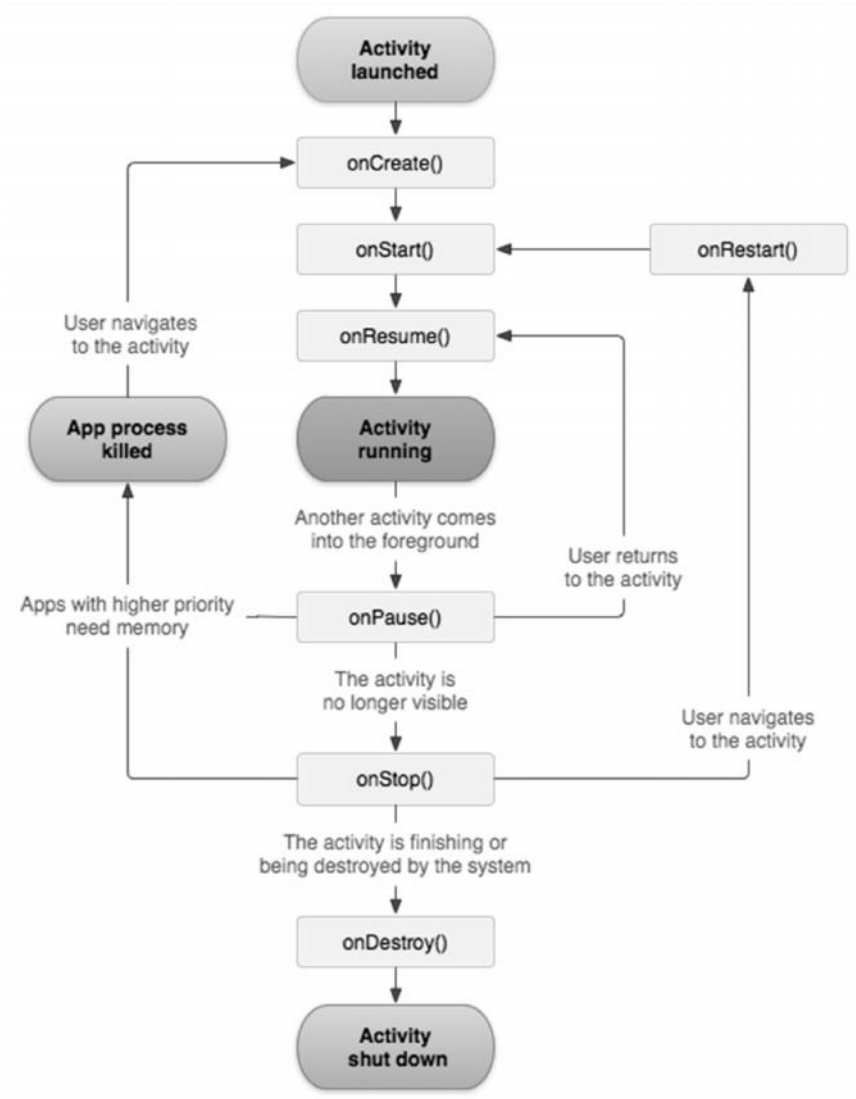
Draw a diagram for activity life cycle and explain it.

- To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, and `onDestroy()`.
- The system invokes each of these callbacks as an activity enters a new state.



Activity Lifecycle

- As the user begins to leave the activity, the system calls methods to dismantle the activity.
- In some cases, this dismantlement is only partial; the activity still resides in memory (such as when the user switches to another app), and can still come back to the foreground.
- If the user returns to that activity, the activity resumes from where the user left off.

- The system's likelihood of killing a given process—along with the activities in it—depends on the state of the activity at the time.
- Activity state and ejection from memory provides more information on the relationship between state and vulnerability to ejection.
- Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods.

Lifecycle Methods

onCreate()

- You must implement this callback, which fires when the system first creates the activity.
- On activity creation, the activity enters the Created state.
- In the `onCreate()` method, you perform basic application startup logic that should happen only once for the entire life of the activity.
- For example, your implementation of `onCreate()` might bind data to lists, initialize background threads, and instantiate some class-scope variables.
- This method receives the parameter savedInstanceState, which is a Bundle object containing the activity's previously saved state.
- If the activity has never existed before, the value of the Bundle object is null.

Your activity does not reside in the Created state. After the `onCreate()` method finishes execution, the activity enters the Started state, and the system calls the `onStart()` and `onResume()` methods in quick succession.

onStart()

- When the activity enters the Started state, the system invokes this callback.
- The `onStart()` call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.
- For example, this method is where the app initializes the code that maintains the UI.
- The `onStart()` method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state.
- Once this callback finishes, the activity enters the Resumed state, and the system invokes the `onResume()` method.

onResume()

- When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the `onResume()` callback.
- This is the state in which the app interacts with the user.
- The app stays in this state until something happens to take focus away from the app.
- Such an event might be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's turning off.
- When an interruptive event occurs, the activity enters the Paused state, and the system invokes the `onPause()` callback.
- If the activity returns to the Resumed state from the Paused state, the system once again calls `onResume()` method.
- For this reason, you should implement `onResume()` to initialize components that you release during `onPause()`.

onPause()

- The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed).
- Use the `onPause()` method to pause operations such animations and music playback that should not continue while the Activity is in the Paused state, and that you expect to resume shortly.
- You can use the `onPause()` method to release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them.
- `onPause()` execution is very brief, and does not necessarily afford enough time to perform save operations.
- For this reason, you should not use `onPause()` to save application or user data, make network calls, or execute database transactions; such work may not complete before the method completes.
- Instead, you should perform heavy-load shutdown operations during `onStop()`.
- If the activity resumes, the system once again invokes the `onResume()` callback.
- If the activity returns from the Paused state to the Resumed state, the system keeps the Activity instance resident in memory, recalling that instance when it the system invokes `onResume()`.

onStop()

- When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the `onStop()` callback.
- This may occur, for example, when a newly launched activity covers the entire screen.
- The system may also call `onStop()` when the activity has finished running, and is about to be terminated.
- In the `onStop()` method, the app should release almost all resources that aren't needed while the user is not using it.
- When your activity enters the Stopped state, the Activity object is kept resident in memory: It maintains all state and member information, but is not attached to the window manager.
- When the activity resumes, the activity recalls this information.
- You don’t need to re-initialize components that were created during any of the callback methods leading up to the Resumed state.
- The system also keeps track of the current state for each View object in the layout, so if the user entered text into an EditText widget, that content is retained so you don't need to save and restore it.

onDestroy()

- Called before the activity is destroyed.
- This is the final call that the activity receives.
- The system either invokes this callback because the activity is finishing due to someone's calling `finish()`, or because the system is temporarily destroying the process containing the activity to save space.
- You can distinguish between these two scenarios with the isFinishing() method.
- The system may also call this method when an orientation change occurs, and then immediately call `onCreate()` to recreate the process (and the components that it contains) in the new orientation.
- The onDestroy() callback releases all resources that have not yet been released by earlier callbacks such as `onStop()`.

What is SQLite database? Write code for adding, updating and removing data from SQLite database.

- SQLite is a opensource SQL database that stores data to a text file on a device.
- Android comes in with built in SQLite database implementation.

- SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC, etc.
- The main package is `android.database.sqlite` that contains the classes to manage your own databases.

Database - Creation

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

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database
name",MODE_PRIVATE,null);
```

Put Information into a Database

Insert data into the database by passing a ContentValues object to the `insert()` method:

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);

// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(TABLE_NAME, null, values);
```

- The first argument for insert() is simply the table name.
- The second argument tells the framework what to do in the event that the ContentValues is empty (i.e., you did not put any values). If you specify the name of a column, the framework inserts a row and sets the value of that column to null. If you specify null, like in this code sample, the framework does not insert a row when there are no values.

Update Rows

```
public void updateRecord(ContactModel contact) {
      database = this.getReadableDatabase();
      ContentValues contentValues = new ContentValues();
      contentValues.put(COLUMN_FIRST_NAME, contact.getFirstName());
      contentValues.put(COLUMN_LAST_NAME, contact.getLastName());
      database.update(TABLE_NAME,  contentValues,  COLUMN_ID  +  "  =  ?",  new
      String[]{contact.getID()});
      database.close();
}

public int update (String table, ContentValues values, String whereClause,
String[] whereArgs)
```

`update()` is the convenient method for updating rows in the database.

Returns:

- The number of rows affected.

Delete Records

```
public void deleteRecord(ContactModel contact) {
    database = this.getReadableDatabase();
    database.delete(TABLE_NAME,     COLUMN_ID     +     "     =     ?",     new
String[]{contact.getID()});
    database.close();
}
```

```
public int delete (String table, String whereClause, String[] whereArgs)
```

`database.delete()` is convenient method for deleting rows in the database

Returns

- The number of rows affected if a whereClause is passed in, 0 otherwise. To remove all rows and get a count pass 1 as the whereClause.

Explain Android Architecture with diagram

- Android is an open source, Linux-based software stack created for a wide array of devices and form factors.
- The major components of the Android platform are:

The Linux Kernel

- The foundation of the Android platform is the Linux kernel.
- For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.
- Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

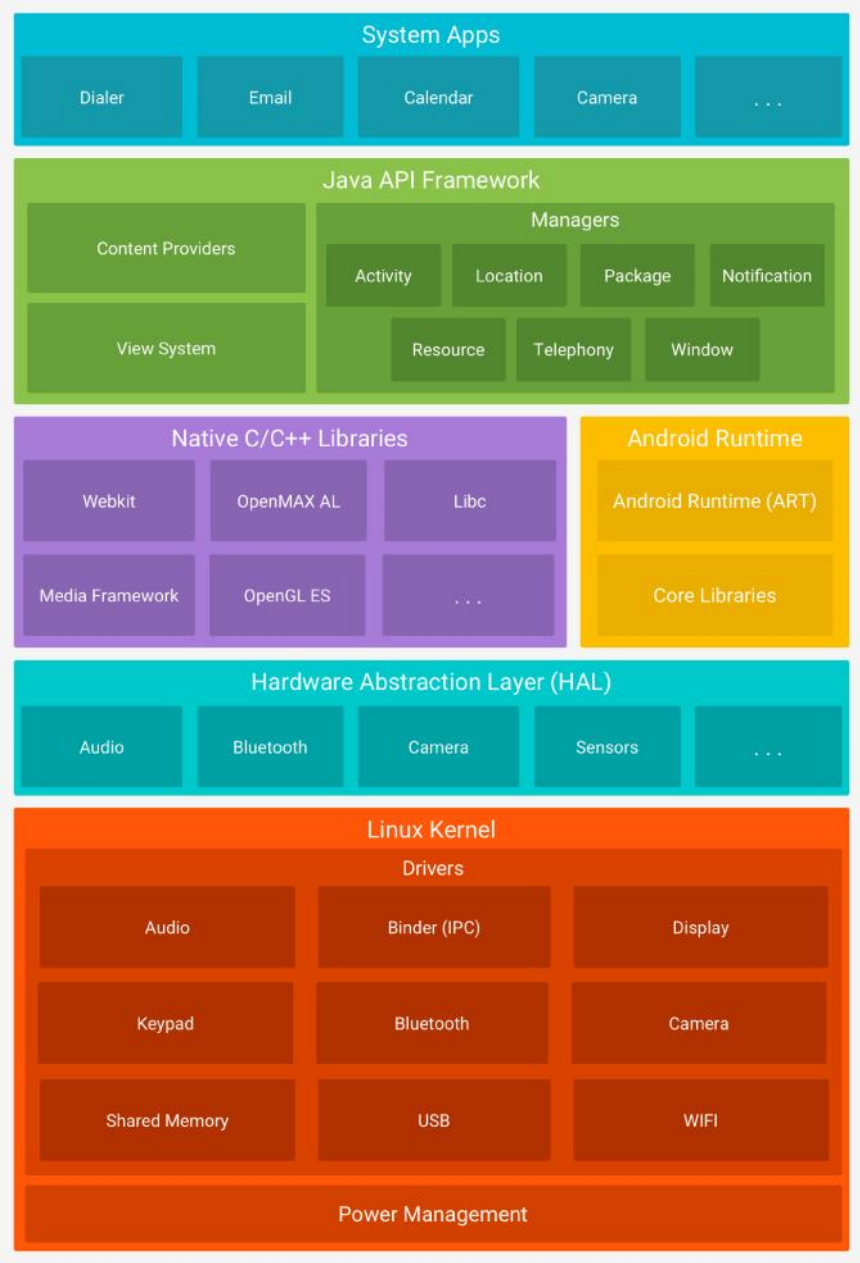
Hardware Abstraction Layer (HAL)

- The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
- The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module.
- When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Android Runtime

- For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART).
- ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed specially for Android that's optimized for minimal memory footprint.
- Build toolchains, such as Jack, compile Java sources into DEX bytecode, which can run on the Android platform.
- Some of the major features of ART include the following:
  - Ahead-of-time (AOT) and just-in-time (JIT) compilation

- Optimized garbage collection (GC)
- Better debugging support



Android Architecture

Native C/C++ Libraries

- Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++.
- The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps.

- For example, you can access OpenGL ES through the Android framework's Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics in your app.
- If you are developing an app that requires C or C++ code, you can use the Android NDK to access some of these native platform libraries directly from your native code.

Java API Framework

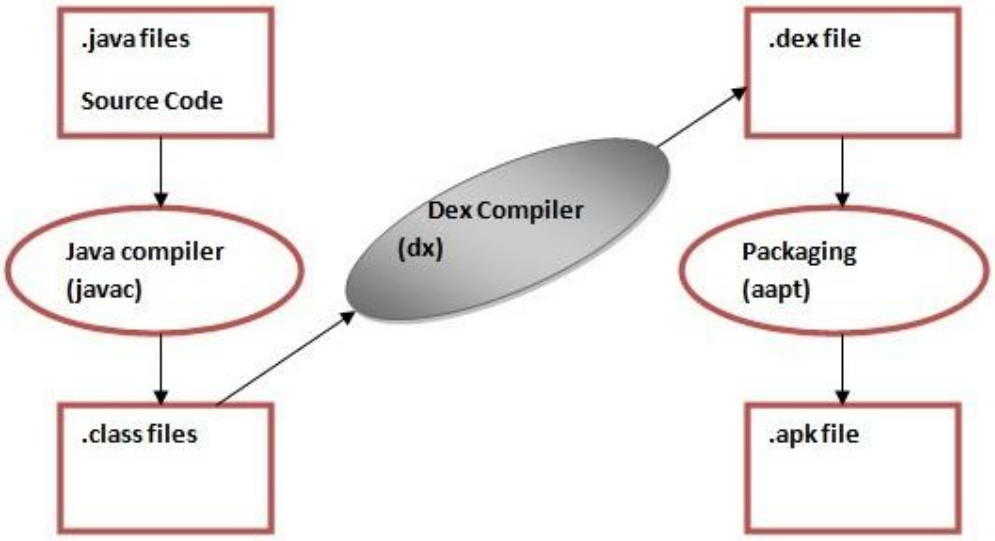
- The entire feature-set of the Android OS is available to you through APIs written in the Java language.
- These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:
  - A rich and extensible View System you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
  - A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
  - A Notification Manager that enables all apps to display custom alerts in the status bar
  - An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack
  - Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

System Apps

- Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more.
- Apps included with the platform have no special status among the apps the user chooses to install.
- So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard.
- The system apps function both as apps for users and to provide key capabilities that developers can access from their own app.
- E.g., if your app would like to deliver an SMS message, you don't need to build that functionality yourself - you can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.

What is the role of DVM in android? Explain it.

- The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices.
- It optimizes the virtual machine for memory, battery life and performance.
- The Dex compiler converts the class files into the .dex file that run on the Dalvik VM.
- Multiple class files are converted into one dex file.



Role of DVM in Android

- The javac tool compiles the java source file into the class file.
- The dx tool takes all the class files of your application and generates a single .dex file.
- It is a platform-specific tool.
- The Android Assets Packaging Tool (aapt) handles the packaging process.

Dalvik virtual machine uses register based architecture. With this architecture, Dalvik Virtual Machine has few advantages over JAVA virtual machine such as:

- Dalvik uses its own 16 bit instruction set than java 8 bit stack instructions, which reduce the dalvik instruction count and raised its interpreter speed.
- Dalvik use less space, which means an uncompressed .dex file is smaller in size(few bytes) than compressed java archive file(.jar file).

List out different types of layouts in android. Explain any two layouts.

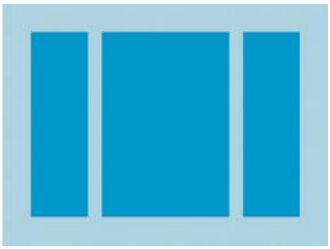
- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:
  - *Declare UI elements in XML.* Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - *Instantiate layout elements at runtime.* Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Common layouts in Android are:

1) Linear Layout
2) Relative Layout
3) Web View
4) Table Layout

5) Frame Layout

Linear Layout



- LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the `android:orientation` attribute.
- All children of a Linear Layout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding).
- A Linear Layout respects margins between children and the gravity (right, center, or left alignment) of each child.

```
<?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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

Relative Layout

- Relative Layout is a view group that displays child views in relative positions.
- The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left or center).

- Relative Layout lets child views specify their position relative to the parent view or to each other (specified by ID).
- So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.
- By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from `RelativeLayout.LayoutParams`.

Some of the many layout properties available to views in a RelativeLayout include:

1) `android:layout_alignParentTop`
If "true", makes the top edge of this view match the top edge of the parent.

2) `android:layout_centerVertical`
If "true", centers this child vertically within its parent.

3) `android:layout_below`
Positions the top edge of this view below the view specified with a resource ID.

4) `android:layout_toRightOf`
Positions the left edge of this view to the right of the view specified with a resource ID.

Sample code illustrating Relative Layout is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
```

```
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

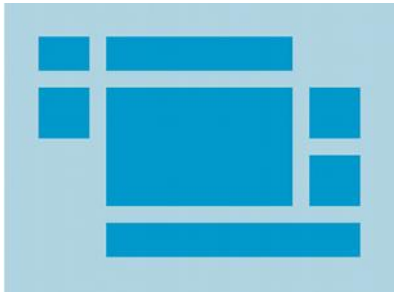
Frame Layout

- Frame Layout is designed to block out an area on the screen to display a single item.
- Generally, Frame Layout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.
- You can, however, add multiple children to a Frame Layout and control their position within the Frame Layout by assigning gravity to each child, using the `android:layout_gravity` attribute.
- Child views are drawn in a stack, with the most recently added child on top.
- The size of the Frame Layout is the size of its largest child (plus padding), visible or not (if the Frame Layout's parent permits). Sample code showing use of FrameLayout has been shown below:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/framelayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_gravity="center"
android:foregroundGravity="fill"
android:foreground="#0f0"><!--foreground color for a FrameLayout-->

<LinearLayout
android:orientation="vertical"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
>

<!-- Imageview will not be shown because of foreground color which is drawn over
it-->

<ImageView
android:layout_width="200dp"
android:layout_height="200dp"
```

```
android:layout_marginBottom="10dp"
android:src="@mipmap/ic_launcher"
android:scaleType="centerCrop"
/>

<!--Textview will not be shown because of foreground color is drawn over it-->

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center_horizontal"
android:text="AndroidSample"/>

</LinearLayout>

</FrameLayout>
```

Table Layout



- A layout that arranges its children into rows and columns.
- A Table Layout consists of a number of `TableRow` objects, each defining a row (actually, you can have other children, which will be explained below).
- Table Layout containers do not display border lines for their rows, columns, or cells.
- Each row has zero or more cells; each cell can hold one View object.
- The table has as many columns as the row with the most cells.
- A table can leave cells empty.
- Cells can span columns, as they can in HTML.
- The width of a column is defined by the row with the widest cell in that column.
- However, a Table Layout can specify certain columns as shrinkable or stretchable by calling `setColumnShrinkable()` or `setColumnStretchable()`.
- If marked as shrinkable, the column width can be shrunk to fit the table into its parent object.
- If marked as stretchable, it can expand in width to fit any extra space.
- The total width of the table is defined by its parent container.
- It is important to remember that a column can be both shrinkable and stretchable.
- In such a situation, the column will change its size to always use up the available space, but never more.

- You can hide a column by calling `setColumnCollapsed()`.
- The children of a Table Layout cannot specify the `layout_width` attribute.
- Width is always `MATCH_PARENT`.
- However, the `layout_height` attribute can be defined by a child; default value is `WRAP_CONTENT`.
- If the child is a `TableRow`, then the height is always `WRAP_CONTENT`.
- Cells must be added to a row in increasing column order, both in code and XML.
- Column numbers are zero-based.
- If you don't specify a column number for a child cell, it will autoincrement to the next available column.
- If you skip a column number, it will be considered an empty cell in that row.
- Although the typical child of a Table Layout is a `TableRow`, you can actually use any View subclass as a direct child of Table Layout.
- The View will be displayed as a single row that spans all the table columns. Sample code showing use of TableLayout has been shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="@string/table_layout_4_open"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_open_shortcut"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:text="@string/table_layout_4_save"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_save_shortcut"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

What is the use of Manifest file in android application? Explain it.

- Every application must have an **AndroidManifest.xml** file in its root directory.
- The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

Among other things, the manifest file does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.

- It describes the components of the application, which include the activities, services, broadcast receivers, and content providers that compose the application.
- It determines the processes that host the application components.
- It declares the permissions that 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 as the application runs. These declarations are present in the manifest only while the application is being developed and are removed before the application is published.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

Manifest file structure

The code snippet below shows the general structure of the manifest file and every element that it can contain. Each element, along with all of its attributes, is fully documented in a separate file.

Here is an example of the manifest file:

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

<manifest>

    <uses-permission />
    <permission />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

    </application>
</manifest>
```

- Only the <manifest> and <application> elements are required.
- They each must be present and can occur only once.
- Most of the other elements can occur many times or not at all.
- The <application> element must be the last element inside the <manifest> element.
- In other words, the </application> closing tag must appear immediately before the </manifest> closing tag. Except for some attributes of the root <manifest> element, all attribute names begin with an android: prefix.

- A permission is a restriction that limits access to a part of the code or to data on the device.
- The limitation is imposed to protect critical data and code that could be misused to distort or damage the user experience.
- If an application needs access to a feature that is protected by a permission, it must declare that it requires the permission with a `<uses-permission>` element in the manifest.

What is Android Runtime? Also explain Android Virtual Device. OR What is virtual device and SDK manager? Explain.

Android runtime (ART)

- Android runtime (ART) is the managed runtime used by applications and some system services on Android.
- ART and its predecessor Dalvik were originally created specifically for the Android project.
- ART as the runtime executes the Dalvik Executable format and Dex bytecode specification.
- ART and Dalvik are compatible runtimes running Dex bytecode, so apps developed for Dalvik should work when running with ART.

ART Features

Here are some of the major features implemented by ART.

- *Ahead-of-time (AOT) compilation* - ART introduces ahead-of-time (AOT) compilation, which can improve app performance. ART also has tighter install-time verification than Dalvik.
- *Improved garbage collection* - ART improves garbage collection in several ways:
  - One GC pause instead of two
  - Parallelized processing during the remaining GC pause
  - Lower total GC time
  - Improved garbage collection ergonomics
  - Compacting GC to reduce background memory usage and fragmentation
- *Development and debugging improvements*
  - Support for sampling profiler
  - Support for more debugging features
  - Improved diagnostic detail in exceptions and crash reports

Android Virtual Device (AVD)

- An Android Virtual Device (AVD) definition lets you define the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.
- The AVD Manager helps you easily create and manage AVDs.
- An AVD contains a hardware profile, system image, storage area, skin, and other properties.

- *Hardware profile*
  The hardware profile defines the characteristics of a device as shipped from the factory. The AVD Manager comes preloaded with certain hardware profiles, such as Nexus phone devices, and you can define and import hardware profiles as needed. You can override some of the settings in your AVD, if needed.
- *System image*
  The AVD Manager helps you choose a system image for your AVD by providing recommendations. It also lets you download system images, some with add-on libraries, like Google APIs.

- *Storage area*
  The AVD has a dedicated storage area on your development machine. It stores the device user data, such as installed apps and settings, as well as an emulated SD card. If needed, you can use the AVD Manager to wipe user data, so the device has the same data as if it were new.
- *Skin*
  An emulator skin specifies the appearance of a device. The AVD Manager provides some predefined skins. You can also define your own, or use skins provided by third parties.

- *AVD and app features*
  Just as with a real device, for apps to use certain features defined in an AVD, such as the camera, it must have the corresponding <uses-feature> setting in the app manifest.

Android SDK Manager

- The Android Development Toolkit (ADT) Bundle provides the Software Development Kit (SDK) and the emulator system image from the latest platform.
- However, you will need other platforms to test your apps on earlier versions of Android.
- You can get components for each platform using the Android SDK Manager.
- It is a set of development tools used to develop applications for Android platform.
- Every time Google releases a new version of Android, a corresponding SDK is also released.
- To be able to write programs with the latest features, developers must download and install each version's SDK for the particular phone.
- The Android SDK includes the following:
  - Required libraries
  - Debugger
  - An emulator
  - Relevant documentation for the Android application program interfaces (APIs)
  - Sample source code
  - Tutorials for the Android OS

Explain the various types of Android Application. OR Explain types of android application. And create any small application in android? Explain.

- Mobile apps are basically little, self-contained programs, used to enhance existing functionality, hopefully in a simple, more user-friendly way.
- They all come with powerful web browsers, meaning you can do pretty much anything you can do on a desktop computer in a phone's browser. Normally, when people talk about apps they are almost always referring to programs that run on mobile devices, such as Smartphones or Tablet Computers.
- Each app provides limited and isolated functionality such as game, calculator or mobile web browsing. Although apps may have avoided multitasking because of the limited hardware resources of the early mobile devices, their specificity is now part of their desirability because they allow consumers to hand-pick what their device are able to do.
- Different types of Apps:
  1) Native App
  2) Web App
  3) Hybrid App

Native App

- Native App has been developed for use on a particular platform or device.
- A native mobile app is a Smartphone application that is coded in a specific programming language, Java for Android operating systems.
- Native mobile apps provide fast performance and a high degree of reliability.
- They also have access to a phone's various devices, such as its camera and address book.
- In addition, users can use some apps without an Internet connection.
- However, this type of app is expensive to develop because it is tied to one type of operating system, forcing the company that creates the app to make duplicate versions that work on other platforms.
- A Native app can only be "Native" to one type of mobile operating system.
- Most video games are native mobile apps.

Web App

- Web App stored on a remote server and delivered over the internet through a browser.
- Web apps are not real apps; they are really websites that, in many ways, look and feel like native applications.
- They are run by a browser and typically written in HTML5.
- Users first access them as they would access any web page: they navigate to a special URL and then have the option of "installing" them on their home screen by creating a bookmark to that page.
- In contrast, a mobile web app is software that uses technologies such as JavaScript or HTML5 to provide interaction, navigation, or customization capabilities.
- These programs run within a mobile device's web browser.
- This means that they're delivered wholly on the fly, as needed, via the internet; they are not separate programs that get stored on the user's mobile device.
- Web apps became really popular when HTML5 came around and people realized that they can obtain native-like–functionality in the browser.

Hybrid App

- This type of application has cross-platform compatibility but can still access a phone's hardware.
- It is developed using platforms such as *Sencha*, *PhoneGap* and *Mosync*.
- Hybrid Apps are like native apps, run on the device, and are written with web technologies (HTML5, CSS and JavaScript).
- Hybrid apps run inside a native container, and leverage the device's browser engine to render the HTML and process the JavaScript locally.
- A web-to-native abstraction layer enables access to device capabilities that are not accessible in Mobile Web applications, such as the accelerometer, camera and local storage.
- Hybrid apps are also popular because they allow cross-platform development: that is, the same HTML code components can be reused on different mobile operating systems, reducing significantly the development costs.

Creating an Android Project

The steps below shows you how to create a new Android project with Android Studio:

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

2) In the **New Project** screen, enter the following values:
Application Name: "My First App"
Company Domain: "example.com"

3) Click **Next**.

4) In the **Target Android Devices** screen, keep the default values and click Next.
The Minimum Required SDK is the earliest version of Android that your app supports, which is indicated by the **API level**. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set.

5) In the **Add an Activity to Mobile** screen, select **Empty Activity** and click **Next**.

6) In the **Customize the Activity** screen, keep the default values and click **Finish**.

After some processing, Android Studio opens and displays a "Hello World" app with default files. Following files will be created after successful processing

- `app > java > com.example.myfirstapp > MainActivity.java`
- `app > res > layout > activity_main.xml`
- `app > manifests > AndroidManifest.xml`
- `Gradle Scripts > build.gradle`

What is parsing? Explain JSON parsing with example.

- Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form.
- XML is a popular format for sharing data on the internet.
- Websites that frequently update their content, such as news sites or blogs, often provide an XML feed so that external programs can keep abreast of content changes.
- Uploading and parsing XML data is a common task for network-connected apps.
- We will see how to parse XML documents and use their data.

- XmlPullParser, which is an efficient and maintainable way to parse XML on Android. Historically Android has had two implementations of this interface:

KXmlParser `via XmlPullParserFactory.newPullParser()`
ExpatPullParser `via Xml.newPullParser()`

JSON Parsing

- JSON stands for JavaScript Object Notation.
- It is an independent data exchange format and is the best alternative for XML.
- Android provides four different classes to manipulate JSON data.
- These classes are `JSONArray`, `JSONObject`, `JSONStringer` and `JSONTokenizer`.

- The first step is to identify the fields in the JSON data in which you are interested in. For example. In the JSON given below we interested in getting temperature only.

```
{
   "sys":
   {
      "country":"GB",
      "sunrise":1381107633,
      "sunset":1381149604
   },
   "weather":[
      {
         "id":711,
         "main":"Smoke",
         "description":"smoke",
         "icon":"50n"
      }
   ],

  "main":
   {
      "temp":304.15,
      "pressure":1009,
   }
}
```

JSON - Elements

An JSON file consist of many components. Here is the table defining the components of an JSON file and their description −

- Array([) - In a JSON file , square bracket ([) represents a JSON array
- Objects({) - In a JSON file, curly bracket ({) represents a JSON object
- Key - A JSON object contains a key that is just a string. Pairs of key/value make up a JSON object
- Value - Each key has a value that could be string, integer or double etc.,

JSON - Parsing

For parsing a JSON object, we will create an object of class JSONObject and specify a string containing JSON data to it. Its syntax is −

String in:

```
JSONObject reader = new JSONObject(in);
```

The last step is to parse the JSON. A JSON file consist of different object with different key/value pair e.t.c. So JSONObject has a separate function for parsing each of the component of JSON file. Its syntax is given below −

```
JSONObject sys  = reader.getJSONObject("sys");
country = sys.getString("country");

JSONObject main  = reader.getJSONObject("main");
temperature = main.getString("temp");
```

The method `getJSONObject` returns the JSON object. The method `getString` returns the string value of the specified key.

Sample JSON Parsing

```
@Override
      protected Void doInBackground(Void... arg0) {
         HttpHandler sh = new HttpHandler();
         // Making a request to url and getting response
         String url = "http://api.twitteruser.info/contacts/";
         String jsonStr = sh.makeServiceCall(url);

         Log.e(TAG, "Response from url: " + jsonStr);
         if (jsonStr != null) {
            try {
               JSONObject jsonObj = new JSONObject(jsonStr);

               // Getting JSON Array node
               JSONArray contacts = jsonObj.getJSONArray("contacts");

               // looping through All Contacts
               for (int i = 0; i < contacts.length(); i++) {
                  JSONObject c = contacts.getJSONObject(i);
                  String id = c.getString("id");
                  String name = c.getString("name");
                  String email = c.getString("email");
                  String address = c.getString("address");
                  String gender = c.getString("gender");

                  // Phone node is JSON Object
                  JSONObject phone = c.getJSONObject("phone");
                  String mobile = phone.getString("mobile");
                  String home = phone.getString("home");
                  String office = phone.getString("office");

                  // tmp hash map for single contact
                  HashMap<String, String> contact = new HashMap<>();

                  // adding each child node to HashMap key => value
                  contact.put("id", id);
                  contact.put("name", name);
                  contact.put("email", email);
                  contact.put("mobile", mobile);

                  // adding contact to contact list
                  contactList.add(contact);
               }
            } catch (final JSONException e) {
               Log.e(TAG, "Json parsing error: " + e.getMessage());
               runOnUiThread(new Runnable() {
                  @Override
                  public void run() {
                     Toast.makeText(getApplicationContext(),
                     "Json parsing error: " + e.getMessage(),
                        Toast.LENGTH_LONG).show();
                  }
               });

            }
```

```
        } else {
           Log.e(TAG, "Couldn't get json from server.");
           runOnUiThread(new Runnable() {
              @Override
              public void run() {
                 Toast.makeText(getApplicationContext(),
                    "Couldn't get json from server.", Toast.LENGTH_LONG).show();
              }
           });
        }

        return null;
     }
```

What is Parsing? Explain XML Parsing with suitable example.

- Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form.
- XML is a popular format for sharing data on the internet.
- Websites that frequently update their content, such as news sites or blogs, often provide an XML feed so that external programs can keep abreast of content changes.
- Uploading and parsing XML data is a common task for network-connected apps.
- We will see how to parse XML documents and use their data.

XmlPullParser, which is an efficient and maintainable way to parse XML on Android. Historically Android has had two implementations of this interface:

```
KXmlParser via XmlPullParserFactory.newPullParser()
ExpatPullParser, via Xml.newPullParser()
```

XML Parsing

- XML stands for Extensible Mark-up Language.
- XML is a very popular format and commonly used for sharing data on the internet.
- This chapter explains how to parse the XML file and extract necessary information from it.
- Android provides three types of XML parsers which are DOM,SAX and XMLPullParser.
- Among all of them android recommend XMLPullParser because it is efficient and easy to use.
- So we are going to use XMLPullParser for parsing XML.
- The first step is to identify the fields in the XML data in which you are interested in.
- E.g., in the XML given below we interested in getting temperature only.

```
<?xml version="1.0"?>
<current>

   <city id="2643743" name="London">
      <coord lon="-0.12574" lat="51.50853"/>
      <country>GB</country>
      <sun rise="2013-10-08T06:13:56" set="2013-10-08T17:21:45"/>
   </city>

   <temperature value="289.54" min="289.15" max="290.15" unit="kelvin"/>
```

```
   <humidity value="77" unit="%"/>
   <pressure value="1025" unit="hPa"/>
</current>
```

XML - Elements

An xml file consist of many components. Here is the table defining the components of an XML file and their description.

- **Prolog** - An XML file starts with a prolog. The first line that contains the information about a file is prolog
- **Events** - An XML file has many events. Event could be like this. Document starts, Document ends, Tag start, Tag end and Text
- **Text** - Apart from tags and events, and xml file also contains simple text. Such as GB is a text in the country tag.
- **Attributes** - Attributes are the additional properties of a tag such as value.

- We will create `XMLPullParser` object , but in order to create that we will first create `XmlPullParserFactory` object and then call its `newPullParser()` method to create `XMLPullParser`. Its syntax is given below –

```
private XmlPullParserFactory xmlFactoryObject = XmlPullParserFactory.newInstance();
private XmlPullParser myparser = xmlFactoryObject.newPullParser();
```

The next step involves specifying the file for `XmlPullParser` that contains XML. It could be a file or could be a Stream. In our case it is a stream.
Its syntax is given below:

```
myparser.setInput(stream, null);
```

The last step is to parse the XML. An XML file consist of events, Name, Text, AttributesValue etc., So XMLPullParser has a separate function for parsing each of the component of XML file. Its syntax is given below –

```
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)  {
   String name=myParser.getName();
   switch (event){
      case XmlPullParser.START_TAG:
      break;

      case XmlPullParser.END_TAG:
      if(name.equals("temperature")){
         temperature = myParser.getAttributeValue(null,"value");
      }
      break;
   }
   event = myParser.next();
}
```

The method `getEventType` returns the type of event that happens. e.g: Document start, tag start. The method `getName` returns the name of the tag and since we are only interested in temperature, so we just check in conditional statement that if we got a temperature tag, we call the method `getAttributeValue` to return us the value of temperature tag.

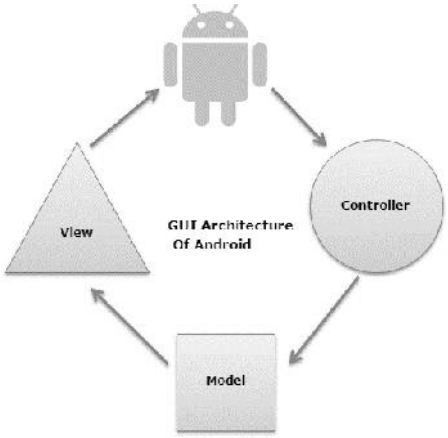
Sample XML Parsing

```
import java.io.InputStream;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
   TextView tv1;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      tv1=(TextView)findViewById(R.id.textView1);

      try {
         InputStream is = getAssets().open("file.xml");

         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
         Document doc = dBuilder.parse(is);

         Element element=doc.getDocumentElement();
         element.normalize();

         NodeList nList = doc.getElementsByTagName("employee");

         for (int i=0; i<nList.getLength(); i++) {

            Node node = nList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
               Element element2 = (Element) node;
               tv1.setText(tv1.getText()+"\nName : " + getValue("name",
element2)+"\n");
               tv1.setText(tv1.getText()+"Surname : " + getValue("surname",
element2)+"\n");
               tv1.setText(tv1.getText()+"-----------------------");
            }
         }

      } catch (Exception e) {e.printStackTrace();}

   }

   private static String getValue(String tag, Element element) {
      NodeList nodeList = element.getElementsByTagName(tag).item(0).getChildNodes();
      Node node = nodeList.item(0);
      return node.getNodeValue();
   }
```

```
}
```

Explain GUI Architecture of android.

- Android GUI is single-threaded, event-driven and built on a library of nestable components.
- The Android UI framework is organized around the common Model-View-Controller pattern.
- It provides structure and tools for building a Controller that handles user input (like key presses and screen taps) and a View that renders graphical information to the screen.



The Model: The model represents data or data container. We can see it as a database of pictures on our device. Let's say, any user wants to hear an audio file, he clicks play button and it triggers an event in our app, now the app will get data from data store or database and as per input and creates data to be sent back to the user. We can refer this data as Model.

The View: The View is the portion of the application responsible for rendering the display, sending audio to speakers, generating tactile feedback, and so on.

Now as per above example, the view in a hypothetical audio player might contain a component that shows the album cover for the currently playing tune. User will always interact with this layer. User action on this layer will trigger events that will go to the application functions.

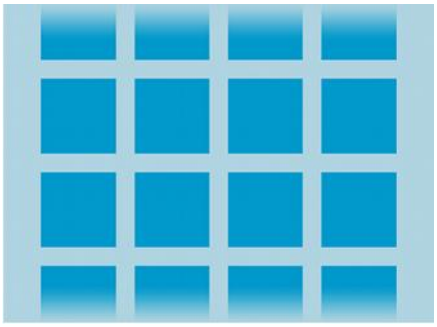
The Controller: The Controller is the portion of an application that responds to external actions: a keystroke, a screen tap, an incoming call, etc. It is implemented as an event queue. On User's action, the control is passed over to controller and this will take care of all logic that needs to be done and prepare Model that need to be sent to view layer.

What is view? Create a simple Grid View application.

- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.
- View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.).
- The `ViewGroup` subclass is the base class for layouts, which are invisible containers that hold other Views and define their layout properties.
- All of the views in a window are arranged in a single tree.
- You can add views either from code or by specifying a tree of views in one or more XML layout files.
- There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

Grid View

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



- Following steps will create a grid of image thumbnails.
- When an item is selected, a toast message will display the position of the image.

Steps to create GridView are as follows:

- Start a new project named HelloGridView.
- Find some photos you'd like to use. Save the image files into the project's res/drawable/ directory.
- Open the res/layout/main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

- This GridView will fill the entire screen.

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

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

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

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

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

To do something when an item in the grid is clicked, the setOnItemClickListener() method is passed a new AdapterView.OnItemClickListener. This anonymous instance defines the onItemClick() callback method to show a Toast that displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

Create a new class called ImageAdapter that extends BaseAdapter:

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

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

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

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

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

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

```
            imageView = (ImageView) convertView;
        }

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

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

- First, this implements some required methods inherited from BaseAdapter.
- `getItem(int)` should return the actual object at the specified position in the adapter.
- `getItemId(int)` should return the row id of the item, but it's not needed here.

- The first method necessary is `getView()`. This method creates a new View for each image added to the ImageAdapter.
- When this is called, a View is passed in, which is normally a recycled object, 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 (if necessary).
  - `setPadding(int, int, int, int)` defines the padding for all sides.

- If the View passed to `getView()` is not null, then the local ImageView is initialized with the recycled View object.
- At the end of the `getView()` method, the position integer passed into the method is used to select an image from the mThumbIds array, which is set as the image resource for the ImageView.
- All that's left is to define the mThumbIds array of drawable resources.
- Run the application.

What is fragment and how we can create it in android application?

- A Fragment represents a behavior or a portion of user interface in an Activity.

- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running.
- A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.
- However, while an activity is running (it is in the resumed lifecycle state), you can manipulate each fragment independently, such as add or remove them.
- When you perform such a fragment transaction, you can also add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred.
- The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the Back button.

Adding a fragment to an activity

- Usually, a fragment contributes a portion of UI to the host activity, which is embedded as a part of the activity's overall view hierarchy.
- There are two ways you can add a fragment to the activity layout:

Declare the fragment inside the activity's layout file.

- In this case, you can specify layout properties for the fragment as if it were a view. For example, here's the layout file for an activity with two fragments:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

- The android:name attribute in the <fragment> specifies the Fragment class to instantiate in the layout.
- When the system creates this activity layout, it instantiates each fragment specified in the layout and calls the onCreateView() method for each one, to retrieve each fragment's layout.
- The system inserts the View returned by the fragment directly in place of the <fragment> element.

Programmatically add the fragment to an existing ViewGroup

- At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a ViewGroup in which to place the fragment.

- To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction.
- You can get an instance of FragmentTransaction from your Activity like this:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

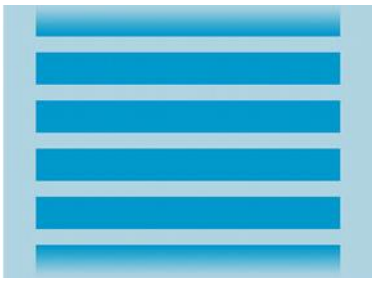
- You can then add a fragment using the add() method, specifying the fragment to add and the view in which to insert it. For example:

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

- The first argument passed to add() is the ViewGroup in which the fragment should be placed, specified by resource ID, and the second parameter is the fragment to add.
- Once you've made your changes with FragmentTransaction, you must call commit() for the changes to take effect.

What is Listview. Write a java class to add any 10 items within Listview.

- ListView is a view group that displays a list of scrollable items.
- The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database query and converts each item result into a view that's placed into the list.



Example to list Planet names in List View:

Basically we create a ListView class and use TextView objects for each row. Each planet name is rendered in a `TextView`.

Main Layout File

Our ListView is defined in the main layout file (res/layout/main.xml) within a `LinearLayout`.

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

    <ListView android:layout_width="fill_parent"
```

```
      android:layout_height="fill_parent"
      android:id="@+id/mainListView">
    </ListView>
</LinearLayout>
```

The resource ID of the `ListView` is `mainListView`, which we will use to get a reference to the `ListView` in our Activity class.

// Find the ListView resource.

```
mainListView = (ListView) findViewById( R.id.mainListView );
```

Row Layout File

Each row in the ListView will be a TextView. The TextView is defined in another file (res/layout/simplerow.xml).

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/rowTextView"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:padding="10dp"
 android:textSize="16sp" >
</TextView>
```

Activity

- Our main activity (`SimpleListViewActivity`) creates an `ArrayAdapter`, which holds the objects to be displayed in the `ListView`.
- The `ArrayAdapter` constructor is passed the resource ID of the `TextView` layout file (`R.layout.simplerow`).
- The `ArrayAdapter` will use it to instantiate a `TextView` for each row.

We then set the ArrayAdapter as our ListView's adapter.

```
package com.example.android;

import java.util.ArrayList;
import java.util.Arrays;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class SimpleListViewActivity extends Activity {

  private ListView mainListView ;
  private ArrayAdapter<String> listAdapter ;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

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

    // Find the ListView resource.
    mainListView = (ListView) findViewById( R.id.mainListView );

    // Create and populate a List of planet names.
    String[] planets = new String[] { "Mercury", "Venus", "Earth", "Mars",
                                      "Jupiter", "Saturn", "Uranus", "Neptune"};
    ArrayList<String> planetList = new ArrayList<String>();
    planetList.addAll( Arrays.asList(planets) );

    // Create ArrayAdapter using the planet list.
    listAdapter    =    new    ArrayAdapter<String>(this,    R.layout.simplerow,
planetList);

    // Add more planets. If you passed a String[] instead of a List<String>
    // into the ArrayAdapter constructor, you must not add more items.
    // Otherwise an exception will occur.
    listAdapter.add( "Ceres" );
    listAdapter.add( "Pluto" );
    listAdapter.add( "Haumea" );
    listAdapter.add( "Makemake" );
    listAdapter.add( "Eris" );

    // Set the ArrayAdapter as the ListView's adapter.
    mainListView.setAdapter( listAdapter );
  }
}
```

What is web services? How it integrates and implements in industrial projects?

- A web service is a standard used for exchanging information between applications or systems of heterogeneous type.
- Software applications written in various programming languages and running on various platforms can use web services to exchange information over Internet using http protocol.

Why Web Service?

Expose method as a service over network:

- Web service is a chunk of code written in some programming language (Say C# or Java) that can be invoked remotely through http protocol.
- Once the Web methods are exposed publically, any applications can invoke it and use the functionality of the web methods.

Application Inter-operability – Connect heterogeneous applications:

- With the help of web service, heterogeneous applications (Java and PHP application) can communicate with each other.
- Web service written in Java can be invoked from PHP by passing the required parameters to web methods.
- This makes the applications platform and technology independent.

Standardized protocol:

- Web Services uses standardized industry standard protocol for the communication which must be adhered and followed by the applications creating Web Service.

Cost effective communication:

- Web service is using SOAP over HTTP protocol for communication which is much cheaper than systems like B2B.

RESTFul Webservice Request and Response – Drill down

- Fist step in designing RESTFul webservice is choosing the right domain name – say weatherinfo.org to retrieve weather information of cities.
- Let us take the first example (weatherinfo.org) which takes city name as input, composes weather information of the city and respond back to the browser.
- Assume the response is in XML format, this can be in other formats like JSON as well.
- Here is the structure of Request and Response:

HTTP Request

The client request from the browser will look like:

```
GET http://weatherinfo.org/getweather/mumbai HTTP/1.1
```

HTTP Response

The server response will look like

```
HTTP/1.1 200 Ok
Date: Mon, 14 Apr 2014 10:20:58 GMT
Content-Type: text/xml
Content-length: 139

<City name="Mumbai" datetime="2014-04-14 10:20:58 GMT" >
<Condition>Scattered Clouds</Condition>
<Temp>33</Temp>
</City>
```

- Line 1 is the initial line which has the HTTP response code – 200 OK, lines 2 through 4 are the HTTP headers, line 5 is the mandatory blank line separating header and body, and lines 6 through 10 constitute the "HTTP Body (or content)" – this part is the data that the response carries and can be in any format, not necessarily XML.

Advantages of using RESTful webservice

- RESTful Web services are designed with less dependence on proprietary middleware (for example, an application server) than the SOAP- and WSDL-based kind.
- As per the RESTful interface design, XML or JSON over HTTP is a powerful interface that allows internal applications, such as Asynchronous JavaScript + XML/JSON (Ajax) - based custom user interfaces, to easily connect, address, and consume resources.
- The great fit between Ajax and REST has increased the amount of attention REST is getting these days.
- Exposing a system's resources through a RESTful API is a flexible way to provide different kinds of applications with data formatted in a standard way. It helps to meet integration requirements that are critical to building systems where data can be easily combined (mashups) and to extend or build on a set of base, RESTful services into something much bigger.