Mobile Application Development (MAD)
GTU # 3170726

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Android Operating System

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
9428231065

Introduction to Android

- Android is an Operating System and programming platform developed by Google.
- It supports devices like mobile phones and other devices, such as tablets, watch, TV, & auto.
- Android includes a Software Development Kit (SDK) that helps you write code and assemble software modules to create apps for Android users.
- Android also provides a marketplace (Play Store) to distribute apps.
- It conquered around **75%** of the global market share by the end of 2020.
- The company named **Open Handset Alliance** developed Android for the first time that is based on the modified version of the Linux kernel and other open-source software.
- **Google** sponsored the project at initial stages and in the year 2005, it acquired the whole company.
- In September 2008, the first Android-powered device launched in the market.

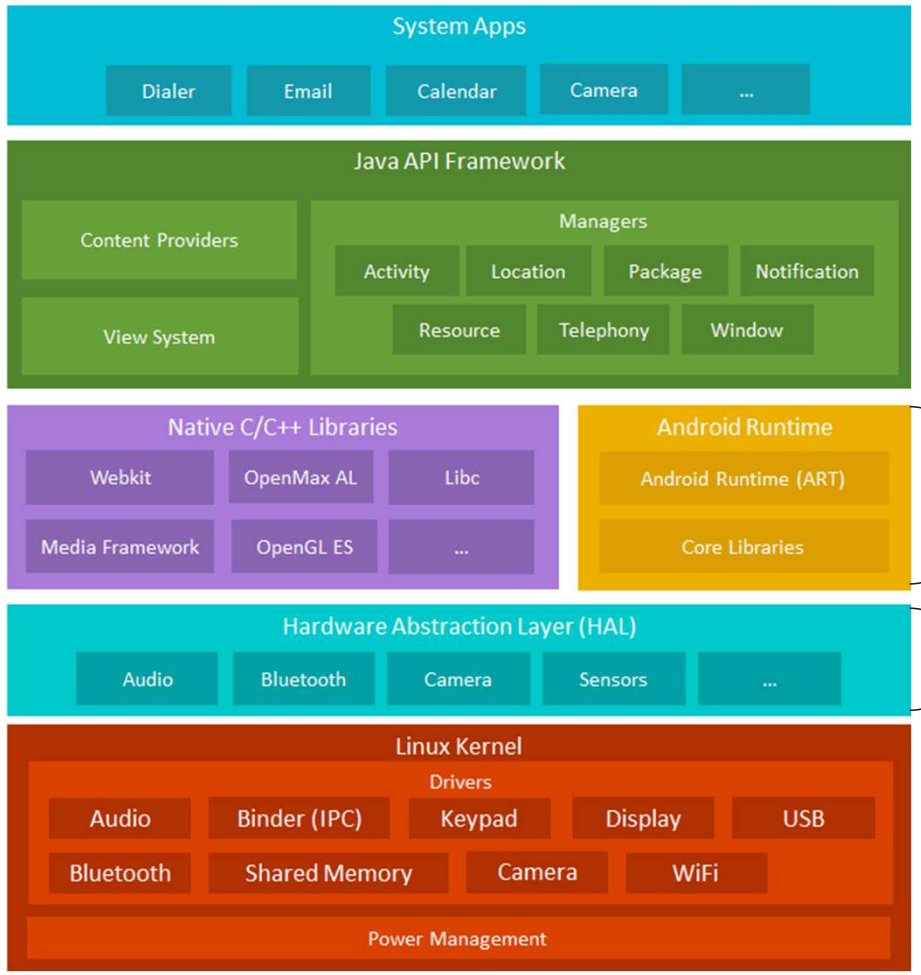
Android System With Architecture

Android comes with a set of core applications such as Contacts, Calendar, Maps, and a browser

You use these APIs to control what your app looks like and how it behaves

You can access OpenGL ES through the native C/C++ libraries to add support for drawing and manipulating 2D and 3D graphics in your app

Underneath everything else lies the Linux kernel. Android relies on the kernel for drivers, and also core services such as security and memory management.

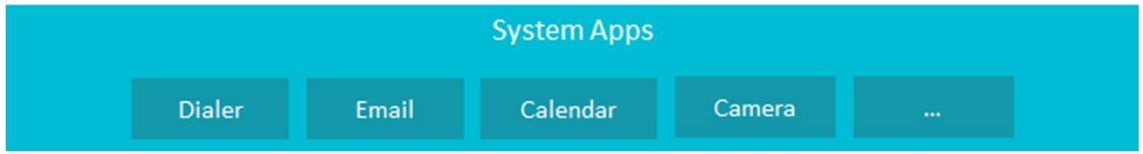


The Android runtime comes with a set of core libraries that implement most of the Java programming language. Each Android app runs in its own process.

HAL provides an interfaces that expose device hardware capabilities to the higher-level Java API framework.

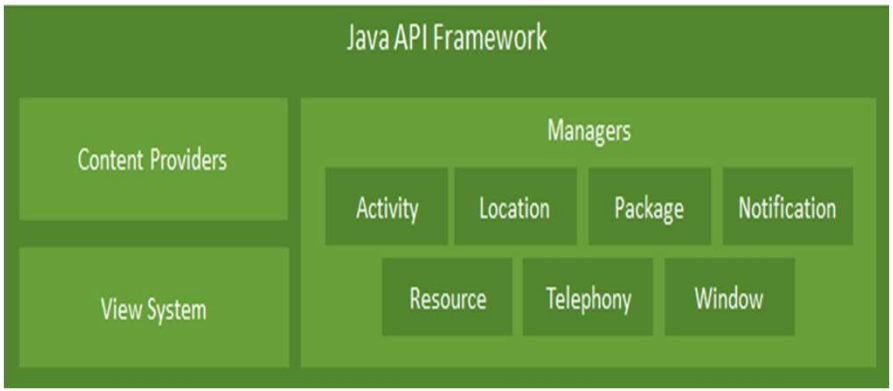
System Apps

- System apps are pre-installed apps in the system partition with your ROM.
- System app is simply an app placed under '**/system/app**' folder on an Android device.
- '/system/app' is a read-only folder.
- Android device users do not have access to this partition. Hence, users cannot directly install or uninstall apps to/from it.
- Apps such as camera, settings, messages, Google Play Store, etc.



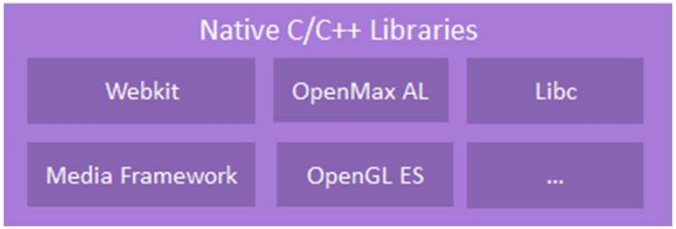
Java Api Framework

- The entire feature-set of the Android OS is available using 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.
- 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.



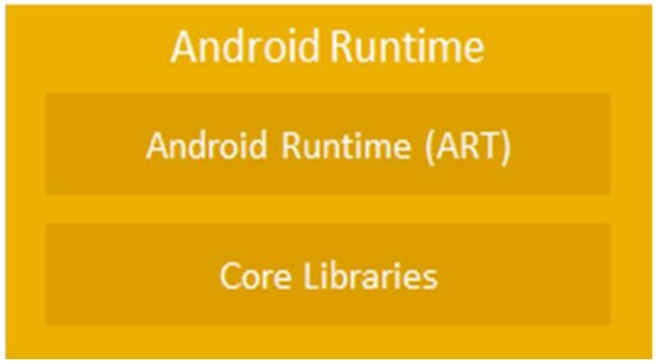
Native C/C++ Libraries

- The Native Development Kit (NDK) is a set of tools that allows you to use C and C++ code with Android.
- It provides **platform libraries** you can use to manage native activities and access physical device components, such as sensors and touch input.
- The NDK may not be appropriate for most novice Android programmers who need to use only Java code and framework APIs to develop their apps.
- However, the NDK can be useful for cases in which you need to do
  - squeeze extra performance out of a device when run games or physics simulations.
  - Reuse your own or other developers' C or C++ libraries.
- Using Android Studio 2.2 and higher, you can use the NDK to compile C and C++ code into a native library and package it into your APK using Gradle



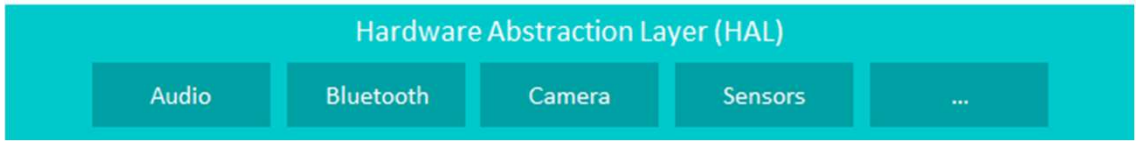
Android Run Time (ART)

- Android runtime (ART) is the managed runtime used by applications and some system services.
- ART and its predecessor Dalvik were originally created specifically for the Android project.
- Google is claiming up to 200% performance improvements overall for ART.
- ART improves garbage collection.



Hardware Abstraction Layer (HAL)

- HAL is responsible for accessing the driver execution hardware.
- A HAL defines a standard interface for hardware vendors to implement, which enables Android to be agnostic about lower-level driver implementations.
- It allows you to implement functionality without affecting or modifying the higher level system.



Linux Kernel

- A kernel is the core part of any operating system.
- Android OS is built on top of the Linux kernel, with some architectural changes made by Google.
- Most importantly, Linux is a portable platform that can be compiled easily on different hardware.
- The kernel acts as an abstraction layer between the software and hardware present on the device.
- Ex. Consider the case of a camera click. What happens when you take a photo using the camera button on your device? At some point, the hardware instruction (pressing a button) has to be converted to a software instruction (to take a picture and store it in the gallery). The kernel contains drivers to facilitate this process.

Android Platforms

- The Android platform is a platform for mobile devices that uses a modified Linux kernel.
- The Android Platform was introduced by the Open Handset Alliance in November of 2007.
- Most applications that run on the Android platform are written in the Java programming language.
- To create an application for the platform, a developer requires the Android SDK, which includes tools and APIs.
- To shorten development time, Android developers typically integrate the SDK.

Android Platforms Contd.

Name	Version Number	Release Date	API Level
Cupcake	1.5	April 27, 2009	3
Donut	1.6	September 15, 2009	4
Eclair	2.0 – 2.1	October 26, 2009	5 – 7
Froyo	2.2 – 2.2.3	May 20, 2010	8
Gingerbread	2.3 – 2.3.7	December 6, 2010	9 – 10
Honeycomb	3.0 – 3.2.6	February 22, 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	October 18, 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	July 9, 2012	16 – 18
KitKat	4.4 – 4.4.4	October 31, 2013	19 – 20
Lollipop	5.0 – 5.1.1	November 12, 2014	21 – 22
Marshmallow	6.0 – 6.0.1	October 5, 2015	23
Nougat	7.0 – 7.1.2	August 22, 2016	24 – 25
Oreo	8.0 – 8.1	August 21, 2017	26 – 27
Pie	9	August 6, 2018	28
Android 10	10	September 3, 2019	29

Hardly anyone use these versions.

Most devices use one of these APIs.

Setup Android Enviroment

- you can start your Android application development on either
  - Mac OS X 10.5.8 or later version with Intel chip.
  - Microsoft Windows XP or later version.
  - Linux including GNU C Library 2.7 or later.
- Java JDK5 or later version
  - You can download the latest version of Java JDK from Oracle's Java site Java SE Downloads.
  - You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup.
  - Finally set PATH and JAVA_HOME environment variable to refer to the directory that contains **java** and **javac.**
- Android Studio
  - You can download the latest version of Android Studio from Developer's site.

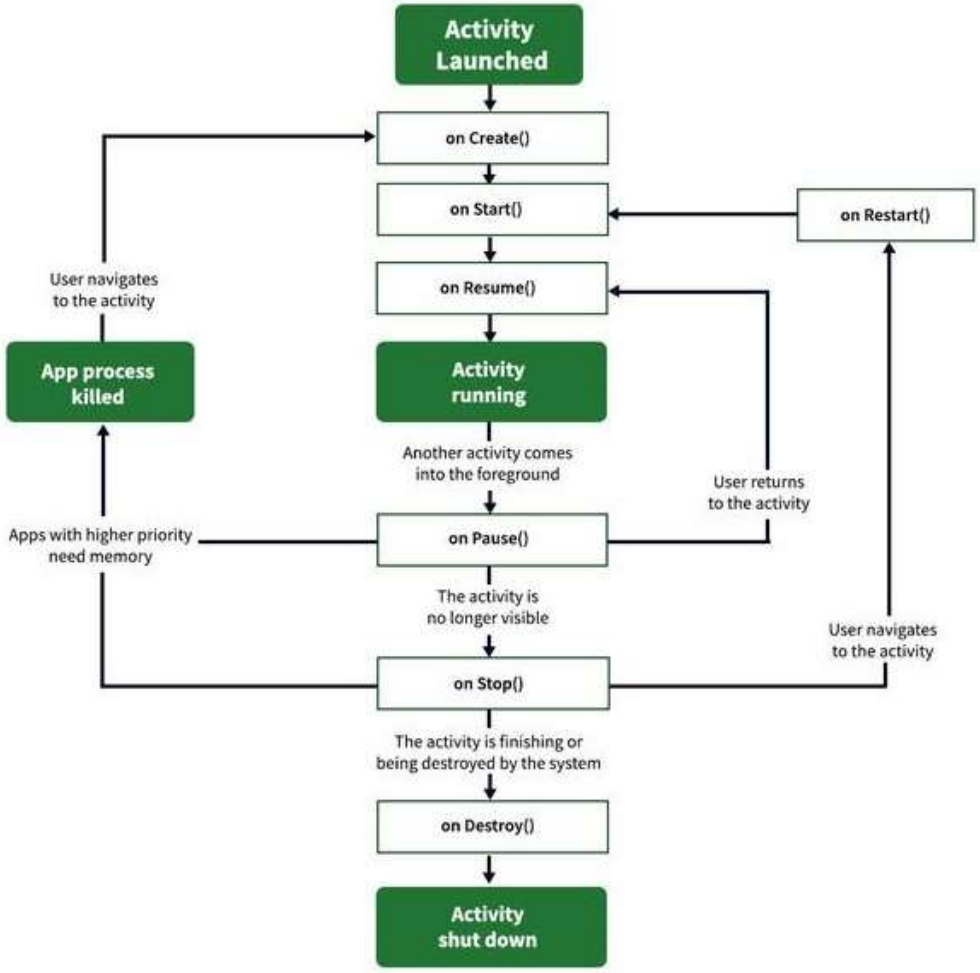
Building Blocks

- The building blocks are components that you use as an developer to build Android apps.
- Main building blocks of android are
  - Activities
    - An activity is usually a single screen that the user sees on the device at one time.
  - Intents
    - Intents are messages that are sent among the major building blocks.
  - Services
    - Services run in the background and don't have any user interface components.
  - Content Providers
    - Content providers are interfaces for sharing data between applications.
  - Broadcast Receivers
    - Broadcast receivers are Android's implementation of a system-wide publish/subscribe mechanism, or more precisely, an Observer pattern.
  - Application Context
    - So far you have seen activities, services, content providers, and broadcast receivers. Together, they make up an application. Another way of saying this is that they live inside the same application context.

Activities

- An application typically has multiple activities, and the user flips back and forth among them.
- As such, activities are the most visible part of your application.
- Just like a website consists of multiple pages, so does an Android application consist of multiple activities.
- As a website has a “home page,” an Android app has a “main” activity, usually the one that is shown first when you launch the application.
- And just like a website has to provide some sort of navigation among various pages, an Android app should do the same.
- We can jump from an activity of one application to another activity in a completely separate application.
  - For example, if you are in your Contacts app and you choose to text a friend, you’d be launching the activity to compose a text message in the Messaging application.

Activity Life Cycle



Activity Life Cycle Contd.

- **onCreate()**
  - It is called when the activity is first created.
  - This is where all the static work is done like creating views, binding data to lists, etc.
  - This method also provides a Bundle containing its previous frozen state, if there was one.

- **onStart()**
  - It is invoked when the activity is visible to the user.
  - It is followed by onResume() if the activity is invoked from the background.
  - It is also invoked after onCreate() when the activity is first started.

- **onRestart()**
  - It is invoked after the activity has been stopped and prior to its starting stage.
  - Thus is always followed by onStart() when any activity is revived from background to on-screen.

Activity Life Cycle Contd.

- **onResume()**
  - It is invoked when the activity starts interacting with the user.
  - At this point, the activity is at the top of the activity stack, with a user interacting with it.
  - Always followed by onPause() when the activity goes into the background or is closed by the user.

- **onPause()**
  - It is invoked when an activity is going into the background but has not yet been killed.
  - It is a counterpart to onResume().
  - When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen).
  - The activity, under the active activity, will not be created until the active activity's onPause() returns, so it is recommended that heavy processing should not be done in this part.

- **onStop()**
  - It is invoked when the activity is not visible to the user.
  - It is followed by **onRestart()** when the activity is revoked from the background, followed by onDestroy().

Activity Life Cycle Contd.

- **onDestroy()**
  - onDestroy() is called before the activity is destroyed.
  - The activity is finishing due to the user completely dismissing the activity or due to finish() being called on the activity.
  - The system is temporarily destroying the activity due to a configuration change ex. device rotation or multi-window mode.

Intents

- An intent is to perform an action on the screen.
- It is mostly used to start activity, send broadcast receiver, start services and send message between two activities.
- It is a messaging object which tells what kind of action to be performed.
- The intent's most significant use is the launching of the activity.
- **Body of Intent:**
  - **action:** The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
  - **data:** The data to operate on, such as a person record in the contacts database, expressed as a Uri
- Basically two intents are there in android.
  - Implicit Intents
  - Explicit Intents

Implicit Intent

- Implicit Intent doesn't specify the component.
- In such a case, intent provides information on available components provided by the system that is to be invoked.
- Implicit intents are used without a class name, where Android will help determine an appropriate Activity to handle the intent.
- For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.javatpoint.com"));
startActivity(intent);
```

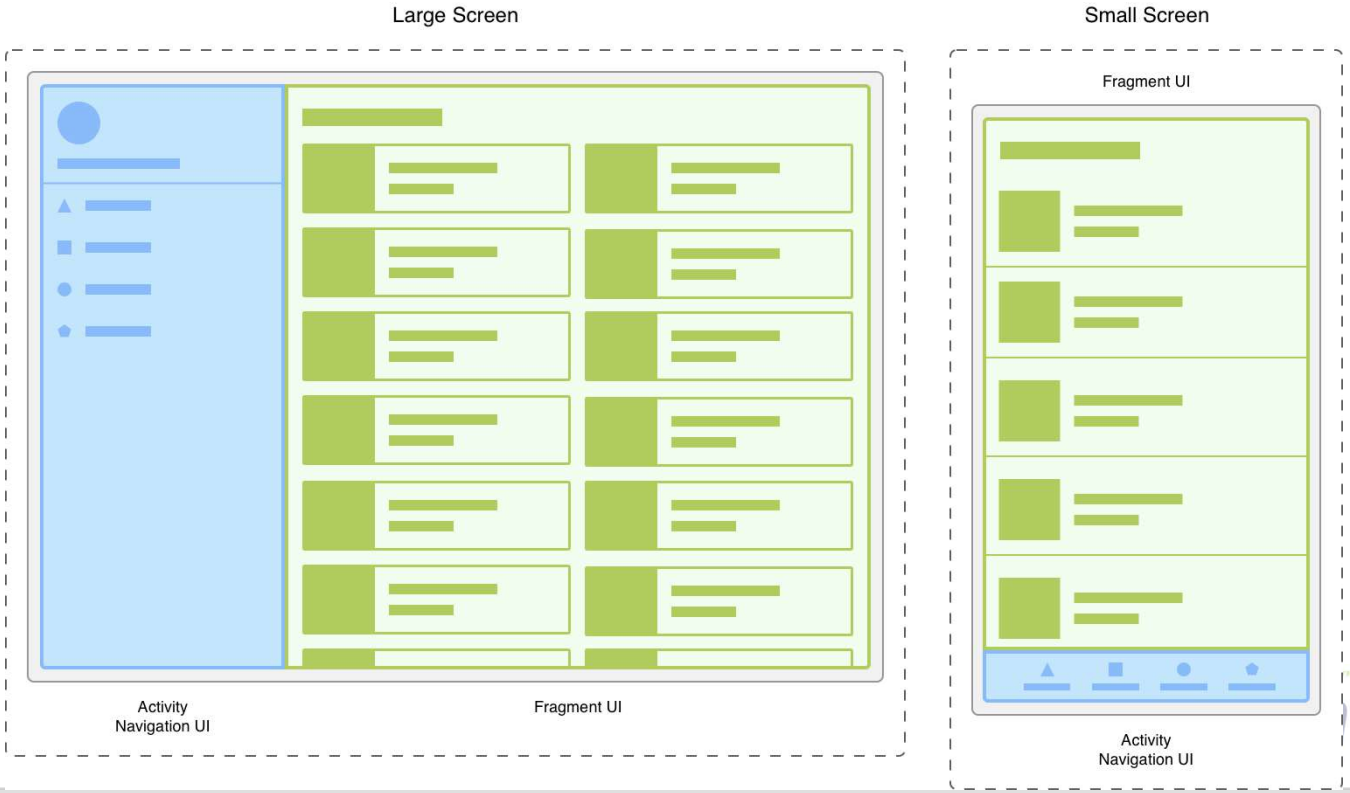
Explicit Intent

- Explicit Intent specifies the component.
- In such a case, intent provides the external class to be invoked.
- Explicit Intent are used with class name, where Android navigate via routes.

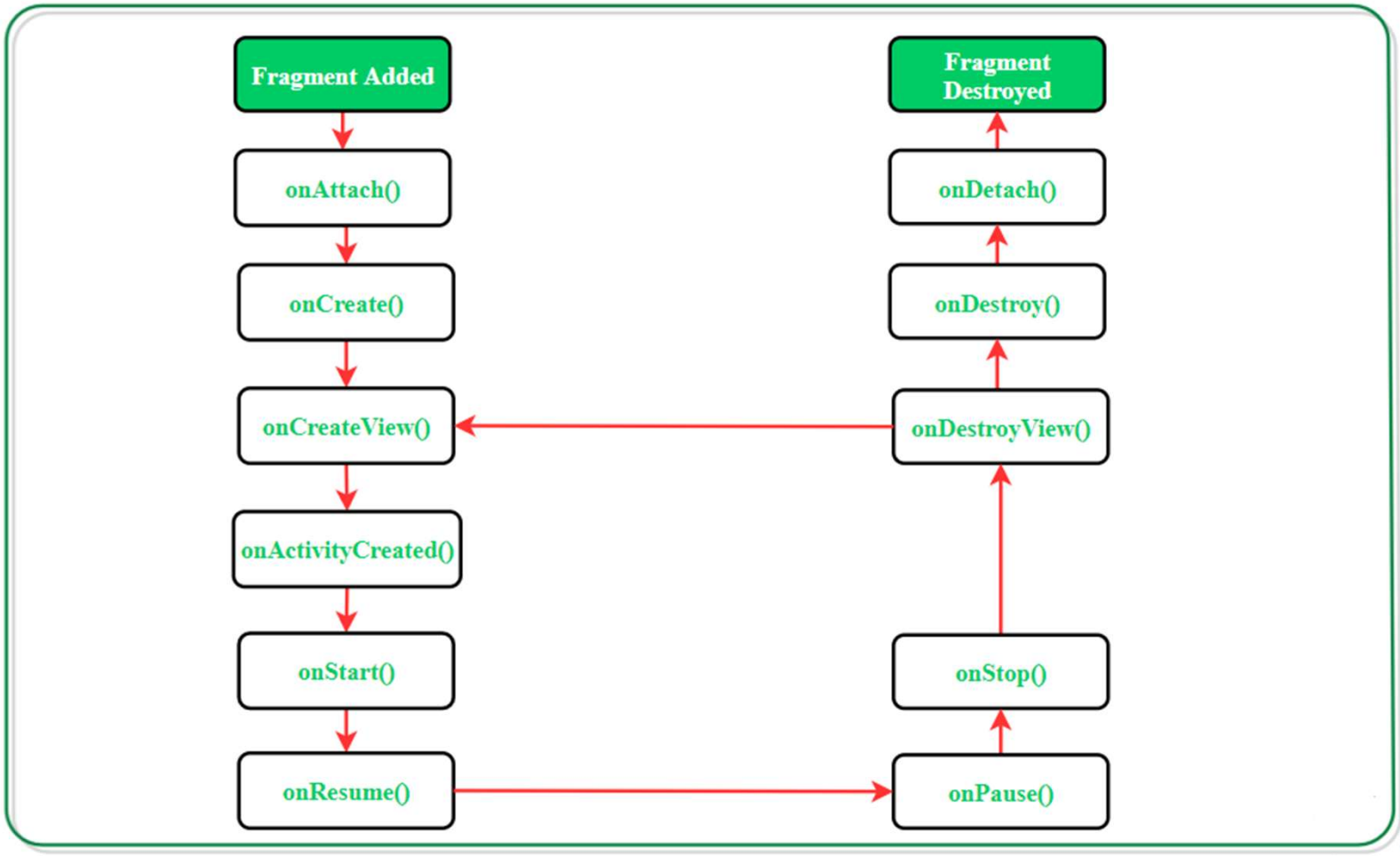
```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

Fragment

- A Fragment represents a reusable portion of app's UI.
- A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events.
- Fragments cannot live on their own they must be hosted by an activity or another fragment.
- Android Fragment is the part of activity, it is also known as sub-activity.
- There can be more than one fragment in an activity.
- Using with fragment transaction. we can move one fragment to another fragment.



Fragment Life Cycle



Fragment Life Cycle

- **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.

Fragment Life Cycle

- **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.

- **onStart():**
  - The onStart() method is called once the fragment gets visible.

- **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.

Fragment Life Cycle

- **onStop():**
  - Fragment going to be stopped by calling onStop().
- **onDestroyView():**
  - Fragment view will destroy after call this method.
- **onDestroy():**
  - onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Thank You

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
+91-9428231065

Mobile Application Development (MAD)
GTU # 3170726

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Android UI

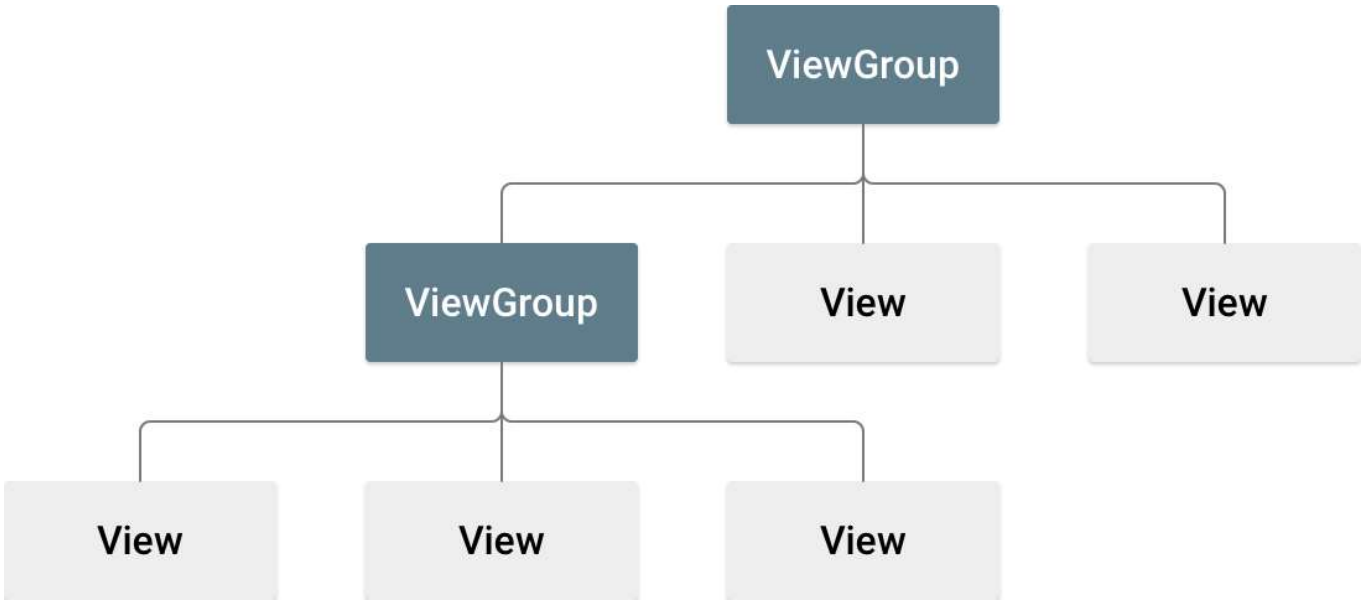
Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
9428231065

Android UI

- Android provides a variety of **Pre-built UI components**.
- Layout objects and UI controls that allow you to build the graphical user interface for your app.
- Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

View Group

- A ViewGroup is a special view that can contain other views (called children.)
- The view group is the base class for layouts and views containers.
- It's child can be **Views** or **View Group**.

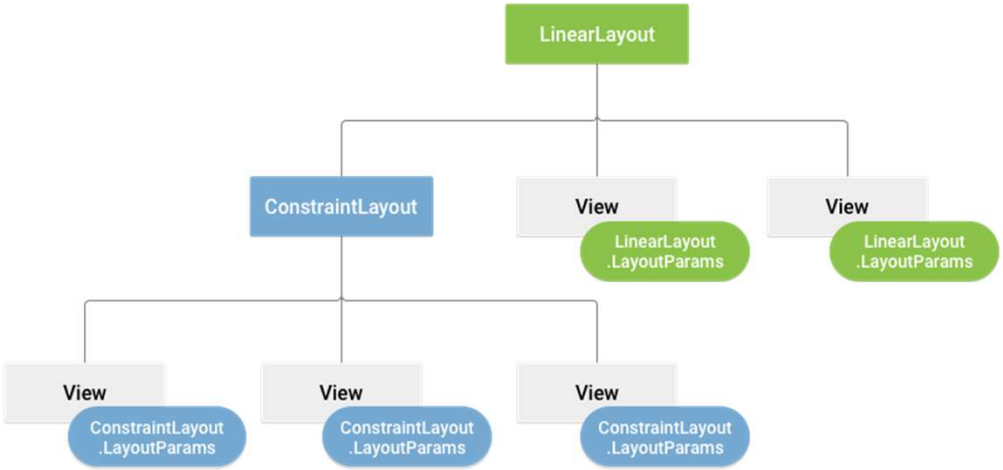


Views

- View class represents the basic building block for user interface components.
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.
- All of the views in a window are arranged in a single tree.
- View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.)

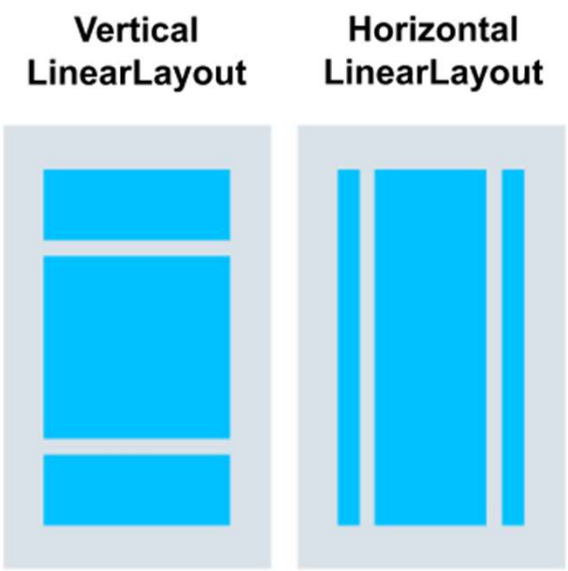
Layouts

- A layout defines the structure for a user interface in your activity.
- All elements in the layout are built using a hierarchy of View and ViewGroup objects.
- A View usually draws something the user can see and interact with.
- **Layout**(View Group) is an invisible container that defines the layout structure for View and other **Layout**(View Group) objects.
- It provide a different layout structure, such as LinearLayout or ConstraintLayout.



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.
- Linear Layout can be created in two direction: **Horizontal** & **Vertical**.
- LinearLayout also supports assigning a **weight** to individual children with the **android:layout_weight** attribute.
- This attribute assigns an important value to a view in terms of how much space it should occupy on the screen.
- A larger weight value allows it to expand to fill any remaining space in the parent view.



Relative Layout

- RelativeLayout is a view group that displays child views in relative positions.
- The position of each view can be specified as relative to sibling elements or in positions relative to the parent RelativeLayout area.
- As it allows us to position the component anywhere, it is considered as most flexible layout.
- Relative layout is the most used layout after the Linear Layout in Android.
- In Relative Layout, you can use “above, below, left and right” to arrange the component’s position in relation to other component.
- In this view group child views can be layered on top of each other.

Frame Layout

- Frame Layout is designed to block out an area on the screen to display a single item.
- It is used to specify the position of Views.
- It contains Views on the top of each other to display only single View inside the FrameLayout.
- You can, add multiple children to a FrameLayout and control their position by using gravity attribute.
- In this the child views are added in a stack and the most recently added child will show on the top.

TextView

- TextView is a UI Component that displays the text to the user on their Display Screen.

Attribution	Description
android:id	This is the ID which uniquely identifies the control.
android:fontFamily	Font family (named by string) for the text.
android:gravity	Specifies how to align the text when the text is smaller than the view.
android:maxHeight	Makes the TextView be at most this many pixels tall.
android:maxWidth	Makes the TextView be at most this many pixels wide.
android:text	Text to display in view.
android:textAllCaps	Make the text in Capital. Possible value either "true" or "false".
android:textColor	Text color. May be a color value, in the form of **"#rgb"**, **"#argb"**, **"#rrggbb"**, or **"#aarrggbb"**.
android:textStyle	Style (bold, italic, bolditalic) for the text. You can use or more of the values separated by '|'. **normal – 0**, **bold – 1**, **italic - 2**
android:typeface	Typeface (normal, sans, serif, monospace) for the text. You can use or more of the values separated by '|'. **normal – 0**, **sans – 1**, **serif – 2**, **monospace – 3**.

EditText

- A EditText is an overlay over TextView that configures itself to be editable.
- It is the predefined subclass of TextView that includes rich editing capabilities.

Attribute	Description
android:autoText	TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:editable	If set, specifies that this TextView has an input method.
android:background	This is a drawable to use as the background.
android:inputType	The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
android:hint	Hint text to display when the text is empty.
android:focusable	It specifies that this edittext gets auto focus or not.

Button

- A Button which can be pressed, or clicked, by the user to perform an action.
- To specify an action when the button is pressed, set a click listener on the button object in the corresponding activity code.
- Every button is styled using the system's default button background it can customize.

Attribute	Description
android:text	This is used to display text on button.
android:background	This is a drawable to use as the background.
android:id	This supplies an identifier name for this view.
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.

Card View

- A CardView is child of FrameLayout with a rounded corner background along with a specific elevation.
- The main usage of CardView is that it helps to give a rich feel and look to the UI design.
- These cards have a default elevation above their containing view group, so the system draws shadows below them.
- Information inside cards that have a consistent look across the platform.
- It can be used for creating items in ListView or inside RecyclerView.

Attribute	Description
card_view:cardBackgroundColor	Can set background color.
card_view:cardCornerRadius	To set Radius or card corner
card_view:cardElevation	To set elevation to tha card
card_view:cardUseCompatPadding	Set compatible padding to cardview based on radius and elevation

ListView

- ListView is a view which groups several items and display them in vertical scrollable list.
- The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.
- An adapter actually acts as a bridge between UI components and the data source that fill data into UI Component.

Attribute	Description
android:divider	This is drawable or color to draw between list items.
android:dividerHeight	This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries	Specifies the reference to an array resource that will populate the ListView.

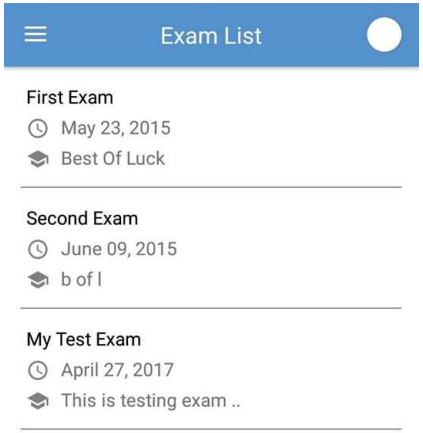
Adapter

- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view.
- In Android development, any time we want to show a vertical list of scrollable items we will use a ListView which has data populated using an Adapter.
- The simplest adapter to use is called an ArrayAdapter because the adapter converts an ArrayList of objects into View items loaded into the ListView container.
- The ArrayAdapter fits in between an ArrayList (data source) and the ListView (visual representation)
- When your ListView is connected to an adapter, the adapter will instantiate rows until the ListView has been fully populated with enough items to fill the full height of the screen. At that point, no additional row items are created in memory.

```
ArrayAdapter<String> itemsAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, items);
```

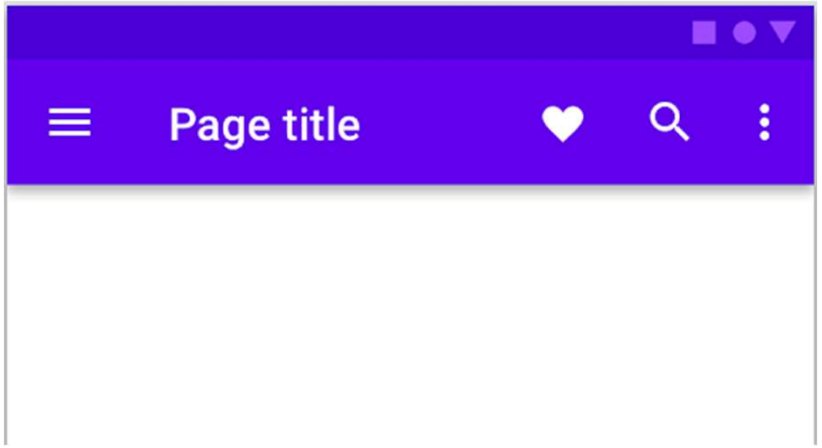
Recycler View

- RecyclerView is a ViewGroup added as a successor of the GridView and ListView.
- RecyclerView is mostly used to design the user interface with the fine-grain control over the lists and grids of android application.
- To implement a basic RecyclerView three sub-parts are needed.
  - 1) The Card Layout : it is an XML layout which will be treated as an item for the list created by the RecyclerView.
  - 2) The ViewHolder: it is a java class that stores the reference to the card layout views that modified during the execution of list.
  - 3) The Data Class: it is a custom java class (getter - setter) that acts as a structure for holding the information for every item of the RecyclerView.
- The adapter is main part for RecyclerView for displaying the list.
  - onCreateViewHolder: which deals with the inflation of the card layout as an item.
  - onBindViewHolder: which deals with the setting of different data and methods related to clicks on particular items.
  - getItemCount: which Returns the length of the RecyclerView.



Material Design Toolbar

- The toolbar gave us so much space for customization and creation for ActionBar.
- MaterialToolbar is a Toolbar that implements certain Material features, such as elevation overlays for Dark Themes and centered titles.
- Toolbars appear a step above the sheet of material affected by their actions.
- Unlike ActionBar, its position is not hardcoded it can be place anywhere according to the need just like any other View in android.
- **Use as an ActionBar:** In an app, the toolbar can be used as an ActionBar in order to provide more customization and a better appearance. All the features of ActionBar such as menu inflation, ActionBarDrawerToogle, etc. are also supported in Toolbar.



Tab Layout

- **TabLayout** is used to implement horizontal tabs.
- Population of the tabs to display is done through TabLayout.Tab instances.

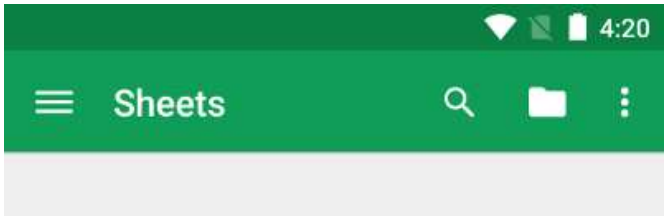
- Create tabs via newTab() From there we can change the tab's label or icon via TabLayout.Tab.setText(int).
- A TabLayout can be setup with a ViewPager
  - Dynamically create TabItems based on the number of pages, their titles, etc.
  - Synchronize the selected tab and tab indicator position with page swipes.
- There are two types of tabs: Fixed tabs, Scrollable tabs.
- 1) Fixed tabs display all tabs on one screen, with each tab at a fixed width.
- 2) Scrollable tabs are displayed without fixed widths. They are scrollable, such that some tabs will remain off-screen until scrolled.

Menus

- To provide a familiar and consistent user experience **menus** are the best.
- It define in separate XML file and use that file in our application based on our requirements.
- We can use menu APIs to represent user actions and other options in our android application activities.
- The Menus in android applications are :
  - 1) Options Menu
  - 2) Context Menu
  - 3) Popup Menu

Options Menu

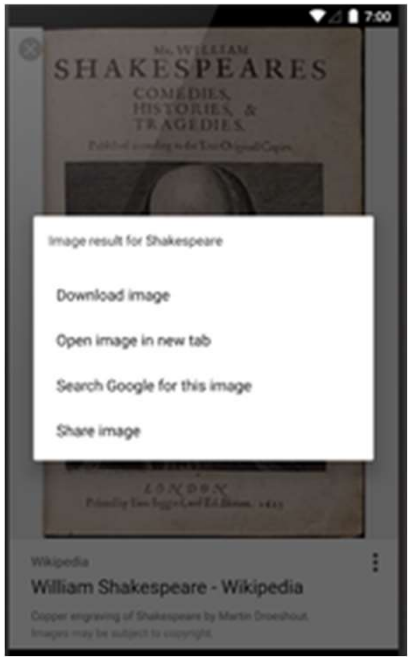
- The options menu is the primary collection of menu items for an activity.
- Options Menu Generally placed on action bar.
- We can declare items for the options menu from either Activity or a Fragment.



- Re-order the menu items with the android:orderInCategory attribute from xml file menu item.
- To specify the options menu for an activity, override onCreateOptionsMenu().
- In this method, you can inflate your menu resource (defined in XML) into the Menu.
- You can also add menu items using add() and retrieve items with findItem().
- When the user clicks a menu item from the options menu onOptionsItemSelected() method is used to get callback.

Contextual Menus

- You can provide a context menu for any view. but they are most often used for items in a RecycerView, ListView, GridView items.
- There are two ways to provide contextual actions.
  - 1) In a floating context menu appears as a floating list of menu items when the user performs a long-click on a view that declares support for a context menu.
  - 2) In the contextual action mode is a system implementation of ActionMode that displays a contextual action bar at the top of the screen with action items ex. Select All Items, Delete Items, etc...
- It can be associate with view using registerForContextMenu() method and pass it the View.
- Menu item inflated to context menu using onCreateContextMenu() method in your Activity or Fragment.
- Menu items click event is handled by onContextItemSelected() in Activity or Fragment.



Popup Menu

- A Popup Menu displays a Menu in a popup window anchored to a View.
- The popup will be shown below the anchored View if there is room(space) otherwise above the View.
- If any Keyboard is visible the popup will not overlap it until the View is touched.
- Touching outside the popup window will dismiss it.

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Thank You

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
+91-9428231065

Mobile Application Development (MAD)
GTU # 3170726

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Database Connectivity

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
9428231065

Storages in Android

- In Android, there are 2 types of storages are available in android.
  - Data storage (Databases, SQLite, Realm, etc..).
  - File storage (SD Card, Phone Storage).
- The system provides several options for you to save your app data.
  - **App-specific storage:** Store files that are only used for your application, either in dedicated directories within an internal storage or different dedicated directories within external storage.
  - **Shared Storage:** Store files that your app intends to share with other apps, including media, documents, and other files.
  - **Preferences:** Store private, primitive data in key-value pairs inside root directories in xml format.
  - **Databases:** Store structured data in a private database using the Room persistence library.
- When storing sensitive data that shouldn't be accessible from any other app use internal storage, preferences, or a database.
- Internal storage has the added benefit of the data being hidden from users.

Shared Preferences

- Shared Preferences can store and retrieve small amounts of data in key/value pairs.
- It can store String, int, float, Boolean in an XML file inside the app.
- It stores data until you clear the preference or uninstall the app from device.
- In order to use shared preferences, you have to call a method **getSharedPreferences().**

```
SharedPreferences sharedpreferences = getSharedPreferences("Settings", Context.MODE_PRIVATE);
```

- The string Settings is the name of the preference file you wish to access If it does not exist, it will be created.
- The mode value designates the default behavior which allow read/write access to only to the application
- Apart from private there are other modes available
  - **MODE_PUBLIC:** will make the file public which could be accessible by other applications on the device.
  - **MODE_PRIVATE:** keeps the files private and secures the user's data.
  - **MODE_APPEND:** it is used while reading the data from the SP file.

Shared Preferences

- If we want to use common preference file and don't wish to specify a file name, we can use default shared preferences.

```
SharedPreferences sharedPreferences = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
```

- Using this way will default the preference file to be stored as
  - /data/data/com.package.name/shared_prefs/com.package.name_preferences.xml
- We can write the data inside shared preference file using **SharedPreferences.Editor.**
- Once editing has been done, one must commit() or apply() the changes made to the file.

Shared Preferences

- There are difference methods available of shared preferences:
  - **contains(String key)**: This method is used to check whether the preferences contain a preference.
  - **edit()**: This method is used to create a new Editor for these preferences.
  - **getAll()**: This method is used to retrieve all values from the preferences.
  - **getBoolean(String key, boolean defValue)**: This method is used to retrieve a boolean value from the preferences.
  - **getFloat(String key, float defValue)**: This method is used to retrieve a float value from the preferences.
  - **getInt(String key, int defValue)**: This method is used to retrieve an int value from the preferences.
  - **getLong(String key, long defValue)**: This method is used to retrieve a long value from the preferences.
  - **getString(String key, String defValue)**: This method is used to retrieve a String value from the preferences.
  - **getStringSet(String key, Set defValues)**: This method is used to retrieve a set of String values from the preferences.

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Thank You

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
+91-9428231065

Mobile Application Development (MAD)
GTU # 3170726

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Applicability to Industrial Projects

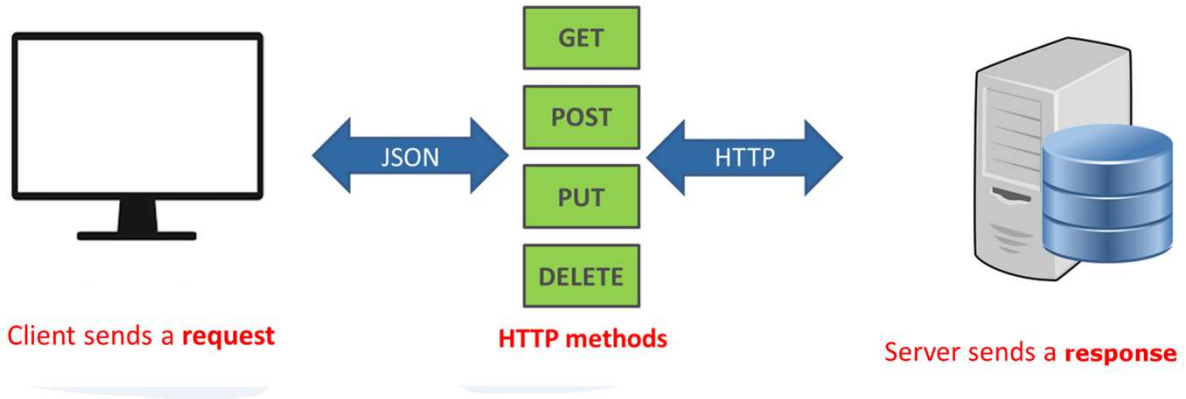
Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
9428231065

Web Services

- The term **Web service** is
  - a service offered by an electronic device to another electronic device, communicating with each other via the World Wide Web.
  - It provides an interface to a database server, A server running on a computer, listening for requests at site.
- In a web service, a web technology such as **HTTP** is used for transferring machine-readable file formats such as XML and JSON.
- AJAX is a dominant technology for web service developing
  - It uses the combination of HTTP servers and JavaScript clients.
  - It provides results in **JSON** or **XML** format.
- **REST** (Representational State Transfer) is an architecture used for Web services that can work at Internet scale.

Web API (Application Programming Interface)

- API stands for Application Programming Interface.
- Web API can be accessed over the web using the HTTP protocol.
- It is a framework that helps you to create and develop HTTP based RESTFUL services.
- Restful APIs do not require XML-based Web service protocols (SOAP and WSDL) to support their interfaces.
- The web API can be developed by using different technologies such as java, ASP.NET, php, etc.



Web API

- Web API receives requests from different types of client devices like mobile, laptop, etc.
- It sends those requests to the webserver to process those requests and returns output to the client.
- Web API is a system to system interaction in which the data or information from one system can be accessed by another system.
- After the completion of execution the resultant data or we can say as output is shown to the viewer.

Service vs API

Web Service	Web API
All web services are APIs.	All APIs are not web services.
It supports only XML.	Response are formatted into XML, JSON, or any other format.
It uses SOAP protocol to send or receive data over the network.	It uses HTTP Protocol to send or receive data over the network.
It is not a light-weight architecture.	API has a light-weight architecture.
It can be used by any client which understands XML.	It can be used by a client which understands JSON or XML.
Web service uses three styles: REST, SOAP, and XML-RPC for communication.	API can be used for any style of communication.

JSON Parsing

- **JSON** (JavaScript Object Notation) is a straightforward data exchange format to interchange the server's data.
- JSON is a lightweight and structured language.
- Android supports all the JSON classes such as JSONStringer, JSONObject, JSONArray, and all other forms to parse the JSON data.
- JSON nodes will start with a square bracket [or with a curly bracket {.
- Square bracket [represents the beginning of a **JSONArray node**.
- Curly bracket { represents a **JSONObject.**

```
{
"Name": "Mehul",
"Estd": 2009,
"age": 10,
"address": {
    "buildingAddress": "5th & 6th Floor Royal Kapsons, A- 118",
    "city": "Sector- 136, Noida",
    "state": "Uttar Pradesh (201305)",
    "postalCode": "201305"
},
```

JSON Parsing

- The elements inside the curly ({}) are known as JSON Objects.
- A list of values inside the square ([]) are known as JSON Array.
- Data is stored as a pair of keys and values.
- The keys can be a name, a number for which the values can be Mehul, 942XXXX etc.
- The functions which we need for JSON parsing are:
  - **get(int index) :** It gets the value of the object type presented in the JSON array.
  - **length() :** we get the length of the array that is received from the server.
  - **getString(String key) :** we can get string value from JSON object.
  - **getJSONObject() :** we can get JSON Object from JSON array or JSON object.

JSON Parsing Example:

JSON Response

```
{
"Name": "Mehul",
"Estd": 1991,
"age": 30,
"address": {
        "buildingAddress": "5th & 6th Floor Royal Kapsons, A- 118",
        "city": "Sector- 136, Noida",
        "state": "Uttar Pradesh (201305)",
        "postalCode": "201305"
        }
}
```

Parsing

```
JSONObject jsonObject = new JSONObject(response);
String name = jsonObject.getString("Name");
String year = jsonObject.getString("Estd");
int age= jsonObject.getInt("age");
JSONObject addressObject = jsonObject.getJSONObject("address");
String buildingAddress = addressObject.getString("buildingAddress");
String city = addressObject.getString("city");
String state = addressObject.getString("state");
String postalCode = addressObject.getString("postalCode");
```

Address JSON Object

AsyncTask

- Asynctask perform heavy tasks in the background independently.
- The basic methods used in an android AsyncTask class are as below;

Method	Description
doInBackground()	✓ This method contains the code which needs to be executed in background. ✓ In this method we can send results multiple times to the UI thread by publishProgress() method. ✓ To notify that the background processing has been completed we just need to use the return statements
onPreExecute()	✓ This method contains the code which is executed before the background processing starts (doInBackground()).
onPostExecute()	✓ This method is called after doInBackground() method completes processing. ✓ Result from doInBackground() is passed to this method.
onProgressUpdate()	✓ This method receives progress updates from doInBackground() method. ✓ This method can use this progress update to update the UI thread.

Example (AsyncTask)

doInBackground()

```
1         @Override
2         protected String doInBackground(String... params) {
3                 publishProgress("Sleeping...");
4                 onProgressUpdate()
5                 try {
6                         int time = Integer.parseInt(params[0])*1000;
7                         Thread.sleep(time);
8                         resp = "Slept for " + params[0] + " seconds";
9                 } catch (InterruptedException e) {
10                        e.printStackTrace();
11                        resp = e.getMessage();
12                } catch (Exception e) {
13                        e.printStackTrace();
14                        resp = e.getMessage();
15                }
16                return resp;
17        }
```

Example

onPreExecute()

```
1       @Override
2       protected String onPreExecute() {
3              super.onPreExecute();
4              pd = new ProgressDialog(this);
5              pd.setMessage("Please wait...It is downloading");
6              pd.setIndeterminate(false);
7              pd.setCancelable(false);
8              pd.show();
9       }
```

onPostExecute()

```
1       @Override
2       protected String onPostExecute(String result) {
3              super.onPostExecute();
4              pd.dismiss();
5       }
```

API call using Asynctask

```
class JSONAsyncTask extends AsyncTask<String, Void, Boolean> {

        @Override
        protected Boolean doInBackground(String... urls) {
                try {
                        HttpGet httppost = new HttpGet("http://xxx.xxx.xxx");
                        HttpClient httpclient = new DefaultHttpClient();
                        HttpResponse response = httpclient.execute(httppost);
                        StatusLine stat = response.getStatusLine();
                        int status = response.getStatusLine().getStatusCode();
                        if (status == 200) {
                                HttpEntity entity = response.getEntity();
                                String data = EntityUtils.toString(entity);
                                JSONObject json = new JSONObject(data);
                                return true;
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }
                catch (JSONException e) {
                        e.printStackTrace();
                } return false;
        }
```

Retrofit

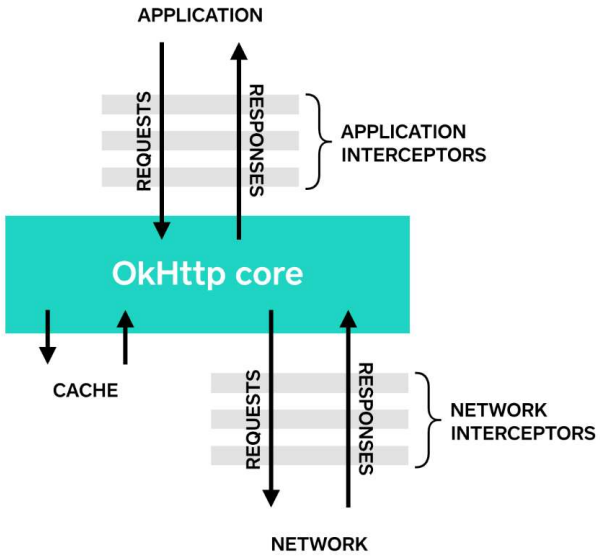
- **Retrofit** is a library developed by Square to handle REST API calls in our android application.
- Retrofit is a type-safe REST client for Android and Java which aims to make it easier to consume RESTful web services.
- Retrofit automatically serialize the JSON response using a model class(getter,setter) which must be defined in advanced for the JSON Structure.
- To serialize JSON we need a converter to convert it into Gson first then convert to model.
- Retrofit by default uses **OkHttp** as the networking layer and is built on top of it.
- Retrofit provides with a list of annotations for each of the HTTP methods: @GET, @POST, @PUT, @DELETE, @PATCH or @HEAD.

Dependencies

```
compile 'com.squareup.retrofit2:retrofit:x.x.x'
compile 'com.google.code.gson:gson:x.x.x'
compile 'com.squareup.retrofit2:converter-gson:x.x.x'
```

OkHttp Interceptors

- Interceptors are a powerful mechanism present in OkHttp that can monitor, rewrite, and retry calls.
  - **Application Interceptors :** It is used to get to logs of application call logs. To register an application interceptor, we need to call addInterceptor() on OkHttpClient.Builder
  - **Network Interceptors :** It will gives logs of request and response of API call. To register a Network Interceptor, invoke addNetworkInterceptor() instead of addInterceptor()



Retrofit Implementation

Api Client

```
class APIClient {
        private static Retrofit retrofit = null;
        static Retrofit getClient() {
                HttpLoggingInterceptor interceptor = new HttpLoggingIntercepto();
                interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
                OkHttpClient client = new OkHttpClient.Builder();
                client.addInterceptor(interceptor).build();
                retrofit = new Retrofit.Builder();
                retrofit.baseUrl("https://reqres.in");
                retrofit.addConverterFactory(0());
                retrofit.client(client).build();
                return retrofit;
        }
}
```

- The getClient() method in the above code will be called every time while setting up a Retrofit interface.

Api Interface

```
        @GET("/api/unknown")
        Call<MultipleResource> doGetListResources();
```

Api Call

Api Call

```
APIInterface apiInterface = APIClient.getClient().create(APIInterface.class);
Call<MultipleResource> call = apiInterface.doGetListResources();
call.enqueue(new Callback<MultipleResource>() {
        @Override
        public void onResponse(Call<MultipleResource> call, Response<MultipleResource> response) {
                Log.d("TAG",response.code()+"");
                String displayResponse = "";
                MultipleResource resource = response.body();
                Integer text = resource.page;
                Integer total = resource.total;
                Integer totalPages = resource.totalPages;
                List<MultipleResource.Datum> datumList = resource.data;
                displayResponse += text + " Page\n" + total + " Total\n" + totalPages
                + " Total Pages\n";
                for (MultipleResource.Datum datum : datumList) {
                        displayResponse += datum.id + " " + datum.name + " " + datum.pantoneValue
                        + " " + datum.year + "\n";
                }
                responseText.setText(displayResponse);
        }
```

Api Call (Contd..)

Api Call

```
20          @Override
21          public void onFailure(Call<User> call, Throwable t) {
22                  call.cancel();
23          }
24  });
```

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Thank You

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
+91-9428231065

Mobile Application Development (MAD)
GTU # 3170726

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Advanced Android Development

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
9428231065

Google Map

- Google Map
  - Location Service and GPS
  - Creating Google Map, Working with Location
  - Location Service with Location Manager
  - Find Current Location and GEO Coding
- Graphics and Animation
  - Work with 2D Graphics
  - Animation
  - Frame Animation, Tween Animation, View Animation
- Multimedia in Android
  - Play Audio Files, and Video Files
- Work in Background
  - Services, Notification Services, and Broadcast Receiver
- Introduction to Firebase with CRUD Operation

Google Map

Section - 1

Build location-aware Apps

- Mobile users take their devices with them everywhere, and adding location awareness to your app offers users a more contextual experience.
- The location APIs available in Google Play services facilitate adding location awareness to your app with automated location tracking, wrong-side-of-the-street detection, geofencing, and activity recognition.
- **Request location permissions**
  - To protect user privacy, apps that use location services must request location permissions.
  - It includes multiple permissions related to location.
  - Which permissions you request, and how you request them, depend on the location requirements for your app's use case.

Types of Location Access

- Each permission has a combination of the following characteristics:
- **Category**: Either foreground location or background location.
  - ***Foreground Location***: If your app contains a feature that shares or receives location information only once, or for a defined amount of time, then that feature requires foreground location access.
  - ***Background Location***: An app requires background location access if a feature within the app constantly shares location with other users or uses the *Geofencing API*.
- **Accuracy**: Either *approximate* location or *precise* location.
  - ***Approximate***: Provides an estimate of the device's location, to within about 1 mile (1.6 km).

```
Permission required is ACCESS_COARSE_LOCATION
```

  - ***Precise***: Provides an estimate of the device's location that is as accurate as possible, usually within about 160 feet (50 meters) and sometimes as accurate as within 10 feet (a few meters) or better.

```
Permission required is ACCESS_FINE_LOCATION
```

Permission Request

Code

```
ActivityResultLauncher<String[]> locationPermissionRequest =
    registerForActivityResult(new ActivityResultContracts
        .RequestMultiplePermissions(), result -> {
            Boolean fineLocationGranted = result.getOrDefault(
                    Manifest.permission.ACCESS_FINE_LOCATION, false);
            Boolean coarseLocationGranted = result.getOrDefault(
                    Manifest.permission.ACCESS_COARSE_LOCATION,false);
            if (fineLocationGranted != null && fineLocationGranted) {
                // Precise location access granted.
            } else if (coarseLocationGranted != null && coarseLocationGranted) {
                // Only approximate location access granted.
            } else {
                // No location access granted.
            }
        }
    );

// ...
locationPermissionRequest.launch(new String[] {
    Manifest.permission.ACCESS_FINE_LOCATION,
    Manifest.permission.ACCESS_COARSE_LOCATION
});
```

Get the last known location

- Using the Google Play services location APIs, your app can request the last known location of the user's device.
- Use the **fused location provider** to retrieve the device's last known location.
- The **fused location provider** is one of the location **API**s in Google Play services.
- It manages the underlying location technology and provides a simple API so that you can specify requirements at a high level, like high accuracy or low power.
- It also optimizes the device's use of battery power.

Example

- To access the **fused location provider**, your app's development project must include **Google Play services**. Download and install the Google Play services component via the SDK Manager and add the library to your project.
- Apps whose features use location services must request location permissions, depending on the use cases of those features.
- In your activity's onCreate() method, create an instance of the **Fused Location Provider Client** as the following code snippet shows.

Code

```
1  private FusedLocationProviderClient fusedLocationClient;
2
3  @Override
4  protected void onCreate(Bundle savedInstanceState) {
5
6          fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
7  }
```

Fused Location

Code

```
1  fusedLocationClient.getLastLocation()
2          .addOnSuccessListener(this, new OnSuccessListener<Location>() {
3              @Override
4              public void onSuccess(Location location) {
5                  // Got last known location. In some rare situations this can be null.
6                  if (location != null) {
7                      // Logic to handle location object
8                  }
9              }
10         });
```

Geo Coding

- ***Geocoding*** is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate.
- ***Reverse geocoding*** is the process of transforming a (latitude, longitude) coordinate into a (partial) address.
- The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code.

Graphics and Animation

Section - 2

Working with 2D Graphics

- Use the **Drawable** class and its subclasses to draw shapes and images.
- There are **two** ways to define and instantiate a **Drawable** besides using the class constructors:
  - Inflate an image resource (a bitmap file) saved in your project.
  - Inflate an XML resource that defines the drawable properties.
- You can add graphics to your app by referencing an image file from your project resources.
- Supported file types are **PNG** (*preferred*), **JPG** (*acceptable*), and **GIF** (*discouraged*).

Example code to use an image created from drawable resource

Code

```
1  ConstraintLayout constraintLayout;
2  protected void onCreate(Bundle savedInstanceState) {
3       super.onCreate(savedInstanceState);
4
5       // Create a ConstraintLayout in which to add the ImageView
6       constraintLayout = new ConstraintLayout(this);
7
8       // Instantiate an ImageView and define its properties
9        ImageView i = new ImageView(this);
10       i.setImageResource(R.drawable.my_image);
11       i.setContentDescription(getResources().getString(R.string.my_image_desc));
12
13
14       // set the ImageView bounds to match the Drawable's dimensions
15       i.setAdjustViewBounds(true);
16       i.setLayoutParams(new
17 ViewGroup.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
18 ViewGroup.LayoutParams.WRAP_CONTENT));
19
20
21       // Add the ImageView to the layout and set the layout as the content view.
22       constraintLayout.addView(i);
23       setContentView(constraintLayout);
24 }
```

Animation with Image View

- You can handle your image resource as a Drawable object as;

Code

```
1 Resources res = context.getResources();
2 Drawable myImage = ResourcesCompat.getDrawable(res, R.drawable.my_image, null);
```

- The XML code below shows how to add a drawable resource to an ImageView in the XML layout:

Code

```
1 <ImageView
2         android:layout_width="wrap_content"
3         android:layout_height="wrap_content"
4         android:src="@drawable/my_image"
5         android:contentDescription="@string/my_image_desc" />
```

Create drawables with XML resourses

- If there is a **Drawable** object that you'd like to create, which isn't initially dependent on variables defined by your code or user interaction, then defining the Drawable in XML is a good option.
- Even if you expect your **Drawable** to change its properties during the user's interaction with your app, you should consider defining the object in XML, as you can modify properties after it's instantiated.
- After you've defined your **Drawable** in XML, save the file in the *res/drawable/* directory of your project.
- The following example shows the XML that defines a **TransitionDrawable** resource, which inherits from **Drawable**:

Code

```
1 <!-- res/drawable/expand_collapse.xml -->
2 <transition
3 xmlns:android="http://schemas.android.com/apk/res/android">
4         <item android:drawable="@drawable/image_expand"/>
5         <item android:drawable="@drawable/image_collapse"/>
6 </transition>
```

Animation with Image View

- The following code instantiates the **TransitionDrawable** and sets it as the content of an **ImageView** object:

Code

```
1  Resources res = context.getResources();
2  TransitionDrawable transition =
3      (TransitionDrawable) ResourcesCompat.getDrawable(res,
4  R.drawable.expand_collapse, null);

5  ImageView image = (ImageView) findViewById(R.id.toggle_image);
6  image.setImageDrawable(transition);

7  // Description of the initial state that the drawable represents.
8  image.setContentDescription(getResources().getString(R.string.collapsed));

9  // Then you can call the TransitionDrawable object's methods.
10 transition.startTransition(1000);

11 // After the transition is complete, change the image's content description
12 // to reflect the new state.
```

Animation

- Animations can add visual cues that notify users about what's going on in your app.
- They are especially useful when the UI changes state, such as when new content loads or new actions become available.
- Animations also add a polished look to your app, which gives it a higher quality look and feel.

Animation Resources

- An animation resource can define one of **two** types of animations:
  - **Property Animation**: Creates an animation by modifying an object's property values over a set period of time with an Animator.
  - **View Animation:** There are two types of animations that you can do with the view animation framework
    - ***Tween animation***: Creates an animation by performing a series of transformations on a single image with an Animation
    - ***Frame animation***: or creates an animation by showing a sequence of images in order with an AnimationDrawable.

Tween Animation

- An animation defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic.

File location	***res/anim/filename.xml*** **The filename will be used as the resource ID.**
Compiled resource datatype	Resource pointer to an Animation.
Resource reference	In Java: R.anim.filename In XML: @[package:]anim/filename

Example

Code

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"] >
    <alpha
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
    <set>
        ...
    </set>
</set>
```

Example

Code

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale

android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
    <set
        android:interpolator="@android:anim/accelerate_interpolator"
        android:startOffset="700">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
```

Code

```
 <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
    </set>
</set>
```

Animation with Image View

- Apply the animation to an ImageView and start the animation:

Code

```
1 ImageView image = (ImageView)
2 findViewById(R.id.image);
3 Animation hyperspaceJump =
4 AnimationUtils.loadAnimation(this,
5 R.anim.hyperspace_jump);
6 image.startAnimation(hyperspaceJump);
```

Frame Animation

- An animation defined in XML that shows a sequence of images in order (like a film).

File location	***res/drawable/filename.xml*** **The filename will be used as the resource ID.**
Compiled resource datatype	Resource pointer to an AnimationDrawable.
Resource reference	In Java: R.drawable.filename In XML: @[package:]drawable.filename

Example

Code

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/rocket_thrust1"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3"
android:duration="200" />
</animation-list>
```

Example

Code

```
1 ImageView rocketImage = (ImageView)
2 findViewById(R.id.rocket_image);
3 rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
4
5 rocketAnimation = rocketImage.getBackground();
6 if (rocketAnimation instanceof Animatable) {
7     ((Animatable)rocketAnimation).start();
8 }
```

Multimedia in Android

Section - 3

Android Multimedia

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

MediaPlayer	This class is the primary API for playing sound and video.
AudioManager	This class manages audio sources and audio output on a device.

Manifest declarations

- **Internet Permission** - If you are using MediaPlayer to stream network-based content, your application must request network access.

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

- **Wake Lock Permission** - If your player application needs to keep the screen from dimming or the processor from sleeping, or uses the MediaPlayer.setScreenOnWhilePlaying() or MediaPlayer.setWakeMode() methods, you must request this permission.

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

- Example to play audio that's available as a local raw resource

```
MediaPlayer mediaPlayer = MediaPlayer.create(context,
R.raw.sound_file_1);
mediaPlayer.start();
```

Example

Code

```
1  Uri myUri = ....; // initialize Uri here
2  MediaPlayer mediaPlayer = new MediaPlayer();
3  mediaPlayer.setAudioAttributes(new AudioAttributes.Builder()
4          .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
5          .setUsage(AudioAttributes.USAGE_MEDIA)
6          .build()
7  );
8  mediaPlayer.setDataSource(getApplicationContext(), myUri);
9  mediaPlayer.prepare();
10 mediaPlayer.start();
```

Animation with Image View

- Playing from a remote URL via HTTP streaming

Code

```
String url = "http://........"; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioAttributes(
    new AudioAttributes.Builder()
        .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
        .setUsage(AudioAttributes.USAGE_MEDIA)
        .build()
);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

Work in Background

Section - 4

Services

- A Service is an application component that can perform long-running operations in the background.
- It does not provide a user interface.
- Once started, a service might continue running for some time, even after the user switches to another application.
- Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC).
- E.g., a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Types of Services

Foreground	A **foreground** service performs some operation that is noticeable to the user. E.g., an audio app would use a foreground service to play an audio track. Foreground services must display a Notification. Foreground services continue running even when the user isn't interacting with the app.
Background	A **background** service performs an operation that isn't directly noticed by the user. E.g., if an app used a service to compact its storage, that would usually be a background service.
Bound	A **service** is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Service Basics

- To create a service, you must create a subclass of Service or use one of its existing subclasses.
- In your implementation, you must override some **callback methods** that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service.

Callback Methods

onStartCommand()	• The system invokes this method by calling *startService*() when another component (such as an activity) requests that the service be started. • When this method executes, the service is started and can run in the background indefinitely. • It is your responsibility to stop the service when its work is complete by calling *stopSelf*() or *stopService*(). • If you only want to provide binding, you don't need to implement this method.
onBind()	• The system invokes this method by calling bindService() when another component wants to bind with the service (to perform RPC). • In your implementation, you must provide an interface that clients use to communicate with the service by returning an IBinder. • You must always implement this method; however, if you don't want to allow binding, you should return null.
onCreate()	• The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either *onStartCommand*() or *onBind*()). • If the service is already running, this method is not called.
onDestroy()	• The system invokes this method when the service is no longer used and is being destroyed. • Your service should implement this to clean up any resources such as threads, registered listeners, or receivers. This is the last call that the service receives.

Declaring a service in the manifest

- You must declare all services in your application's manifest file, just as you do for activities and other components.
- To declare your service, add a <service> element as a child of the <application> element.

Code

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Example

Code

```
1  public class HelloService extends Service {
2    private Looper serviceLooper;
3    private ServiceHandler serviceHandler;
4
5
6    // Handler that receives messages from the thread
7    private final class ServiceHandler extends Handler {
8        public ServiceHandler(Looper looper) {
9            super(looper);
10       }
11       @Override
12       public void handleMessage(Message msg) {
13           // Normally we would do some work here, like download a file.
14           // For our sample, we just sleep for 5 seconds.
15           try {
16               Thread.sleep(5000);
17           } catch (InterruptedException e) {
18               // Restore interrupt status.
19               Thread.currentThread().interrupt();
20           }
21           // Stop the service using the startId, so that we don't stop
22           // the service in the middle of handling another job
23           stopSelf(msg.arg1);
24       }
25   }
```

Example

Code

```
@Override
  public void onCreate() {
    HandlerThread thread = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();

    serviceLooper = thread.getLooper();
    serviceHandler = new ServiceHandler(serviceLooper);
  }


  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
      Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

      Message msg = serviceHandler.obtainMessage();
      msg.arg1 = startId;
      serviceHandler.sendMessage(msg);

      return START_STICKY;
  }
```

Example

Code

```
  @Override
  public IBinder onBind(Intent intent) {
      // We don't provide binding, so return null
      return null;
  }

  @Override
  public void onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
  }
}
```

An activity can start the example service in the previous section (HelloService) using an explicit intent with *startService*(), as shown here:

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Broadcast Service

- Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern.
- These broadcasts are sent when an event of interest occurs.
- E.g., the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging.
- Apps can also send custom broadcasts, e.g., some new data has been downloaded.
- Apps can register to receive specific broadcasts.
- When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.
- Apps can receive broadcasts in two ways:
  - **Manifest-declared receivers**: If you declare a broadcast receiver in your manifest, the system launches your app (if the app is not already running) when the broadcast is sent.
  - **Context-registered receivers**

Steps to declare broadcast receiver in the manifest

- Specify the <receiver> element in your app's manifest. The intent filters specify the broadcast actions your receiver subscribes to.

Code

```
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
<intent-filter>
     <action android:name="android.intent.action.BOOT_COMPLETED"/>
     <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
</intent-filter>
</receiver>
```

- Subclass **BroadcastReceiver** and implement *onReceive*(**Context**, **Intent**).

Example

Code

```
1  public class MyBroadcastReceiver extends BroadcastReceiver {
2          private static final String TAG = "MyBroadcastReceiver";
3          @Override
4          public void onReceive(Context context, Intent intent) {
5              StringBuilder sb = new StringBuilder();
6              sb.append("Action: " + intent.getAction() + "\n");
7              sb.append("URI: " + intent.toUri(Intent.URI_INTENT_SCHEME).toString()
8  + "\n");
9              String log = sb.toString();
10             Log.d(TAG, log);
11             Toast.makeText(context, log, Toast.LENGTH_LONG).show();
12         }
13     }
```

- The system package manager registers the receiver when the app is installed.
- The receiver then becomes a separate entry point into your app which means that the system can start the app and deliver the broadcast if the app is not currently running.

Broadcast Service

- The system creates a new **BroadcastReceiver** component object to handle each broadcast that it receives.
- This object is valid only for the duration of the call to *onReceive*(**Context**, **Intent**).
- Once your code returns from this method, the system considers the component no longer active.

Context-registered receivers

- To register a receiver with a context, perform the following steps:
  - Create an instance of **BroadcastReceiver**.

```
BroadcastReceiver br = new MyBroadcastReceiver();
```

  - Create an **IntentFilter** and register the receiver by calling *registerReceiver*(**BroadcastReceiver**, **IntentFilter**):

```
IntentFilter filter = new
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
this.registerReceiver(br, filter);
```

  - To stop receiving broadcasts, call *unregisterReceiver*(android.content.**BroadcastReceiver**).
- **Note**: Be sure to unregister the receiver when you no longer need it or the context is no longer valid.

Sending Broadcasts

- Android provides three ways for apps to send broadcast:
  - The *sendOrderedBroadcast*(**Intent**, **String**) method sends broadcasts to one receiver at a time.
  - The *sendBroadcast*(**Intent**) method sends broadcasts to all receivers in an undefined order. This is called a Normal Broadcast.
  - The **LocalBroadcastManager**.*sendBroadcast* method sends broadcasts to receivers that are in the same app as the sender. If you don't need to send broadcasts across apps, use local broadcasts.

Sending Broadcasts

- **Sample code to send a broadcast by creating an Intent and calling *sendBroadcast*(Intent).**

```
Intent intent = new Intent();
intent.setAction("com.example.broadcast.MY_NOTIFICATION");
intent.putExtra("data", "Nothing to see here, move along.");
sendBroadcast(intent);
```

Notifications

- A notification is a message that Android displays outside your app's UI to provide the user with reminders, communication from other people, or other timely information from your app.
- Users can tap the notification to open your app or take an action directly from the notification.
- Since **Android 1.0**, the notification system UI and the notification-related APIs have continually evolved.
- To use the latest notification API features while still supporting older devices, use the support library notification API: **NotificationCompat** and its subclasses, as well as **NotificationManagerCompat**.
- A notification in its most basic and compact form (also known as collapsed form) displays an icon, a title, and a small amount of content text.
- Set the notification's content and channel using a **NotificationCompat**.**Builder** object.

Example

- The following example shows how to create a notification with the following:

Code

```
1 NotificationCompat.Builder builder = new
2 NotificationCompat.Builder(this, CHANNEL_ID)
3         .setSmallIcon(R.drawable.notification_icon)
4         .setContentTitle(textTitle)
5         .setContentText(textContent)
6         .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

  - A small icon, set by *setSmallIcon*(). This is the only user-visible content that's required.
  - A title, set by *setContentTitle*().
  - The body text, set by *setContentText*().
  - The notification priority, set by *setPriority*(). The priority determines how intrusive the notification should be on Android 7.1 and lower.

Set the notification's tap action

- Every notification should respond to a tap, usually to open an activity in your app that corresponds to the notification.
- You must specify a content intent defined with a **PendingIntent** object and pass it to *setContentIntent*().

Example

Code

```
1  // Create an explicit intent for an Activity in your app
2  Intent intent = new Intent(this, AlertDetails.class);
3  intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
4  PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
5
6
7  NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
8          .setSmallIcon(R.drawable.notification_icon)
9          .setContentTitle("My notification")
10         .setContentText("Hello World!")
11         .setPriority(NotificationCompat.PRIORITY_DEFAULT)
12         // Set the intent that will fire when the user taps the notification
13         .setContentIntent(pendingIntent)
14         .setAutoCancel(true);
```

Show the notification

- To make the notification appear, call **NotificationManagerCompat**.*notify*(), passing it a unique ID for the notification and the result of **NotificationCompat**.**Builder**.*build*().

Code

```
1 NotificationManagerCompat notificationManager =
2 NotificationManagerCompat.from(this);
3 notificationManager.notify(notificationId, builder.build());
```

Introduction to Firebase

Section - 5

Firebase

- **Firebase** is a *mobile platform* that helps you quickly develop high-quality apps, grow your user base, and earn more money.
- Firebase is made up of complementary features that you can mix-and-match to fit your needs, with Google Analytics for Firebase at the core.
- Benefits of using Firebase
  - Move fast
  - Forget about infrastructure
  - Make smart, data-driven decisions
  - Work across platforms
  - Scale with ease
  - Get free support

Firebase Products

- Cloud Firestore
- Machine Learning
- Cloud Functions
- Authentication
- Hosting
- Cloud Storage
- Realtime Database
- Crashlytics
- Performance Monitoring
- Test Lab
- App Distribution
- Google Analytics
- In-App Messaging
- Predictions
- A/B Testing
- Cloud Messaging
- Remote Config
- Dynamic Links

Configure Firebase

- Configure Firebase Platform
- Add your app to the Firebase Console
  - Add project by specifying the name of the Project.
  - Enable or disable the Google Analytics for your project and then click Continue.
  - Choose or create a Google Analytics account, and then click Create Project.
  - Add an app to your project by clicking Android icon on the screen shown.
    - Add android package name
    - App nickname (optinal)
    - SHA-1 (optional)
  - Download "*google-services.json*" configuration file and add it to your project app folder and then click Next.
  - Do few gradle changes as required by Firebase.
  - Run your application for verification and click Continue to Console.
- Installing the SDK

Firebase Application

- Store data with Cloud Firestore
- Sign in users with Firebase Auth
- Records events with Analytics
- Personalize with Remote Config
- Catch errors with Crashlytics
- and much more...

Store data with Cloud Firestore

- Check the authentication parameters in the Firebase Console.
- Add dependency for firebase-storage with the latest version.
- Initialize the Cloud Firestore

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
```

- Add data
  - Cloud Firestore stores data in documents, which are stored in collections.
  - Cloud Firestore creates collections and documents implicitly the first time you add data to the document.
  - It is not necessary that you create collections or documents explicitly.

Create Data

Code

```
// Create a new user with a first and last name
Map<String, Object> user = new HashMap<>();
user.put("first", "Darshan");
user.put("last", "University");
user.put("year", 2021);


// Add a new document with a generated ID
db.collection("users")
        .add(user)
        .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
            @Override
            public void onSuccess(DocumentReference documentReference) {
                Log.d(TAG, "DocumentSnapshot added with ID: " +
documentReference.getId());
            }
        })
        .addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.w(TAG, "Error adding document", e);
            }
        });
```

Read Data

Code

```
1  db.collection("users")
2   .get()
3   .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
4         @Override
5         public void onComplete(@NonNull Task<QuerySnapshot> task) {
6             if (task.isSuccessful()) {
7                 for (QueryDocumentSnapshot document : task.getResult()) {
8                     Log.d(TAG, document.getId() + " => " +
9  document.getData());
10                }
11            } else {
12                Log.w(TAG, "Error getting documents.",
13 task.getException());
14            }
15        }
16    });
```

Update Data

Code

```
1  // Create an initial document to update
2  DocumentReference frankDocRef = db.collection("users").document("frank");
3  Map<String, Object> initialData = new HashMap<>();
4  initialData.put("name", "Frank");
5  initialData.put("age", 12);
6
7
8  Map<String, Object> favorites = new HashMap<>();
9  favorites.put("food", "Pizza");
10 favorites.put("color", "Blue");
11 favorites.put("subject", "Recess");
12 initialData.put("favorites", favorites);
13
14
15 ApiFuture<WriteResult> initialResult = frankDocRef.set(initialData);
16 // Confirm that data has been successfully saved by blocking on the operation
17 initialResult.get();
18
19
20 // Update age and favorite color
21 Map<String, Object> updates = new HashMap<>();
22 updates.put("age", 13);
23 updates.put("favorites.color", "Red");
24
25
26 // Async update document
27 ApiFuture<WriteResult> writeResult = frankDocRef.update(updates);
28 // ...
29 System.out.println("Update time : " + writeResult.get().getUpdateTime());
```

Delete Data

Code

```
1  void deleteCollection(CollectionReference collection, int batchSize) {
2    try {
3  // retrieve a small batch of documents to avoid out-of-memory errors
4  ApiFuture<QuerySnapshot> future = collection.limit(batchSize).get();
5  int deleted = 0;
6  // future.get() blocks on document retrieval
7  List<QueryDocumentSnapshot> documents = future.get().getDocuments();
8  for (QueryDocumentSnapshot document : documents) {
9     document.getReference().delete();
10    ++deleted;
11 }
12 if (deleted >= batchSize) {
13    // retrieve and delete another batch
14    deleteCollection(collection, batchSize);
15 }
16   } catch (Exception e) {
17 System.err.println("Error deleting collection : " + e.getMessage());
18   }
19 }
```

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Thank You

Prof. Mehul D Bhundiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
mehul.bhundiya@darshan.ac.in
+91-9428231065