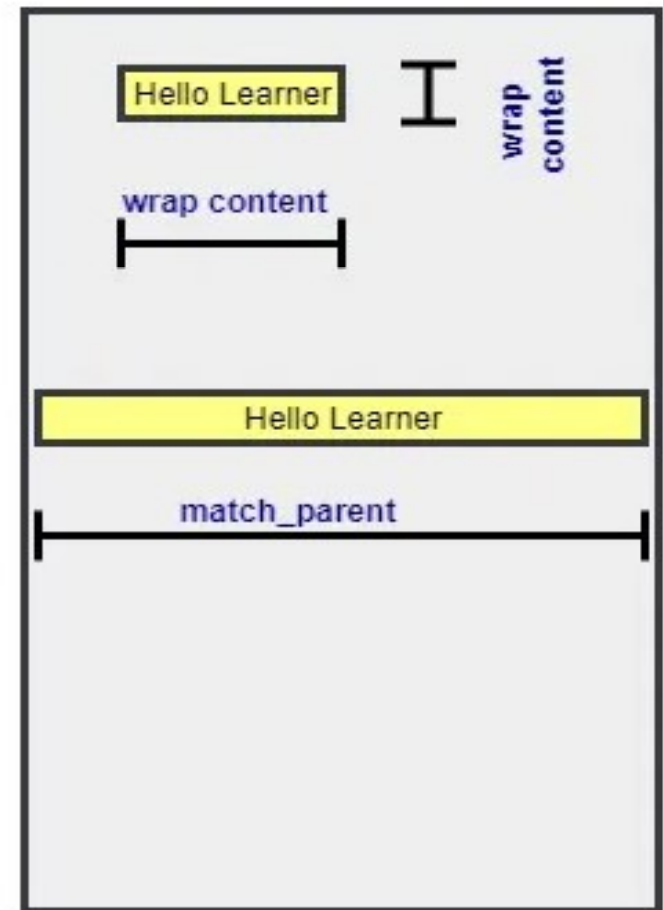
```
android:layout_width="fill_parent"
```

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

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

```
/>
```

```
</LinearLayout>
```



# AbsoluteLayout

- ✓ The `AbsoluteLayout` enables you to specify the exact location of its children

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

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

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

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

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

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

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

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

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

```
<Button
```

```
android:id="@+id/button1"
```

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

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

```
android:layout_marginLeft="17dp"
```

```
android:layout_marginTop="20dp"
```

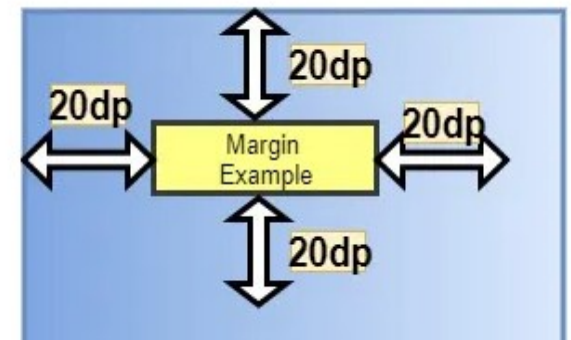
```
android:layout_x="126dp"
```

```
android:layout_y="10dp"
```

```
android:text="@string/btn1str" />
```

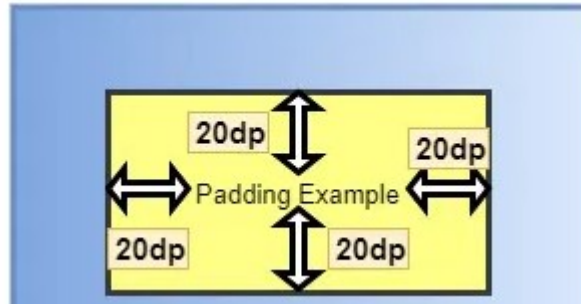
```
</AbsoluteLayout>
```

```
android:layout_marginRight="20dp"
android:layout_marginLeft="20dp"
android:layout_marginTop="20dp"
android:layout_marginBottom="20dp"
```



# Padding

Padding can be considered as margin but inside the View.



## TableLayout

- The TableLayout groups views into rows and columns. You use the `<TableRow>` element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell.

## RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other.

### Attributes for Views:

Attribute	Description
android:layout_above	Positions the bottom edge of this view above the given anchor view ID.
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID.
android:layout_alignEnd	Makes the end edge of this view match the end edge of the given anchor view ID.
android:layout_alignLeft	Makes the left edge of this view match the left edge of the given anchor view ID.
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent.
android:layout_alignParentEnd	If true, makes the end edge of this view match the end edge of the parent.
android:layout_alignParentLeft	If true, makes the left edge of this view match the left edge of the parent.



## Attributes for Views in relative layout

Attribute	Description
<code>android:layout_alignParentRight</code>	If true, makes the right edge of this view match the right edge of the parent.
<code>android:layout_alignParentStart</code>	If true, makes the start edge of this view match the start edge of the parent.
<code>android:layout_alignParentTop</code>	If true, makes the top edge of this view match the top edge of the parent.
<code>android:layout_alignRight</code>	Makes the right edge of this view match the right edge of the given anchor view ID.
<code>android:layout_alignStart</code>	Makes the start edge of this view match the start edge of the given anchor view ID.
<code>android:layout_alignTop</code>	Makes the top edge of this view match the top edge of the given anchor view ID.

## Attributes for Views in relative layout

Attribute	Description
<code>android:layout_centerVertical</code>	If true, centers this child vertically within its parent.
<code>android:layout_below</code>	Positions the top edge of this view below the given anchor view ID.
<code>android:layout_centerHorizontal</code>	If true, centers this child horizontally within its parent.
<code>android:layout_centerInParent</code>	If true, centers this child horizontally and vertically within its parent.

## FrameLayout

- The FrameLayout is a placeholder on screen that you can use to display a single view. Views that you add to a FrameLayout are always anchored to the top left of the layout.

# ScrollView

1. pickerView, ListViews, autoCompleteTextView
2. Menus, ProgressBar, SeekBar, CalendarView

## ProgressBar

- The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.

Ex: downloading files/movies.

## AutoCompleteTextView

- The AutoCompleteTextView is a view that is similar to EditText (in fact it is a subclass of EditText), except that it shows a list of completion suggestions automatically while the user is typing.

```
<AutoCompleteTextView
android:id="@+id/simpleAutoCompleteTextView" android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#000"
android:hint="Enter Your Name Here"
android:padding="15dp"
android:textColorHint="#fff"
android:textStyle="bold|italic" />
```

# Menus

- **Options menu** - Displays information related to the current activity. In Android, you activate the options menu by pressing the MENU key.
- **Context menu** - Displays information related to a particular view on an activity. In Android, to activate a context menu you tap and hold on to it.



# OptionsMenu

- **Options menu** - Displays information related to the current activity. In Android, you activate the options menu by pressing the MENU key.
- You Need to override two methods i) `onCreateOptionsMenu( )` and `onOptionsItemSelected( )` methods.
- `onCreateOptionsMenu( )` is called when you pressed MENU button.
  - In this method, call helper method to display options menu.
- When a menu item is selected, `onOptionsItemSelected( )` method is called.

## **MenuItem class:**

**add( ) method:** `add(int groupId, int itemId, int order, String title)`

**groupId** → group id of the menuitem

**itemId** → unique item ID

**Order** → the order in which the item should be displayed

**Title** → the text to display for the menu item

Menu mnu;

Ex: `MenuItem mnu1=mnu.add(0,0,0,"item 1");`

`mnu.setAlphabeticshortcut('a');`

`mnu.setIcon(R.drawable.icon);`

- `getItemId( )` method of **MenuItem** is used to determine the **MenuItem** selected.

## ContextMenu

- A context menu is usually associated with a view on an activity, and it is displayed when the user taps and holds an item.
- For example, if the user taps on a Button view and holds it for a few seconds, a context menu can be displayed.
- If you want to associate a context menu with a view on an activity, you need to call the **setOnCreateContextMenuListener()** method of that particular view.<sup>7</sup>
- When the user taps and holds the Button view, the **onCreateContextMenu()** method is called.
- when an item inside the context menu is selected, the **onContextItemSelected()** method is called.

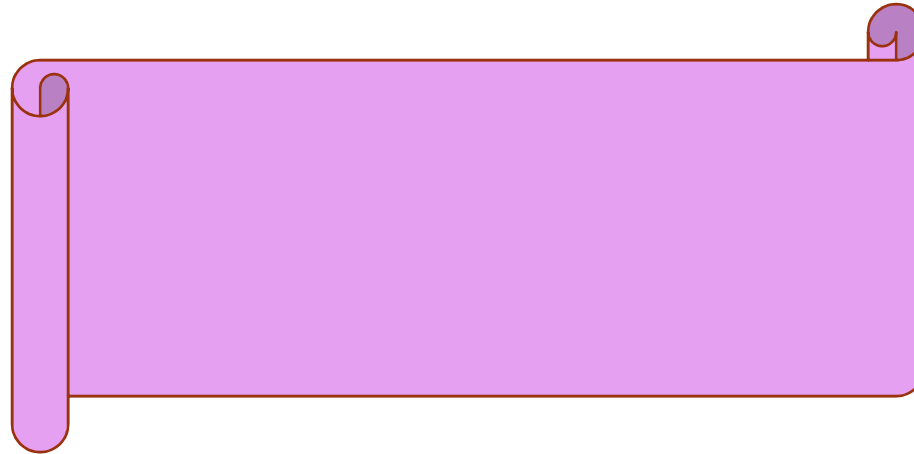
# Context Menu

- Displays information related to a particular view on an activity.
- To activate a context menu you tap and hold on to it for a few seconds.
- To associate a context menu with a view, call `setOnCreateContextMenuListener()` method of that particular view.

Ex: `btn.setOnCreateContextMenuListener(this);`

- `onCreateContextMenu()` is called whenever you tap and hold a view for a few seconds.
  - In this method call `CreateMenu()` method to create a context menu.
  - An item selection from context menu calls `onContextItemSelected()` method.

# Application Class



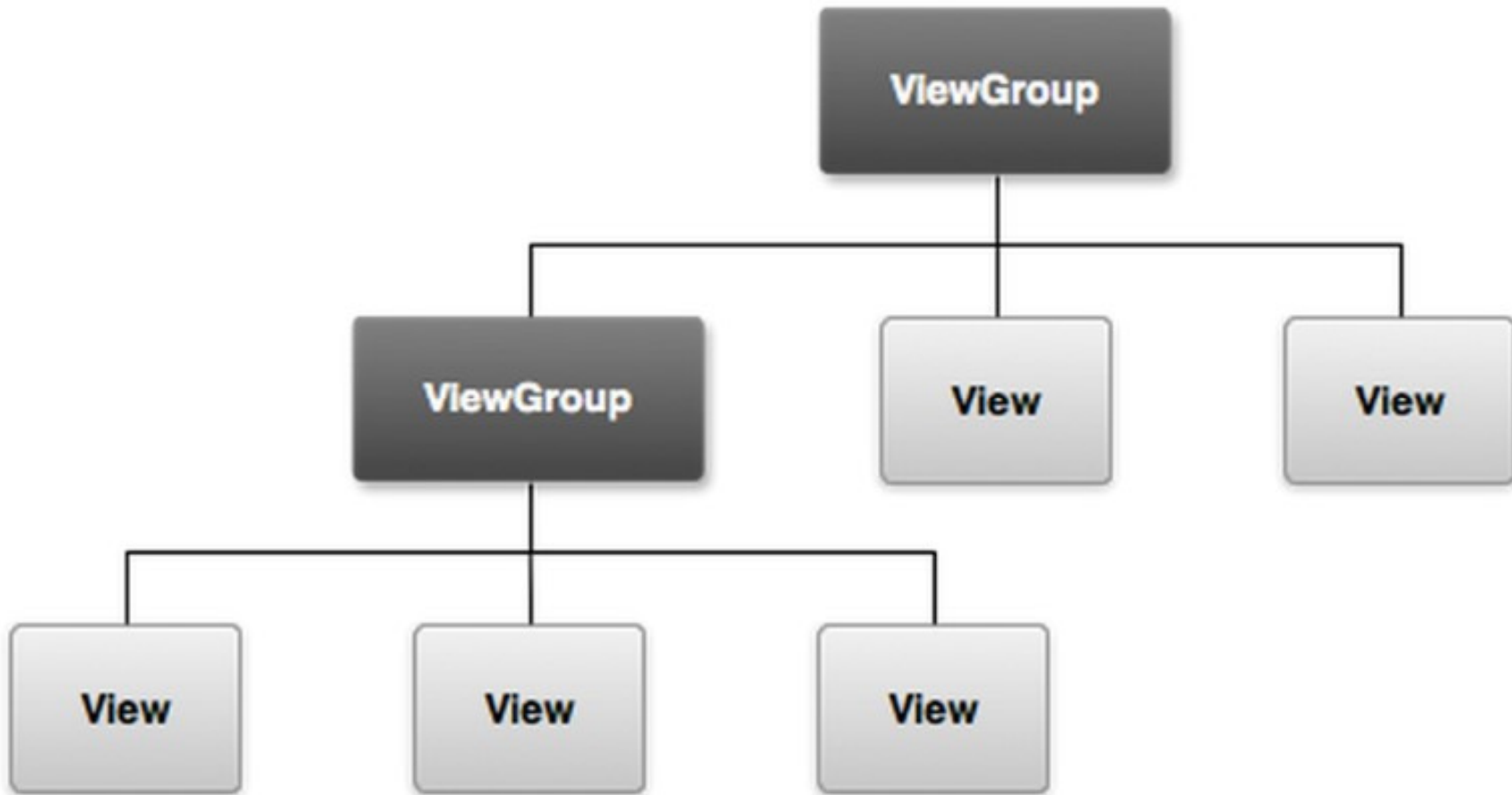
1. ViewGroups
2. Listviews
3. Adapting to Display orientation,
4. Managing Changes to Screen Orientation
5. using Adapters.

- 1.

# ViewGroups

- A ViewGroup is an invisible container for one or more views.
- A Layout is also a ViewGroup
- Examples for ViewGroups are: ViewGroup, ScrollView and all Layouts.
- A ViewGroup derives from `android.view.ViewGroup`

# ViewGroups



## Adapting to Display Orientation

- Android supports two orientations: 1. Portrait  
2. Landscape
  - When you change the orientation of the device, your current activity is actually destroyed and then re-created.

Two techniques for handling display orientations:

- 1) Anchoring → fixing the views to the four edges of the screen. When the screen orientation changes, the views can anchor to the edges neatly.
- 2) Resizing & Repositioning → Resizing and Repositioning each and every view according to the current screen orientation. This is the effective technique.



## Controlling orientation of Activity

- setting the screen orientation to landscape:

### 1) In onCreate() method:

```
//---change to landscape mode---
```

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

**Or**

### 2) in AndroidManifest.xml file:

```
<activity
 android:screenOrientation="landscape">
</activity>
```

## Detecting orientation changes

- in `onCreate()` method:

`@Override`

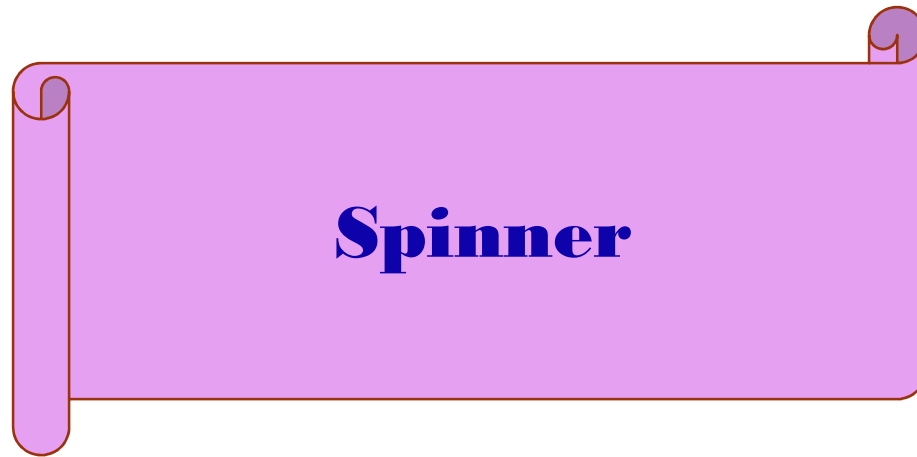
```
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 if(getResources().getConfiguration().orientation ==
 Configuration
 .ORIENTATION_LANDSCAPE) {
 Log.d("StateInfo", "Landscape");
 } else if(getResources().getConfiguration().orientation == Configuration
 .ORIENTATION_PORTRAIT) {
 Log.d("StateInfo", "Portrait");
 }
}
```

# **List Views: ListView & Spinner**



## List View

- List views are views that enable you to display a long list of items.
- Two types of List Views → ListView & SpinnerView
- Spinner view displays a single item from the list at a time.
- List view displays more items at a time.



# Spinner

- In Android, Spinner provides a quick way to select one value from a set of values.
- Android spinners are nothing but the drop down-list seen in other programming languages. In a default state, a spinner shows its currently selected value.
- It provides a easy way to select a value from a list of values.
- In Simple Words we can say that a spinner is like a combo box of AWT or swing where we can select a particular item from a list of items.
- Spinner is associated with Adapter view so to fill the data in spinner we need to use one of the Adapter class
- To fill the data in a spinner we need to implement an adapter class.
- A spinner is mainly used to display only text field so we can implement Array Adapter for that.
- We can also use Base Adapter and other custom adapters to display a spinner with more customize list. Suppose if we need to display a textview and a imageview in spinner item list then array adapter is not enough for that.
- Whenever any item from the spinner is selected, **onItemSelected()** method is executed.

# Spinner

## ArrayAdapter:

- An adapter is a bridge between UI component and data source that helps us to fill data in UI component.
- It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like as [list view](#), [grid view](#), spinner.
- Whenever you have a list of single items which is backed by an array, you can use Array Adapter.

- **AutoCompleteTextView**
- **SeekBar**
- **ProgressBar**
- **ListViews – ListView, Spinner**
- **GridView**
- **Toolbar**
- **Externalizing Resources**
- **Application Components**
- **RecyclerView**
- **VideoView**
- **ViewPager**
- **PickerViews – TimePicker, DatePicker**

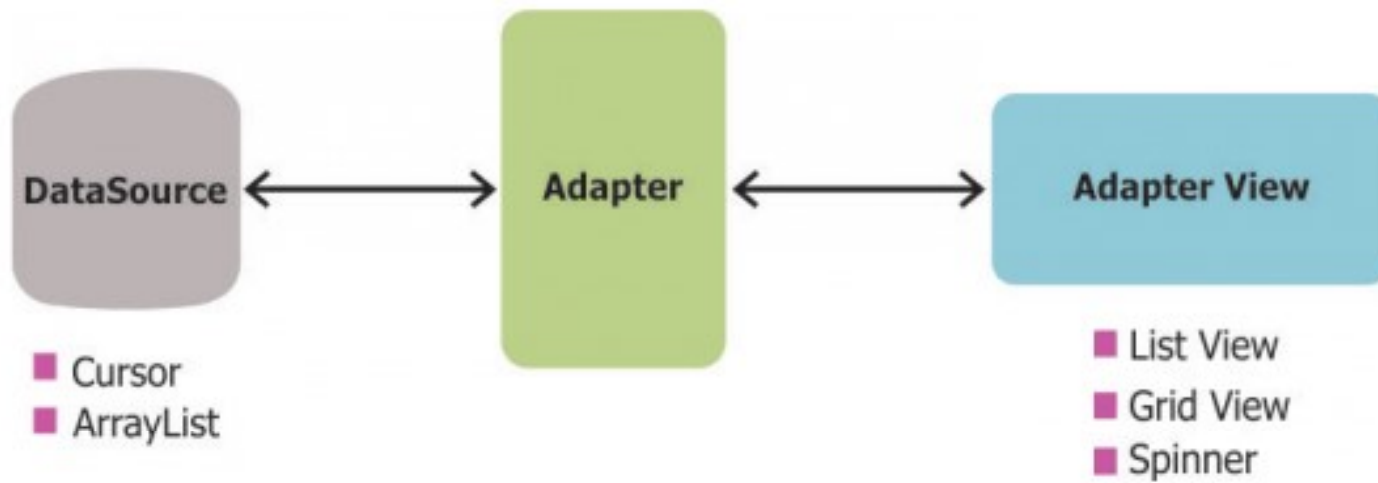




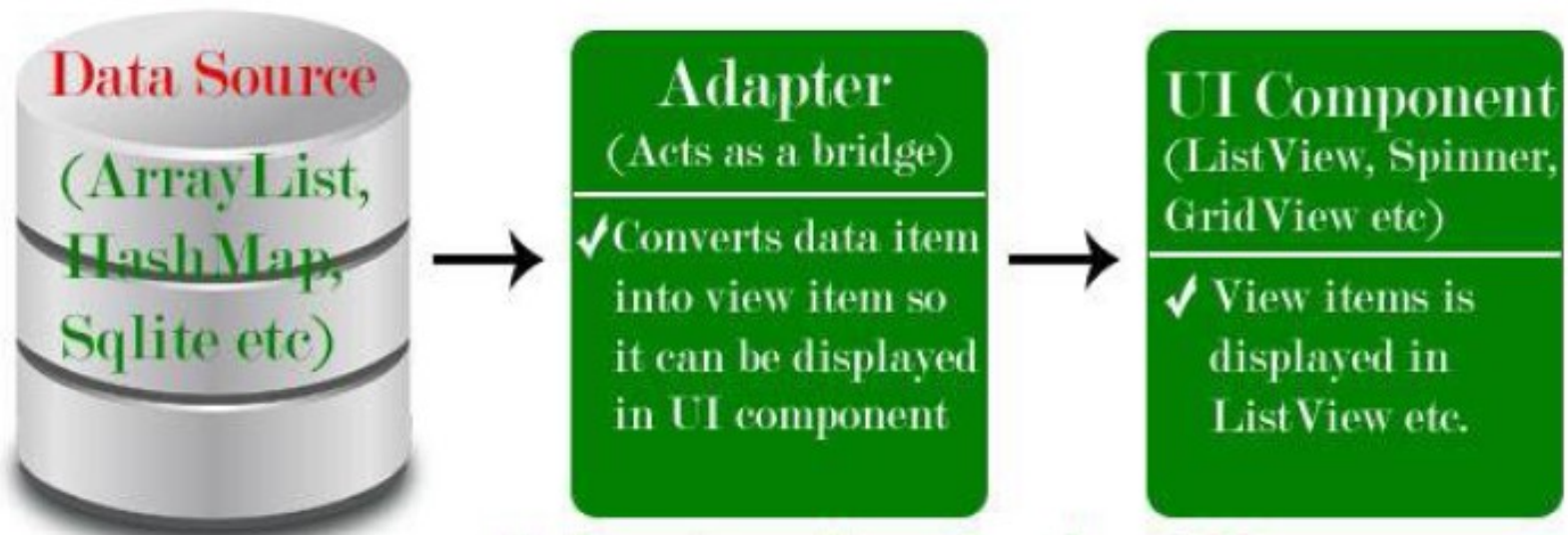
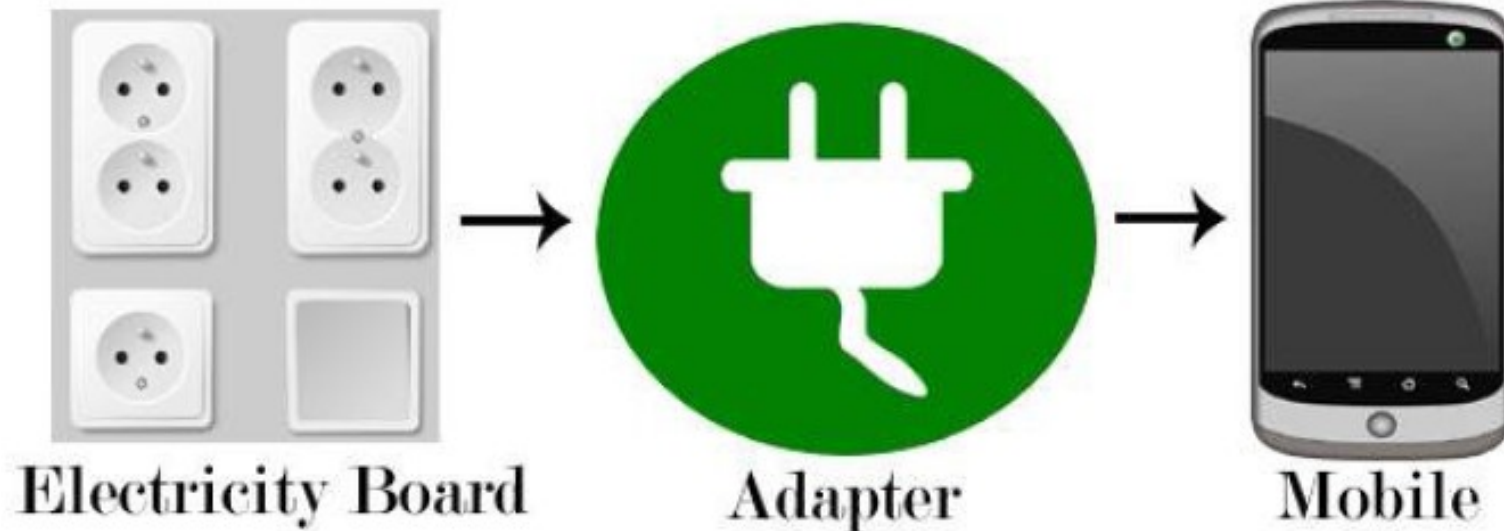
## Displaying Pictures and Menus

- ImageView
- Gallery
- GridView
- ImageSwitcher
- The Gallery view together with an ImageView to display a series of thumbnail images so that when one is selected, the selected image is displayed in the ImageView.
- ImageSwitcher when used with Gallery view provides some animation to the image ( fade-in, fade-out)

# Adapter



# Adapter



# Adapters

- **BaseAdapter** → It is parent adapter for all other adapters
- **SimpleAdapter** → In Android SimpleAdapter is an easy Adapter to map static data to views defined in an XML file(layout).
- **ListAdapter**
- **ArrayAdapter** → It is used whenever we have a list of single items which is backed by an array
- **CursorAdapter**
- **SpinnerAdapter**
- **ImageAdapter**
  
- **Custom Array Adapter** → It is used whenever we need to display a custom list
- **Simple Adapter** → It is an easy adapter to map static data to views defined in your XML file
- **Custom Simple Adapter** → It is used whenever we need to display a customized list and needed to access the child items of the list or grid
  
- You create the **ImageAdapter** class (which extends the BaseAdapter class) so that it can bind to the Gallery view with a series of ImageView views.
- The **BaseAdapter** class acts as a bridge between an AdapterView and the data source that feeds data into it.

## Adapter Views:

- ListView
- GridView
- Spinner
- Gallery

## SimpleAdapter

- In Android SimpleAdapter is an easy Adapter to map static data to views defined in an XML file(layout).
- You also specify an XML file that defines the views used to display the row, and a mapping from keys in the Map to specific views.

### Constructor:

```
public SimpleAdapter (Context context, List<? extends Map<String, ?>>
data, int resource, String[] from, int[] to)
```

**Where:** **Context:** The context where the View associated with this SimpleAdapter is running

**List:** A List of Maps. Each entry in the List corresponds to one row in the list. The Maps contain the data for each row, and should include all the entries specified in "from".

**resource:** Resource identifier (id) of a view layout that defines the views for this list item. The layout file should include at least those named views defined in "to" .

**from:** A list of column names (strings) that will be added to the Map associated with each item.

**to:** The views that should display column in the "from" parameter. These should all be TextViews.

The first N views in this list are given the values of the first N columns in the from parameter.

## SpinnerAdapter

- Extended [Adapter](#) that is the bridge between a [Spinner](#) and its data. A spinner adapter allows to define two different views: one that shows the data in the spinner itself and one that shows the data in the drop down list when the spinner is pressed.

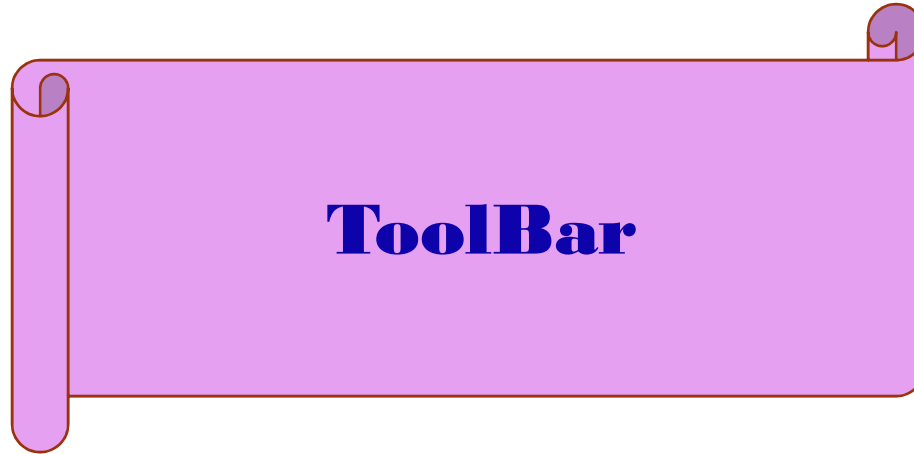


# **GridView**

# GridView

- The GridView shows items in a two-dimensional scrolling grid. You can use the GridView together with an ImageView to display a series of images.







# **ProgressBar**

# ProgressBar

- In Android, ProgressBar is used to display the status of work being done like analyzing status of work or downloading a file etc.
- In Android, by default a progress bar will be displayed as a spinning wheel but If we want it to be displayed as a horizontal bar then we need to use style attribute as horizontal.
- It mainly use the “**android.widget.ProgressBar**” class.
- To add a progress bar to a layout (xml) file, you can use the <ProgressBar> element. By default, a progress bar is a spinning wheel (an indeterminate indicator).
- To change to a horizontal progress bar, apply the progress bar’s horizontal style.
- **Important Methods Used In ProgressBar:**
- **1. getMax()** – returns the maximum value of progress bar
- We can get the maximum value of the progress bar in java class. This method returns a integer value. Below is the code to get the maximum value from a Progress bar.
- // initiate the progress bar
- `ProgressBar simpleProgressBar=(ProgressBar) findViewById(R.id.simpleProgressBar);`  
`int maxValue=simpleProgressBar.getMax(); // get maximum value of the progress bar`

# ProgressBar

- **2. getProgress()** – returns current progress value
- We can get the current progress value from a progress bar in `java` class. This method also returns an integer value. Below is the code to get current progress value from a Progress bar.
- ```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar); // initiate the progress bar
int progressValue=simpleProgressBar.getProgress(); // get progress value from the progress bar.
```
- **Attributes of ProgressBar In Android:**
- **1. id:** id is an attribute used to uniquely identify a Progress bar.
- **2. max:** max is an attribute used in android to define maximum value of the progress can take. It must be an integer value like 100, 200 etc.
- **3. progress:** progress is an attribute used in android to define the default progress value between 0 and max. It must be an integer value.
- **4. progressDrawable:** progress drawable is an attribute used in Android to set the custom drawable for the progress mode.
- **5. background:** background attribute is used to set the background of a Progress bar. We can set a color or a drawable in the background of a Progress bar.
- **6. indeterminate:** indeterminate attribute is used in Android to enable the indeterminate mode. In this mode a progress bar shows a cyclic animation without an indication of progress. This mode is used in application when we don't know the amount of work to be done. In this mode the actual working will not be shown. `android:indeterminate="true"`

ProgressBar

- To display horizontal styled ProgressBar:

```
<ProgressBar android:id="@+id/simpleProgressBar"  
    android:layout_width="fill_parent" android:layout_height="wrap_content"  
    style="@style/Widget.AppCompat.ProgressBar.Horizontal" />
```

progressDrawable: progress drawable is an attribute used in Android to set the custom drawable for the progress mode.

Example Code:

```
<ProgressBar android:id="@+id/simpleProgressBar"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:max="100"  
    android:progress="60"  
    android:layout_marginTop="100dp"  
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"  
    android:progressDrawable="@drawable/custom_progress" />
```

Fragments



Fragments

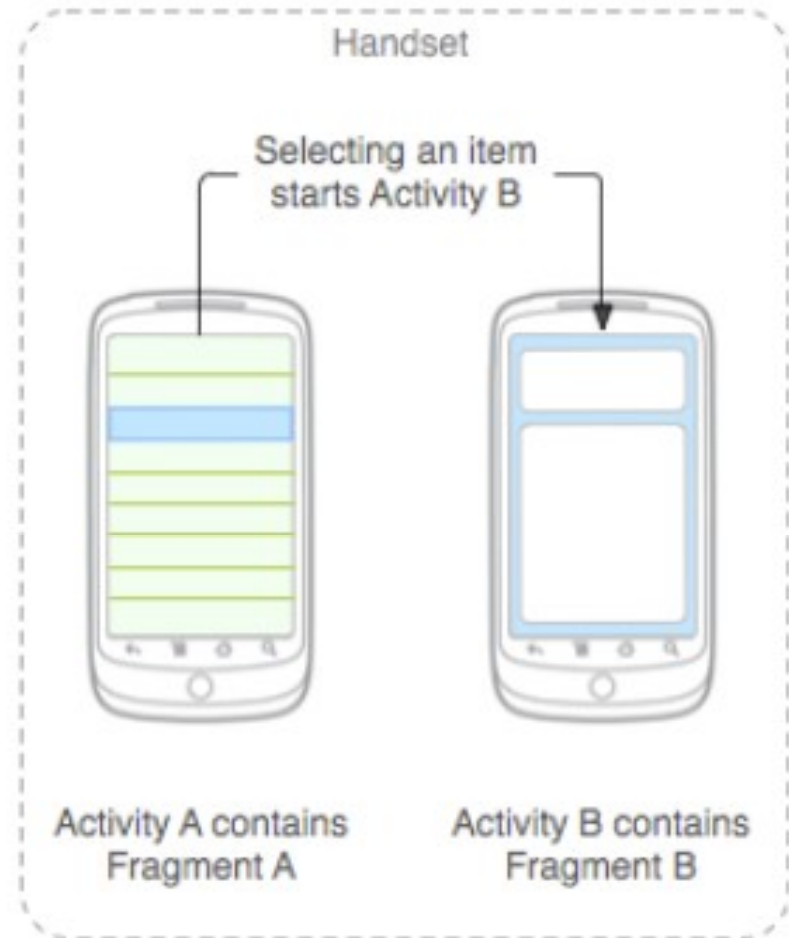
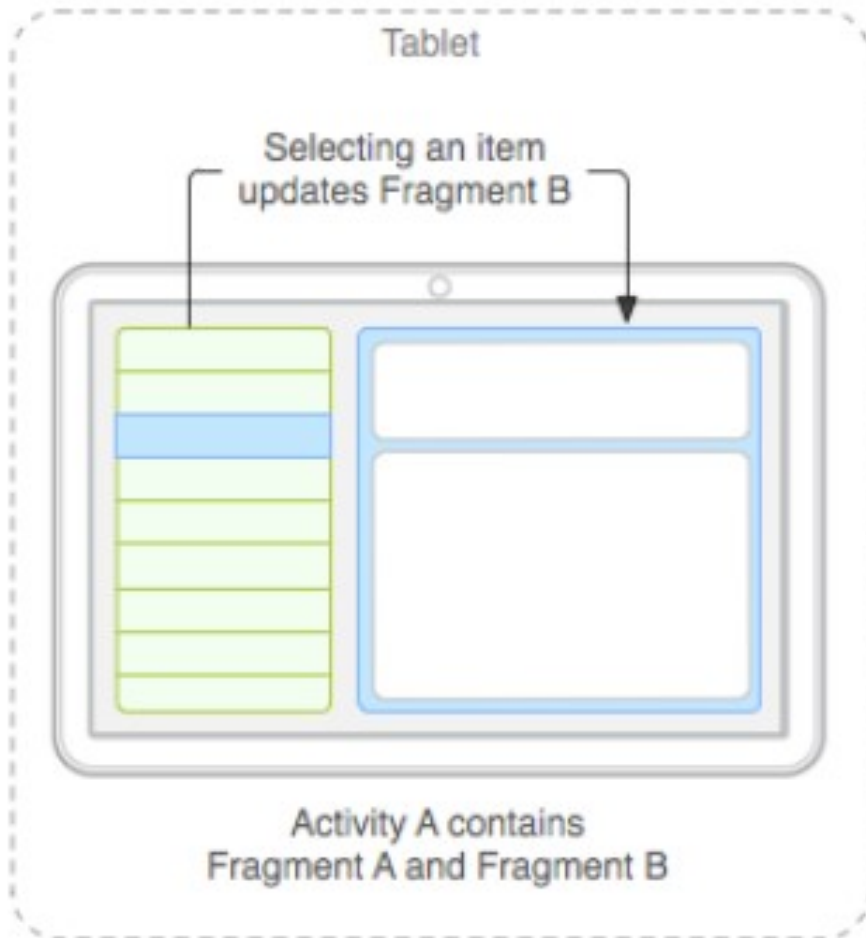
- **Android Fragment** is the part of activity, it is also known as sub-activity.
- It represents a portion of UI that the user sees on the screen.
- A Fragment is a combination of an XML layout file and a java class much like an Activity.
- Fragments are standalone components that can contain views, events and logic.
- There can be more than one fragment in an activity.
- Fragments represent multiple screen inside one activity.
- Android Fragments cannot exist outside an activity.
- **Each fragment has its own life cycle methods** that is affected by activity life cycle because fragments are embedded in activity.
- Fragments encapsulate views and logic so that it is easier to reuse within activities.
- The **FragmentManager** class is responsible to make interaction between fragment objects.
- Another name for Fragment can be **Sub-Activity** as they are part of Activities.
- **Fragments** can be dynamically added and removed as per the requirements.
- Fragments improve the adaptability & user experience by making the UI flexible for all devices.

Uses of Fragments

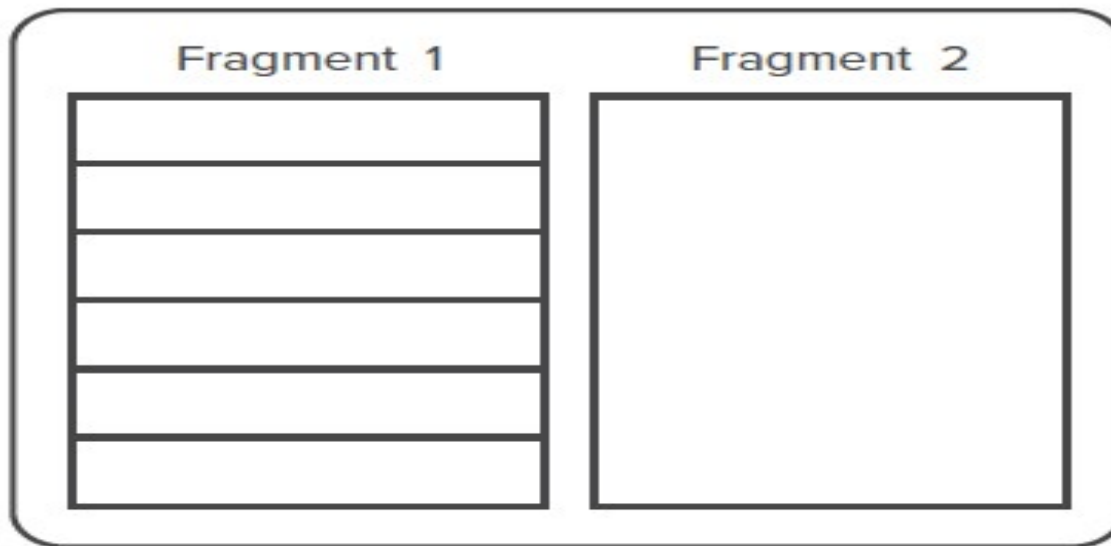
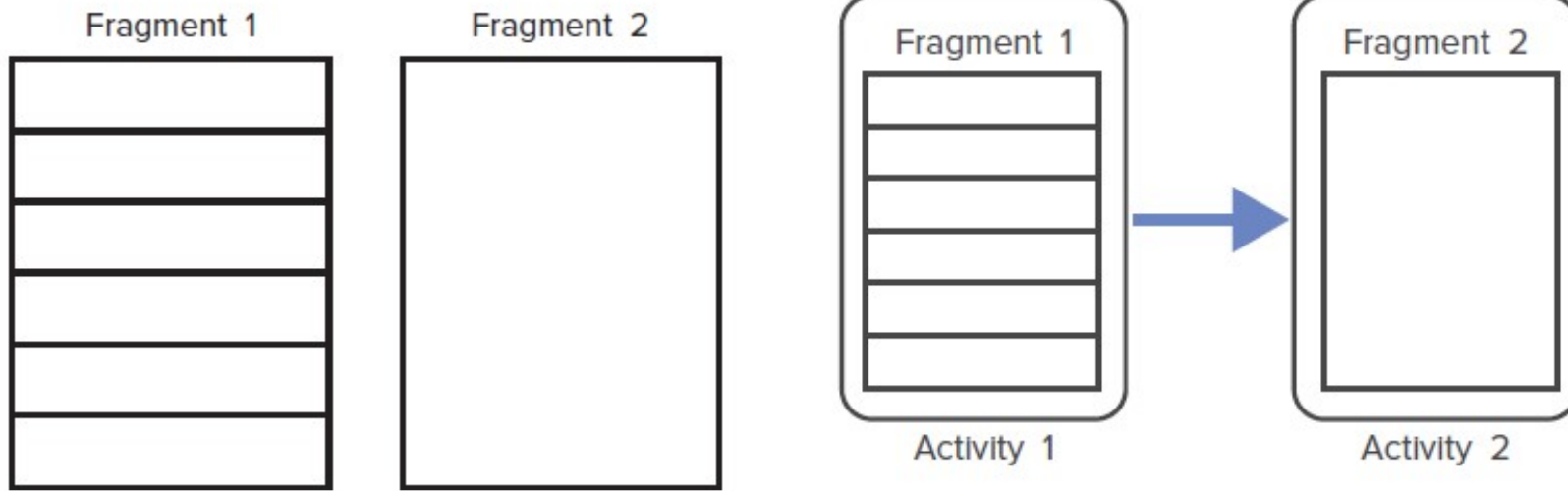
Uses:

- **Reusing View and Logic Components** - Fragments enable re-use of parts of your screen including views and event logic over and over in different ways across many disparate activities. For example, using the same list across different data sources within an app.
- **Tablet Support** - Often within apps, the tablet version of an activity has a substantially different layout from the phone version which is different from the TV version. Fragments enable device-specific activities to reuse shared elements while also having differences.
- **Screen Orientation** - Often within apps, the portrait version of an activity has a substantially different layout from the landscape version. Fragments enable both orientations to reuse shared elements while also having differences.

Fragments

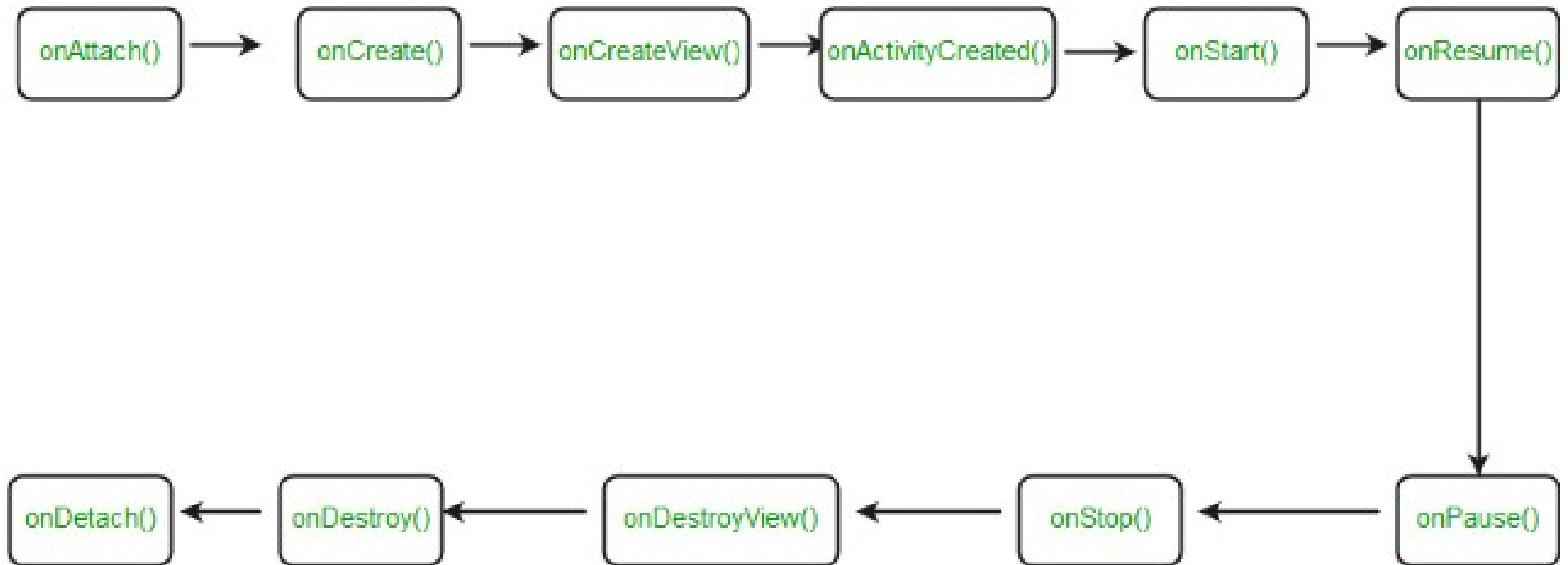


Fragments



Activity 1

Fragment Lifecycle



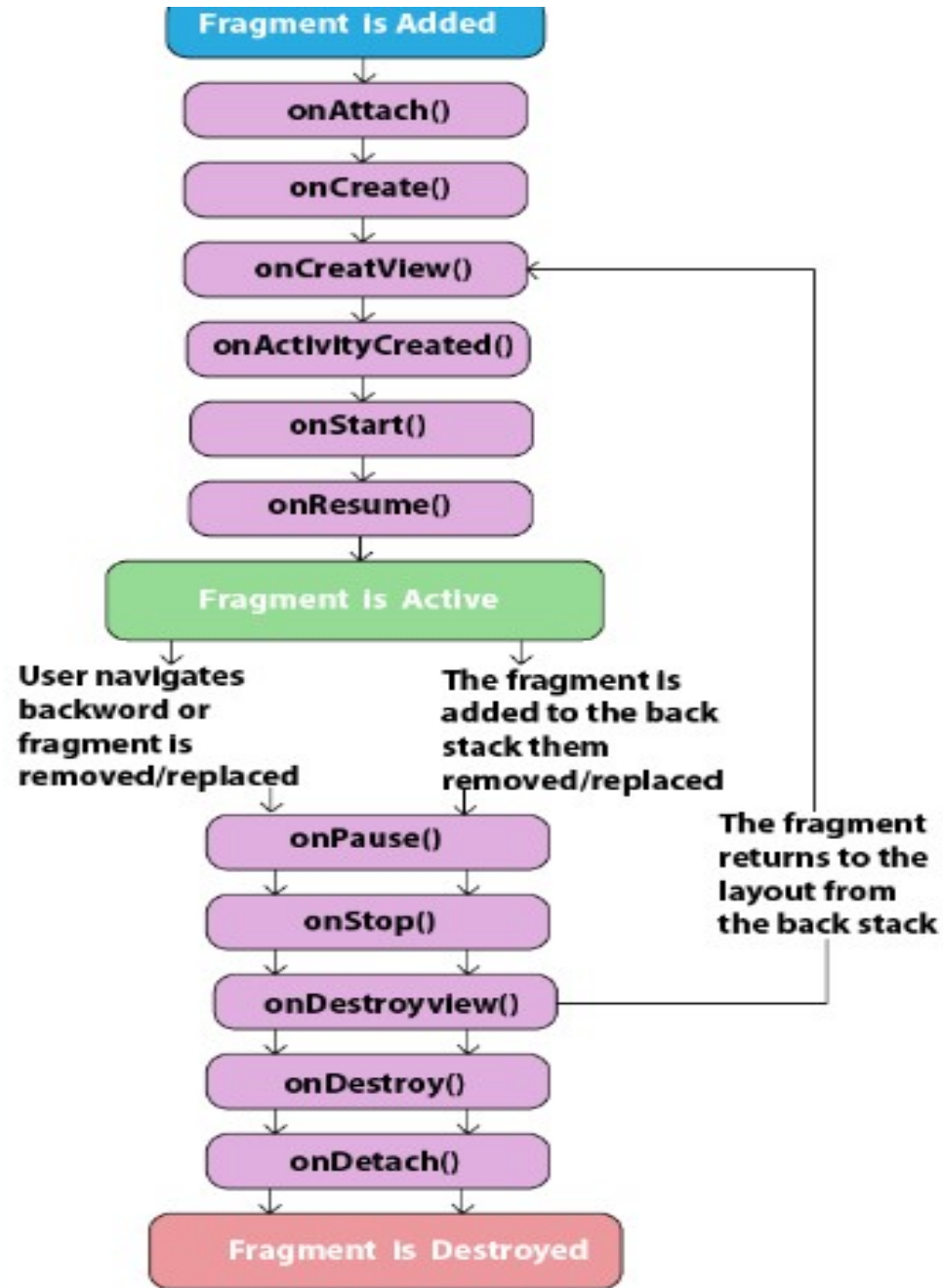
Fragment Lifecycle

- **onAttach()** : The fragment instance is associated with an activity instance. The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
- **onCreate()** : The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView()** : The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **onActivityCreated()** : The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
- **onStart()** : The onStart() method is called once the fragment gets visible.

Fragment Lifecycle

- **onResume()** : Fragment becomes active.
- **onPause()** : The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
- **onStop()** : Fragment going to be stopped by calling onStop()
- **onDestroyView()** : Fragment view will destroy after call this method
- **onDestroy()** : called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

Fragment Lifecycle



Fragment Lifecycle methods

| SL. NO. | METHOD | DESCRIPTION |
|---------|---|--|
| 1) | onAttach(Activity) | it is called only once when it is attached with activity. |
| 2) | onCreate(Bundle) | It is used to initialize the fragment. |
| 3) | onCreateView(LayoutInflater, ViewGroup, Bundle) | creates and returns view hierarchy. |
| 4) | onActivityCreated(Bundle) | It is invoked after the completion of onCreate() method. |
| 5) | onViewStateRestored(Bundle) | It provides information to the fragment that all the saved state of fragment view hierarchy has been restored. |
| 6) | onStart() | makes the fragment visible. |
| 7) | onResume() | makes the fragment interactive. |
| 8) | onPause() | is called when fragment is no longer interactive. |
| 9) | onStop() | is called when fragment is no longer visible. |
| 10) | onDestroyView() | allows the fragment to clean up resources. |
| 11) | onDestroy() | allows the fragment to do final clean up of fragment state. |
| 12) | onDetach() | It is called immediately prior to the fragment no longer being associated with its activity. |

ListFragment

- A list fragment is a fragment that contains a ListView, displaying a list of items from a data source such as an array or a Cursor.
- A list fragment is very useful, as you may often have one fragment that contains a list of items (such as a list of RSS postings), and another fragment that displays details about the selected posting.
- To create a list fragment, you need to extend the ListFragment base class.

Fragment Types

Fragment Types:

1. ListFragment :
1. DialogFragment
2. PreferenceFragment

ListFragment

A list fragment is a fragment that contains a `ListView`, displaying a list of items from a data source such as an array or a `Cursor`.

In the class extended from `Fragment`,

```
public static class ExampleFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup  
container, Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.example_fragment, container, false);  
    }  
}
```

Id → The resource ID of the layout you want to inflate.

Container → parent of the inflated layout.

Boolean value (3rd param) → Indicates whether the inflated layout should be attached to the [ViewGroup](#) (the second parameter) during inflation.

DialogFragment

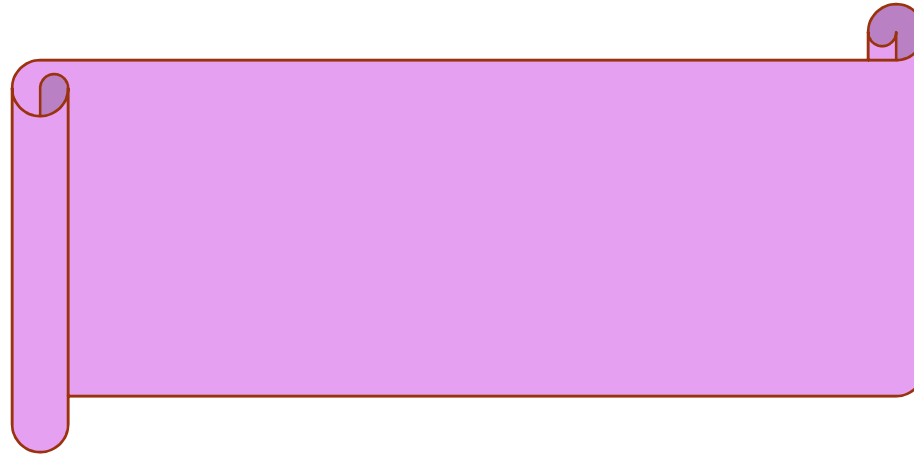
- A dialog fragment floats on top of an activity and is **displayed modally**.
- Dialog fragments are useful for cases in which you need to obtain the user's response before continuing with execution.
- To create a dialog fragment, you need to extend the DialogFragment base class.
- Call **getFragmentManager()** of Activity class to Return the FragmentManager for interacting with fragments associated with this fragment's activity.
- abstract FragmentTransaction **beginTransaction()** Start a series of edit operations on the Fragments associated with this FragmentManager.

PreferenceFragment

- Your Android applications will typically provide preferences that allow users to personalize the application for their own use.
- For example, you may allow users to save the login credentials that they use to access their web resources, or save information such as how often the feeds must be refreshed (such as in an RSS reader application), and so on.
- In Android, you can use the PreferenceActivity base class to display an activity for the user to edit the preferences

PreferenceFragment

- Handling of user preferences (editing, and storing) can be handled by using a hierarchy of preferences via XML.
- Easy and powerful way of handling user preferences.
- PreferenceFragment contains a hierarchy of preference objects as lists. These preferences will automatically save to SharedPreferences as the user interacts with them.
- To retrieve an instance of SharedPreferences that the preference hierarchy in this fragment will use, call `PreferenceManager.getDefaultSharedPreferences(android.content.Context)` with a context in the same package as this fragment.



1. WebView
2. Gallery view
3. ImageSwitcher
4. Fragments – ListFragment,
DialogFragment &
PreferenceFragment
5. Dialogs

WebView

- WebView is a view that display web pages inside your application.
- You can also specify HTML string and can show it inside your application using WebView. WebView makes turns your application to a web application.
- In order to load a web url into the WebView, you need to call a method **loadUrl(String url)** of the WebView class, specifying the required url.

Ex: `WebView browser = (WebView) findViewById(R.id.webview);
browser.loadUrl("http://www.google.com");`

WebView class methods:

getUrl() → return the url of the current page.

getTitle() → This method return the title of the current page.

destroy() → This method destroy the internal state of WebView.

clearHistory() → This method will clear the WebView forward and backward history.

canGoForward() → This method specifies the WebView has a forward history item.

canGoBack() → This method specifies the WebView has a back history item.

getProgress() → This method gets the progress of the current page.

Gallery view

- The Gallery is a view that shows items (such as images) in a center-locked, horizontal scrolling list.

ImageAdapter

```
public class ImageAdapter extends BaseAdapter {  
    public ImageAdapter(Context c) { ... }  
    //---returns the number of images---  
    public int getCount() { ... }  
    //---returns the item---  
    public Object getItem(int position) { ... }  
    //---returns the ID of an item---  
    public long getItemId(int position) { ... }  
    //---returns an ImageView view---  
    public View getView(int position, View convertView, ViewGroup parent) { ... }  
}
```

ImageAdapter

- **getView()** method returns a View at the specified position.
- 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:
- **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.
- **setScaleType(ImageView.ScaleType)** declares that images should be cropped toward the center.

ImageAdapter

- **getView()** method returns a View at the specified position.
- 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:
- **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.
- **setScaleType(ImageView.ScaleType)** declares that images should be cropped toward the center.

ImageSwitcher

- Gallery view together with an ImageView to display a series of thumbnail images so that when one is selected, it is displayed in the ImageView.
- Gallery view together with an ImageSwitcher to display a series of images so that when one is selected, an animation occurs in taking transition from one image to another image.