Unit – 3 Android Application Basics

A) Anatomy of an Android Application

An Android application consists of various components that work together to provide a rich and interactive user experience. The key components of an Android application include:

1. Activities:

- An activity represents a single screen with a user interface.
- It is a crucial building block of an Android application and typically corresponds to a user interface (UI) screen.
- Activities manage user interactions, handle user input, and coordinate with other components.

```
Ex: public class MyService extends Service {

    }
```

2. Services:

- Services are components that run in the background to perform long-running operations or handle tasks without a user interface.
- They are often used for tasks such as playing music, handling network operations, or updating data in the background.

```
Ex: public class MyService extends Service {

    }
```

3. Broadcast Receivers:

- Broadcast receivers respond to system-wide broadcast announcements or events.
- They are used to listen for and respond to system events, such as low battery, incoming SMS, or network connectivity changes.

```
Ex:  public class MyReceiver extends BroadcastReceiver {
     public void onReceive(context,intent){}


}
```

4. Content Providers:

- Content providers manage access to a structured set of data, often stored in a SQLite database.
- They allow applications to share data with each other and provide a standardized interface to access and manipulate data.

Ex:
```
public class MyContentProvider extends ContentProvider
{
   public void onCreate(){}
}
```

5. Intent:

- Intents are a messaging system that allows components to request functionality from other components within the application or even from external applications.
- They can be explicit, specifying the target component, or implicit, allowing the system to find an appropriate component to handle the request.

6. Fragments:

- Fragments represent a modular portion of a user interface or behavior within an activity.
- They are used to create flexible and responsive UI designs for both phones and tablets.

7. Layouts and Views:

- Layouts define the structure of the user interface by arranging views and other layout elements.
- Views are the UI components, such as buttons, text fields, and images.

8. Manifest File:

- The AndroidManifest.xml file provides essential information about the application to the Android system.

- It includes details such as the application's package name, version, permissions, and the components (activities, services, etc.) that the application consists of.

9. Resources:

- Resources are external elements used by the application, such as images, strings, and layouts.
- They are stored in the `res/` directory and are referenced in the code.

10. Gradle Build System:

- Android applications are typically built using the Gradle build system.
- The build.gradle files specify dependencies, build configurations, and other settings.

Understanding how these components interact and collaborate is essential for developing effective Android applications. The Android development framework provides a powerful and flexible structure for creating diverse and feature-rich mobile applications.

a) Mastering Important Android Terminology

• APK: Stands for Android Package. It is the file format used to distribute and install Android applications.

• ADB: Stands for Android Debug Bridge. It is a command-line tool used for communication with an Android device or emulator.

• API: Stands for Application Programming Interface. It defines a set of rules and protocols that allow software applications to interact with each other.

• SDK: Stands for Software Development Kit. It provides tools, libraries, and documentation for developing Android applications.

• Manifest file (AndroidManifest.xml): An XML file containing essential information about the application, including its package name, components, permissions, and version.

• API Level : A version of the Android API, indicating the features and capabilities available to developers.

• Emulator: A software-based virtual device that mimics the behavior of a physical Android device, allowing developers to test and debug applications.

b) Using the application context

• The Application Context represents the global context of an Android application.
• It provides access to application-specific resources and information.
• The Application Context can be accessed from any component within the application, such as activities, services, or broadcast receivers.
• It is useful for retrieving resources, accessing system services, or obtaining application-level information

- **List of Functionalities of Activity Context**

The Application context in Android provides access to various functionalities and resources throughout the entire lifecycle of an application. Here is a list of functionalities and use cases for the application context:

1) Access to Resources:
Retrieve application-specific resources such as strings, colors, and dimensions.

2) File and Asset Access:

Access files and assets bundled with the application.

3) Access to System Services:
Obtain system services like NotificationManager, AlarmManager, ConnectivityManager, etc.

4)Launching Activities:
Start activities using the startActivity() method.

5)Broadcasting Intents:
Broadcast and listen for intents using methods like sendBroadcast().

6)Accessing Shared Preferences:
Access application-wide preferences using getSharedPreferences().

7)Application Information:
Retrieve information about the application, such as its package name, version code, etc.

8)Accessing Application-specific Directories:
Obtain paths to directories specific to the application, like the cache directory.

9)Creating Toasts:
Display toast messages using the Toast.makeText() method.

10)Accessing Class Loaders:
Obtain the class loader for the application.

11)Creating Dialogs:
Create dialogs using AlertDialog.Builder or other dialog-related methods.

12)Theme and Style Information:
Retrieve information about the application's theme and style.

13)Global Context in Libraries:
When working with libraries that require a context, using the application context can help prevent memory leaks associated with Activity contexts.

14)Accessing the Main Thread Handler:
Obtain the Handler associated with the main thread using getMainLooper().

15)Managing Lifecycles:
Since the application context is not tied to the lifecycle of a specific component, it is useful for operations that need to persist throughout the entire application lifecycle.

C) Performing Application Tasks with Activities

- Activities are the building blocks of the user interface in an Android application.
- They handle the presentation of UI elements and manage user interactions.
- Activities can be launched individually or organized in a task stack to enable navigation between different screens or activities.

- **Declare activities:-**
  To declare your activity, open your manifest file and add an element as a child of the element.

  For example:

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

D) <u>Organizing Activity Components with Fragments:</u>

- A Fragment represents a reusable portion of app's UI.
- Fragment class in Android is used to build dynamic User Interface. A great advantage of fragment is that it simplifies the task of creating UI for multiple screen size.
- 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.

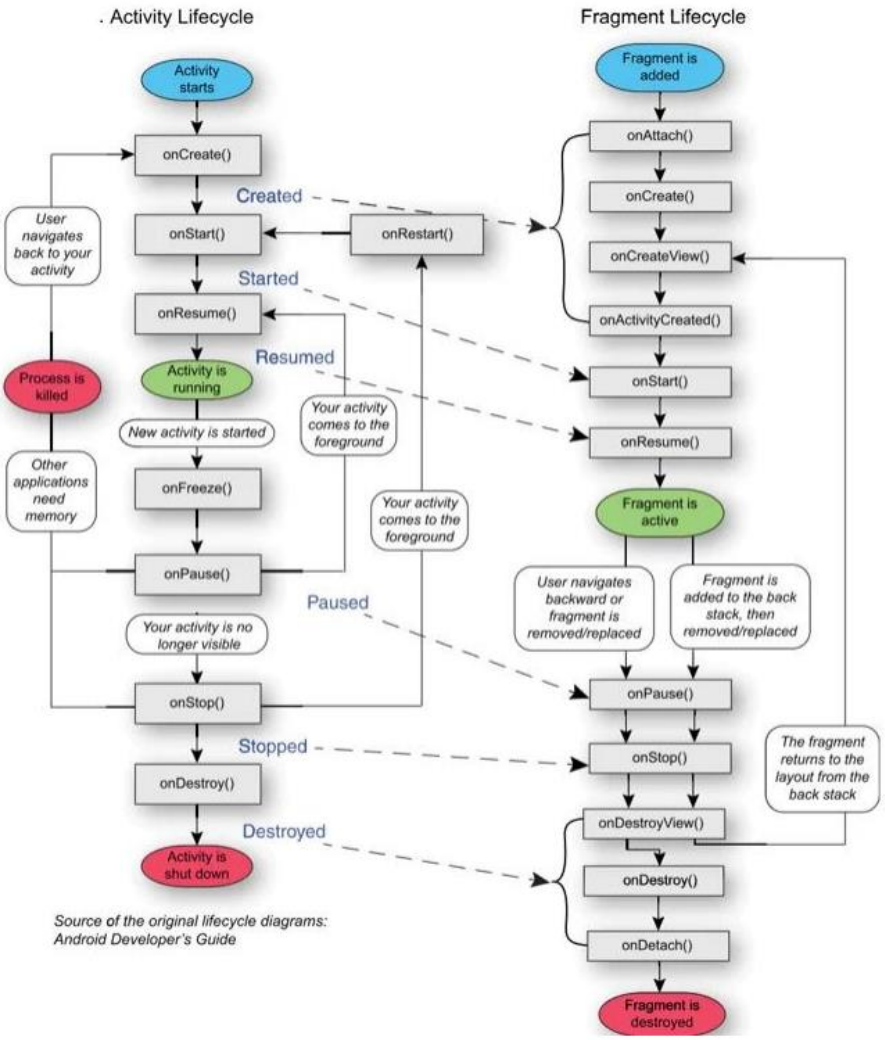
IMPORTANT OF FRAGMENTS:

<u>Modularity and Reusability:</u>
Fragments promote modularity by allowing developers to break down the UI and functionality into smaller, self-contained components. These components can be reused across multiple activities, improving code organization and maintainability.

Adaptability to Different Screen Sizes:
Fragments are particularly valuable for creating responsive user interfaces that can adapt to various screen sizes. On larger screens, such as tablets, multiple fragments can be displayed side by side, enhancing the user experience.

Code Organization and Maintainability:
By encapsulating specific UI components and behaviors in fragments, code becomes more organized and easier to maintain. This separation of concerns makes it simpler to understand, update, and extend the functionality of an application.

Android Fragment Lifecycle:



onAttach():

- The fragment instance is associated with an activity instance.
- The fragment and the activity is not fully initialized.
- Typically you get in this method a reference to the activity which uses the fragment for further initialization work.

onCreate():

- The system calls this method when creating the fragment.
- You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

onCreateView() :

- The system calls this callback when it's time for the fragment to draw its user interface for the first time.
- To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout.
- You can return null if the fragment does not provide a UI.

onActivityCreated():

- The onActivityCreated() is called after the onCreateView() method when the host activity is created.
- Activity and fragment instance have been created as well as the view hierarchy of the activity.

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.

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.

Android Fragment Example

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context="example.javatpoint.com.fragmentexample.MainActivity">

    <fragment
        android:id="@+id/fragment1"
        android:name="example.javatpoint.com.fragmentexample.Fragment1"
        android:layout_width="0px"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />

    <fragment
        android:id="@+id/fragment2"
        android:name="example.javatpoint.com.fragmentexample.Fragment2"
        android:layout_width="0px"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />

</LinearLayout>
```

File: fragment_fragment1.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F5F5DC"
    tools:context="example.javatpoint.com.fragmentexample.Fragment1">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />

</FrameLayout>
```

File: fragment_fragment2.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F0FFFF"
    tools:context="example.javatpoint.com.fragmentexample.Fragment2">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />

</FrameLayout>
```

MainActivity Class

File: MainActivity.java

```
package example.javatpoint.com.fragmentexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

File: fragment1.java

```
package example.javatpoint.com.fragmentexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {

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

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
      // Inflate the layout for this fragment
      return inflater.inflate(R.layout.fragment_fragment1, container, false);
   }
}
```

File: fragment2.java

```
package example.javatpoint.com.fragmentexample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment2 extends Fragment {

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

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_fragment2, container, false);
    }

}
```

Output: