UNIT IV: Intents, Fragments, Toast and Alert Dialogs

4.1 Application context

Android apps are popular for a long time and it is evolving to a greater level as user's expectations are that they need to view the data that they want in an easier smoother view

- It allows us to access resources.
- It allows us to interact with other Android components by sending messages.
- It gives you information about your app environment.

1. **It is the context of the current/active state of the application.**
2. **It is used to get information about the activity and application.**
3. **It is used to get access to resources, databases, and shared preferences, etc.**
4. **Both the Activity and Application classes extend the Context class.**

Types of Context in Android
There are mainly two types of context are available in Android.

1. **Application Context and**
2. **Activity Context**

Application Context

This context is tied to the life cycle of an application. Mainly it is an instance that is a singleton and can be accessed via **getApplicationContext()**. Some use cases of Application Context are:

- If it is necessary to create a singleton object
- During the necessity of a library in an activity

getApplicationContext():
It is used to return the context which is linked to the Application which holds all activities running inside it. When we call a method or a constructor, we often have to pass a context and often we use **"this"** to pass the activity context or **"getApplicationContext"** to pass the application context.

This method is generally used for the application level and can be used to refer to all the activities. For example, if we want to access a variable throughout the android app, one has to use it via **getApplicationContext()**.

Activity Context

It is the activity context meaning each and every screen got an activity. For example, EnquiryActivity refers to EnquiryActivity only and AddActivity refers to AddActivity only.

- The user is creating an object whose lifecycle is attached to an activity.
- Whenever inside an activity for UI related kind of operations like toast, dialogue, etc,

getContext():
It returns the Context which is linked to the Activity from which it is called. This is useful when we want to call the context from only the current running activity.

getBaseContext():
The base context is set by the constructor or **setBaseContext()**.This method is only valid if we have a **ContextWrapper**.

this:
"this" argument is of a type "Context". To explain this context let's take an example to show a Toast Message using "this".

4.2 Intents and its Types.

It is quite usual for users to witness a jump from one application to another as a part of the whole process

Intents are used for navigating among various activities within the same application

Android application components can connect to other Android applications. This connection is based on a task description represented by an Intent object.

Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications

Uses of Intent

1. Sending the User to Another App
2. Setting a Launcher Activity
3. Switch Between Current Activity to Next Activity

Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.

There are two types of intents in android:

1. Explicit Intent
2. Implicit Intent.

Explicit Intent

Explicit Intent specifies the component. In such a case, intent provides the external class to be invoked.

Explicit Intents are used to connect the application internally.

In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One

Activity to another activity using Intent. In the similar way we can start a service to download a file in background process.

Explicit Intent work internally within an application to perform navigation and data transfer.

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

Implicit Intent

In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application.

The basic example of implicit Intent is to open any web page

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

4.3 Notifications

A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area.

Create and Send Notifications

Step 1 - Create Notification Builder

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
```

Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following –

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling **setContentIntent()**.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
```

```
stackBuilder.addParentStack(ResultActivity.class);


// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

Step 4 - Issue the notification

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call **NotificationCompat.Builder.build()** method on builder object before notifying it. This method combines all of the options that have been set and return a new **Notification** object.

```
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

4.4 Introduction to Fragments.

A Fragment **is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of** sub-activity**.**

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.
- We can add, replace or remove Fragment's in an Activity while the activity is running. For performing these operations we need a Layout(Relative Layout, FrameLayout or any other layout) in xml file and then replace that layout with the required Fragment.

Need Of Fragments In Android:

Before the introduction of Fragment's we can only show a single Activity on the screen at one given point of time so we were not able to divide the screen and control different parts separately. With the help of Fragment's we can divide the screens in different parts and controls different parts separately.

By using Fragments we can comprise multiple Fragments in a single Activity. Fragments have their own events, layouts and complete life cycle. It provide flexibility and also removed the limitation of single Activity on the screen at a time.

Basic Fragment Code In XML:

```
<fragment
android:id="@+id/fragments"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

4.5 Creating and Adding Fragments.

Create A Fragment Class In Android Studio:

For creating a Fragment firstly we extend the Fragment class, then override key lifecycle methods to insert our app logic, similar to the way we would with an Activity class. While creating a Fragment we must use onCreateView() callback to define the layout and in order to run a Fragment.

Now add new fragment in existing project by right clicking on java/package -> New -> Select Fragment and name them as fragment_first .

1. FragmentActivity: The base class for all activities using compatibility based Fragment (and loader) features.

2. Fragment: The base class for all Fragment definitions

3. FragmentManager: The class for interacting with Fragment objects inside an activity

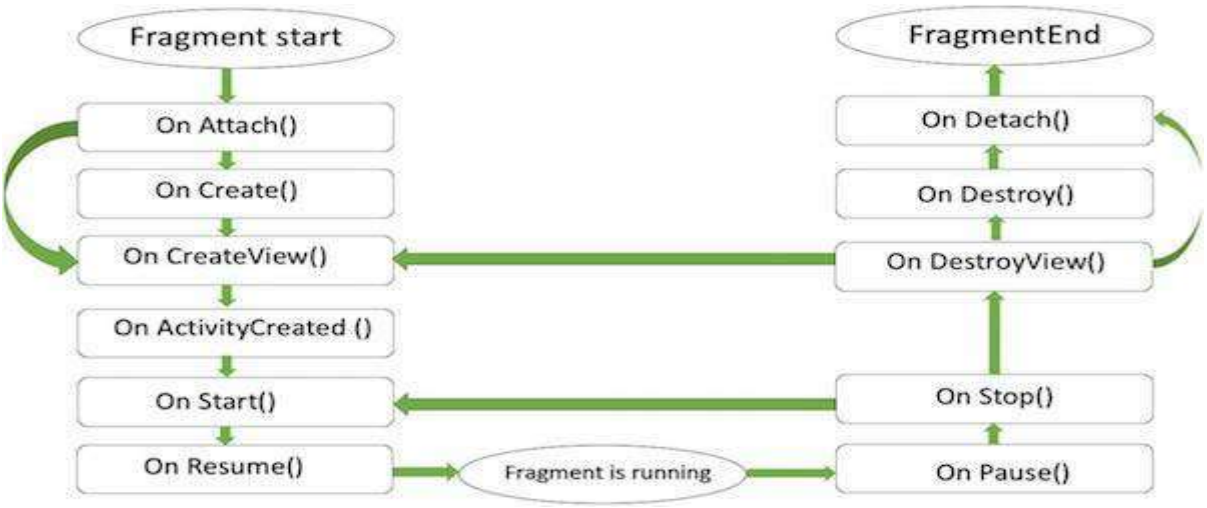
4. FragmentTransaction: The class for performing an atomic set of Fragment operations such as Replace or Add a Fragment.

Adding Fragment Class In Android Studio:

```
FirstFragment fragment=new First Fragment();
FragmentManager fm = getFragmentManager();
// create a FragmentTransaction to begin the transaction and replace the Fragm
ent
FragmentTransaction fragmentTransaction = fm.beginTransaction();
// replace the FrameLayout with new Fragment
fragmentTransaction.replace(R.id.frameLayout, fragment);
fragmentTransaction.commit(); // save the changes
```

5: Now add another new fragment in existing project by right clicking on java/package -> New -> Select Fragment and name them as fragment_first .

4.6 Lifecycle of Fragment



onAttach()
The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized.

onCreate()
The system calls this method when creating the fragment.

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.

onActivityCreated()
The onActivityCreated() is called after the onCreateView() method when the host activity is created.

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.

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.

4.7 Toast, Custom Toast

Toast is used to display information for a period of time. It contains a message to be displayed quickly and disappears after specified period of time. It does not block the user interaction. Toast is a subclass of Object class. In this we use two constants for setting the duration for the Toast. Toast notification in android always appears near the bottom of the screen. We can also create our custom toast by using custom layout (xml file).

Important Methods Of Toast:

1. makeText(Context context, CharSequence text, int duration): This method is used to initiate the Toast. This method take three parameters First is for the application Context, Second is text message and last one is duration for the Toast.

Constants of Toast: Below is the constants of Toast that are used for setting the duration for the Toast.

1. LENGTH_LONG: It is used to display the Toast for a long period of time. When we set this duration the Toast will be displayed for a long duration.

2. LENGTH_SHORT: It is used to display the Toast for short period of time. When we set this duration the Toast will be displayed for short duration.

2. show(): This method is used to display the Toast on the screen. This method is display the text which we create using makeText() method of Toast.

3. setGravity(int,int,int): This method is used to set the gravity for the Toast. This method accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

4. setText(CharSequence s): This method is used to set the text for the Toast. If we use makeText() method and then we want to change the text value for the Toast then we use this method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG);

// initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);    // set gravity for the Toast.

toast.setText("Changed Toast Text"); // set the text for the Toast

toast.show(); // display the Toast


 LayoutInflater li = getLayoutInflater();

    //Getting the View object as defined in the customtoast.xml file

    View layout = li.inflate(R.layout.customtoast,(ViewGroup) findViewById(R.id.custom_toast_layout));



    //Creating the Toast object

    Toast toast = new Toast(getApplicationContext());

    toast.setDuration(Toast.LENGTH_SHORT);

    toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);

    toast.setView(layout);//setting the view of custom toast layout

    toast.show();
```

4.8 Alert Dialog, Creating Custom Alert Dialog

Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

Android AlertDialog is composed of three regions: title, content area and action buttons.

Android AlertDialog is the subclass of Dialog class.

setTitle(CharSequence)

This method is used to set the title of AlertDialog.

setMessage(CharSequence)

This method is used to set the message for AlertDialog.

setIcon(int)

This method is used to set the icon over AlertDialog.

setCancelable(boolean cancel able)

This method sets the property that the dialog can be cancelled or not

setPositiveButton(CharSequence text, DialogInterface.OnClickListener listener) – This component add positive button

setNegativeButton(CharSequence text, DialogInterface.OnClickListener listener) – This component add negative button

setNeutralButton(CharSequence text, DialogInterface.OnClickListener listener) – This component simply add a new button and on this button developer can set any other onclick functionality like cancel button on alert dialog.

Alert Dialog Example

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);

    alertDialogBuilder.setMessage("Are you sure,You wanted to make decision");
```

```
    alertDialogBuilder.setPositiveButton("yes",new
DialogInterface.OnClickListener() {

      @Override

      public void onClick(DialogInterface arg0, int arg1) {

      }

    });

    alertDialogBuilder.setNegativeButton("No",new
DialogInterface.OnClickListener() {

      Override

      public void onClick(DialogInterface dialog, int which) {

        finish();

      }

    });

.setNeutralButton("Cancel", new DialogInterface.OnClickListener() {

                    @Override

                    public void onClick(DialogInterface dialog, int which) {

                    }

          });

    AlertDialog alertDialog = alertDialogBuilder.create();

    alertDialog.show();

  }
```


```
final Dialog dialog = new Dialog(context);
dialog.setContentView(R.layout.custom);
Button dialogButton = (Button) dialog.findViewById(R.id.dialogButtonOK);

// if button is clicked, close the custom dialog
dialogButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              dialog.dismiss();

Toast.makeText(getApplicationContext(),"Dismissed..!!",Toast.LENGTH_SHORT)
.show();
 }
});
dialog.show();
```

Thank You