Chapter – 5

Working with Fragments, Dialogs & Android Preferences

❖ Fragments

Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screens inside one activity.

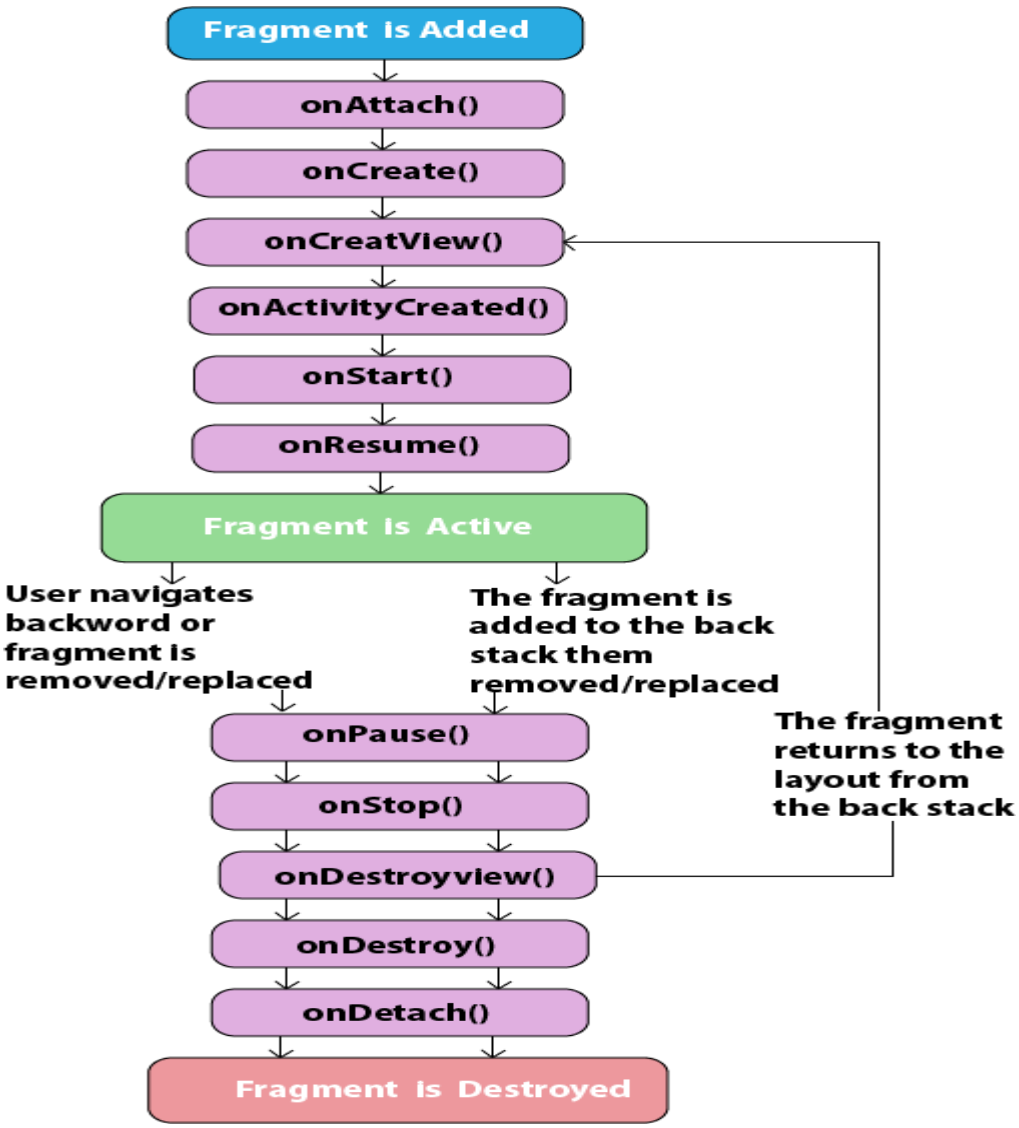
A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments can't live on their own. They must be *hosted* by an activity or another fragment. The fragment's view hierarchy becomes part of, or *attaches to*, the host's view hierarchy.

Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.

Each fragment has its own life cycle methods that are affected by activity life cycle because fragments are embedded in activity.

The **FragmentManager** class is responsible to make interaction between fragment objects.

Fragment Life Cycle : The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment.



Fragment Life Cycle Methods

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

Android Fragment When to call Method

Consider Fragment-1 is A and Fragment-2 is B and A is attached to the Main Activity

1. If you can replace B with A.

   A's call back:

   onDestroyView()
   onDestroy()
   onDetach()

   B's call back:

   onAttach()
   onCreate()
   onCreateView()
   onViewCreated()

2. If you can replace B with A without Losing resources.

   A's call back:

   onDestroy()
   onDetach()

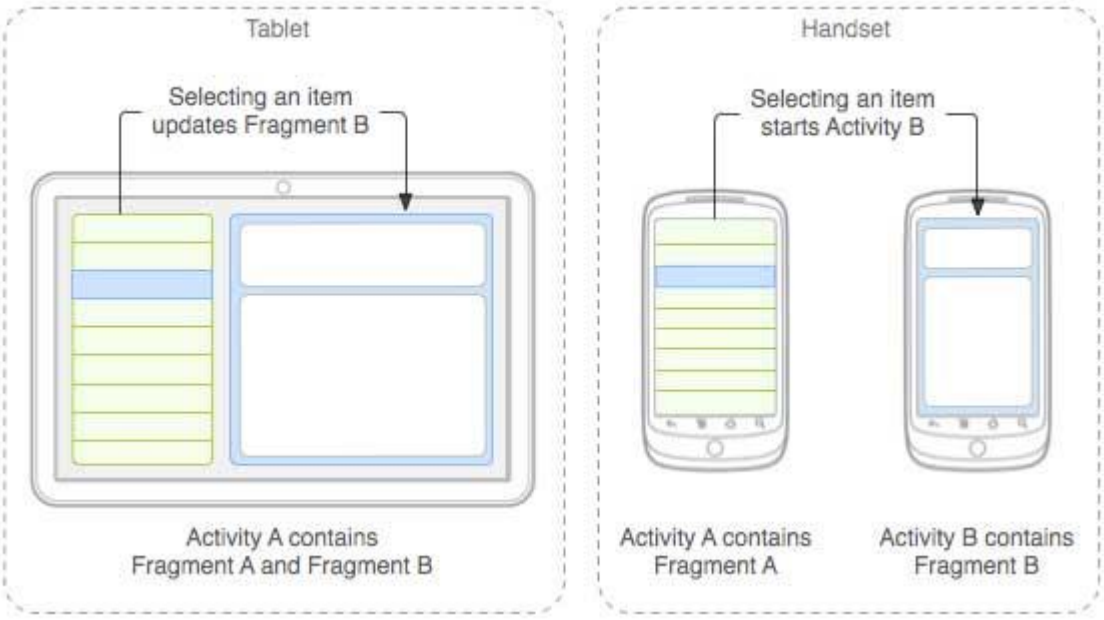
   B's call back:

   onAttach()
   onCreate()
   onCreateView()
   onViewCreated()

- **Understanding Fragments**

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

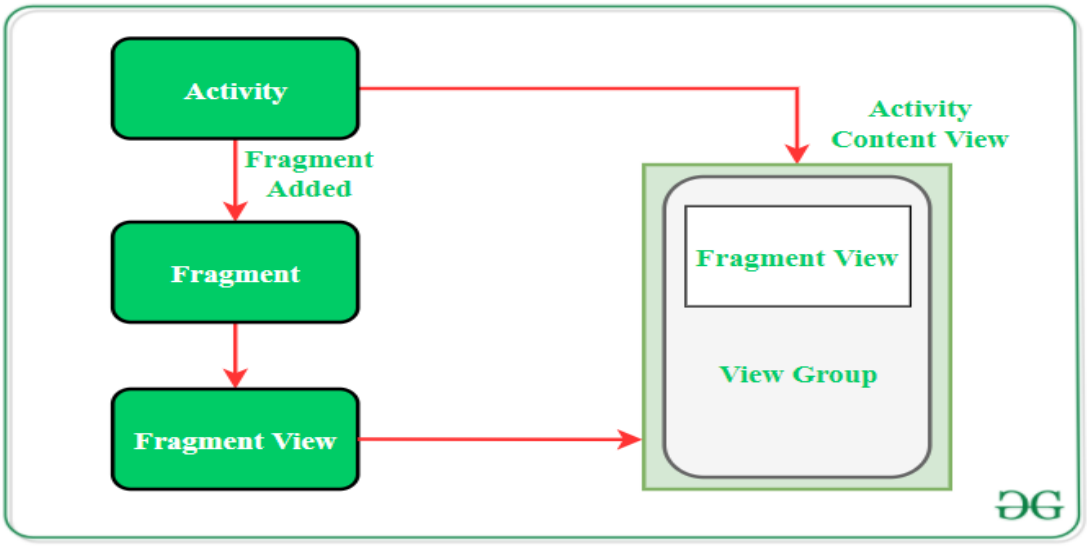


The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of

articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

You create fragments by extending **Fragment** class and you can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element. By dividing the activity's layout multiple fragments can be added in it.

Below is the pictorial representation of fragment interaction with the activity:



Types of Android Fragments :

1. **Single Fragment :** Display only one single view on the device screen. This type of fragment is mostly used for mobile phones.

2. **List Fragment :** This Fragment is used to display a list-view from which the user can select the desired sub-activity. The menu drawer of apps like Gmail is the best example of this kind of fragment.

3. **Fragment Transaction :** This kind of fragments supports the transition from one fragment to another at run time. Users can switch between multiple fragments like switching tabs.

Importance of Fragment :

There are many use cases for fragments but the most common use cases include:

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

- **Using Android Support Package for Fragments**

The Android Support Library provides backward compatiblity for fragments and other features.

To use fragment, you need to include the support library in your project.

You can add the support library to your project by adding the necessary dependencies in your build.gradle file.

The support library provide classes like Fragment, FragmentManager, and FragmentTransaction to work with fragment.

Fragments require a dependency on the AndroidX Fragment library. You need to add the Google Maven repository to your project's settings.gradle file in order to include this dependency.

```
dependencyResolutionManagement {
      repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
      repositories {
      google()
       ...
     }
  }
```

To include the AndroidX Fragment library to your project, add the following dependencies in your app's build.gradle file:

```
dependencies {
def fragment_version = "1.6.2"

 // Java language implementation
 implementation "androidx.fragment:fragment:$fragment_version"
 // Kotlin
 implementation "androidx.fragment:fragment-ktx:$fragment_version"

 }
```

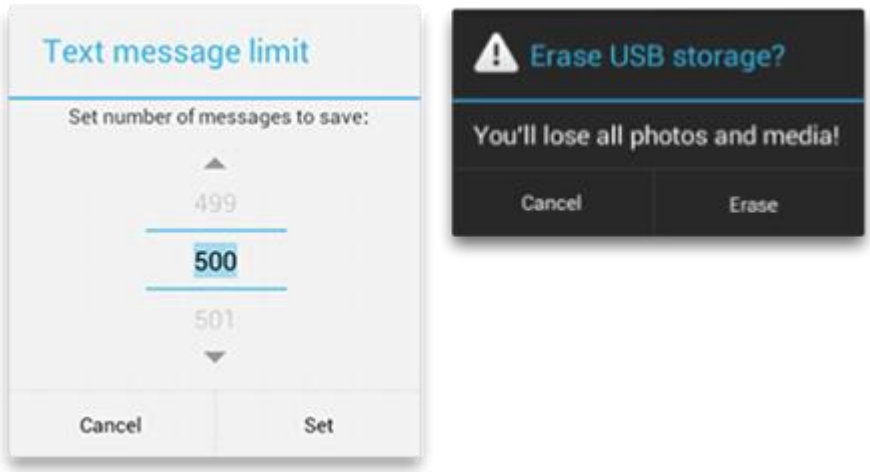
To create a fragment, extend the AndroidX Fragment class, and override its methods to insert your app logic, similar to the way you would create an Activity class. To create a minimal fragment that defines its own layout, provide your fragment's layout resource to the base constructor, as shown in the following example:

```
class ExampleFragment extends Fragment {
   public ExampleFragment() {
      super(R.layout.example_fragment);
   }
}
```

- **Dialogs**

Dialogs is a small window that prompts the user to make a decision or enter additional information. A Dialog does not fill the screen and is normally used for events that require users to take an action before they can proceed.

Dialogs are a common UI component used to display information, receive user input, or perform actions.



In Android, you can create following types of dialogs :

1. Alert Dialog
2. Simple Dialog
3. Character Picker Dialog

4. Date Picker Dialog
5. Time Picker Dialog
6. Progress Dialog

- **Exploring Different Types of Dialogs**

○ Alert Dialog

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

Alert Dialog code has three methods :

1. **setTitle()** method for displaying the alert dialog box title
2. **setMessage()** method for displaying the message
3. **setIcon()** method is used to set the icon on the Alert dialog box.

The following example demonstrates the use of AlertDialog in android.

Here is the code of res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  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:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
```

```
 android:paddingTop="@dimen/activity_vertical_margin"
 android:paddingBottom="@dimen/activity_vertical_margin"
 tools:context=".MainActivity">

 <TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Alert Dialog"
   android:id="@+id/textView"
   android:textSize="35dp"
   android:layout_alignParentTop="true"
   android:layout_centerHorizontal="true" />
 <Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Alert dialog"
   android:id="@+id/button"
   android:layout_below="@+id/imageView"
   android:layout_alignRight="@+id/textView2"
   android:layout_alignEnd="@+id/textView2"
   android:layout_marginTop="42dp"
   android:onClick="open"
   android:layout_alignLeft="@+id/imageView"
   android:layout_alignStart="@+id/imageView" />

</RelativeLayout>
```

Here is the code of src/MainActivity.java

```
package com.example.sairamkrishna.myapplication;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

```
   public void open(View view){
      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) {
            Toast.makeText(MainActivity.this,"You clicked yes
               button",Toast.LENGTH_LONG).show();
         }
      });

      alertDialogBuilder.setNegativeButton("No",new
DialogInterface.OnClickListener() {
         Override
         public void onClick(DialogInterface dialog, int
which) {
            finish();
         }
      });

      AlertDialog alertDialog = alertDialogBuilder.create();
      alertDialog.show();
   }
}
```

○ Character Picker Dialog

Android CharacterPickerDialog is a dialog box that permits the user to select "accented characters" of a base character. Oftentimes, the **CharacterPickerDialog** is useful because not all of the phones have the **Keyboard Layout** that is appropriate for a particular language.

Here is the code of activity_main.xml file::

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

   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="142dp"
      android:text="Button" />

   <EditText
      android:id="@+id/editText1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
```

```
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="55dp"
        android:ems="10" >

        <requestFocus />
    </EditText>

</RelativeLayout>
```

Here is the code of MainActivity.java file::

```
package com.bipinrupadiya.charachterpickerdemo;

import android.app.Activity;
import android.os.Bundle;
import android.text.method.CharacterPickerDialog;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

 private CharacterPickerDialog CPD = null;
 String chrOptions = "0123456789";

 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  final EditText t = (EditText) findViewById(R.id.editText1);
  Button button = (Button) this.findViewById(R.id.button1);
  button.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
    CPD.setTitle("Select Charecter");
    CPD.show();

   }
  });

  CPD = new CharacterPickerDialog(this, new View(this), null, chrOptions,
    true) {
   public void onClick(View v) {

    String c = "";
    c = ((Button) v).getText().toString();
    Toast.makeText(getApplicationContext(), "" + c,
      Toast.LENGTH_SHORT).show();

    t.setText(t.getText().toString() + c);
    dismiss();
   }
```

```
  };
 }
}
```

○ Date Picker Dialog

In Android, a Datepicker dialog is used to allow users to select a date from a graphical calendar interface. It's commonly used in forms or other situations where users need to input a date.

A Datepicker is like a little calendar that pops up when you need to pick a date in an app or a website. Instead of typing in the date manually, you can just click or tap on the Datepicker, and it shows you a calendar where you can select the date you want.

Let's see the simple example of datepicker widget in android.

activity_main.xml File ::

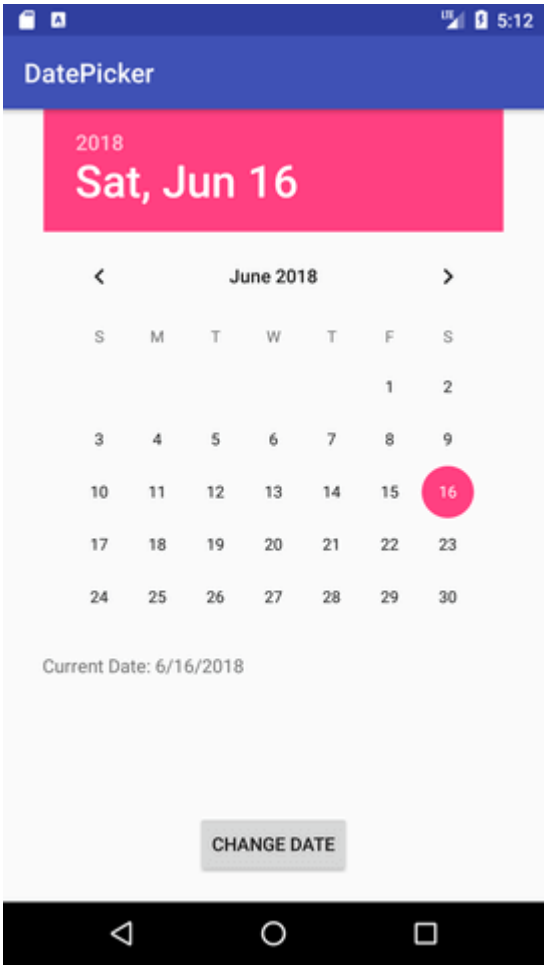
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="example.javatpoint.com.datepicker.MainActivity">

<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_above="@+id/button1"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_marginBottom="102dp"
android:layout_marginLeft="30dp"
android:layout_marginStart="30dp"
```

```
android:text="" />

<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:layout_centerHorizontal="true"
android:layout_marginBottom="20dp"
android:text="Change Date" />

<DatePicker
android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_above="@+id/textView1"
android:layout_centerHorizontal="true"
android:layout_marginBottom="36dp" />

</RelativeLayout>
```

MainActivity.java file::

```
package example.javatpoint.com.datepicker;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    DatePicker picker;
    Button displayDate;
```

```
    TextView textview1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textview1=(TextView)findViewById(R.id.textView1);
        picker=(DatePicker)findViewById(R.id.datePicker);
        displayDate=(Button)findViewById(R.id.button1);

        textview1.setText("Current Date: "+getCurrentDate());

        displayDate.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view) {

                textview1.setText("Change Date: "+getCurrentDate());
            }

        });

    }
    public String getCurrentDate(){
        StringBuilder builder=new StringBuilder();;
        builder.append((picker.getMonth() + 1)+"/");//month is 0 based
       builder.append(picker.getDayOfMonth()+"/");
        builder.append(picker.getYear());
        return builder.toString();
    }
}
```



○ TimePicker Dialog

Android TimePicker **widget is used to select date. It allows you to select time by hour and minute. You cannot select time by seconds.**

Let's see a simple example of android time picker.

activity_main.xml File

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

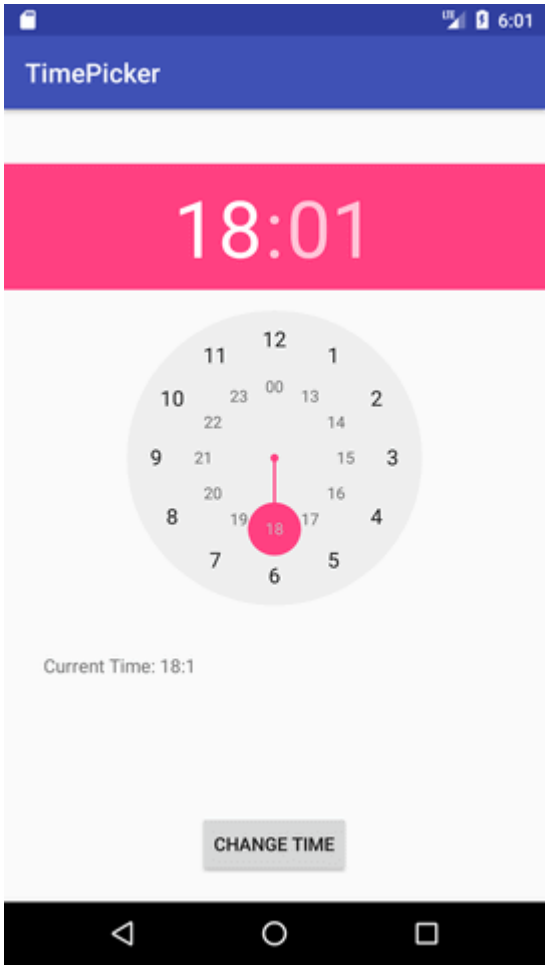
```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.timepicker.MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/button1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="102dp"
        android:layout_marginLeft="30dp"
        android:layout_marginStart="30dp"
        android:text="" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="20dp"
        android:text="Change Time" />

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
       android:layout_above="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="36dp" />
</RelativeLayout>
```

MainActivity.java

```
package example.javatpoint.com.timepicker;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends AppCompatActivity {
    TextView textview1;
    TimePicker timepicker;
    Button changetime;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textview1=(TextView)findViewById(R.id.textView1);
        timepicker=(TimePicker)findViewById(R.id.timePicker);
        //Uncomment the below line of code for 24 hour view
        timepicker.setIs24HourView(true);
        changetime=(Button)findViewById(R.id.button1);

        textview1.setText(getCurrentTime());

        changetime.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view) {
                textview1.setText(getCurrentTime());
            }
        });

    }

    public String getCurrentTime(){
        String currentTime="Current Time: "+timepicker.getCurrentHour()+":"+timepick
er.getCurrentMinute();
```

```
        return currentTime;
    }

}
```



- **ProgressBar Dialog**

We can display the **android progress bar** dialog box to display the status of work being done e.g downloading file, analyzing status of work etc.

Here is the example of progress bar dialog..

Activity_main.xml file

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity" >

   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="116dp"
      android:text="download file" />

</RelativeLayout>
```

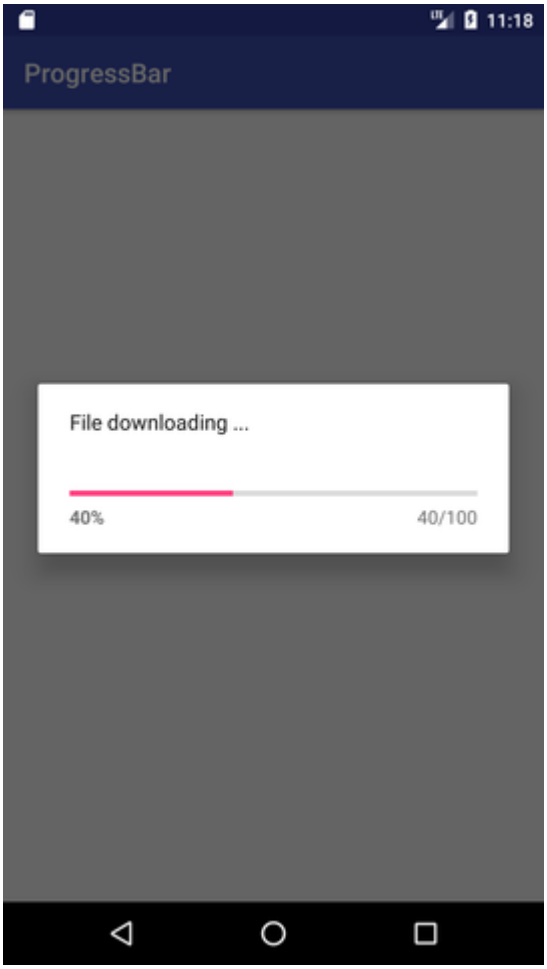
MainActivity.java file

```
   package example.javatpoint.com.progressbar;

   import android.app.ProgressDialog;
   import android.os.Handler;
   import android.support.v7.app.AppCompatActivity;
   import android.os.Bundle;
   import android.view.View;
   import android.widget.Button;

   public class MainActivity extends AppCompatActivity {
   Button btnStartProgress;
   ProgressDialog progressBar;
   private int progressBarStatus = 0;
   private Handler progressBarHandler = new Handler();
   private long fileSize = 0;
   @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
addListenerOnButtonClick();
}
public void addListenerOnButtonClick() {
btnStartProgress = findViewById(R.id.button);
btnStartProgress.setOnClickListener(new View.OnClickListener(){

@Override
public void onClick(View v) {
// creating progress bar dialog
progressBar = new ProgressDialog(v.getContext());
progressBar.setCancelable(true);
progressBar.setMessage("File downloading ...");
progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressBar.setProgress(0);
progressBar.setMax(100);
progressBar.show();
//reset progress bar and filesize status
progressBarStatus = 0;
fileSize = 0;

new Thread(new Runnable() {
public void run() {
while (progressBarStatus < 100) {
// performing operation
progressBarStatus = doOperation();
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
// Updating the progress bar
progressBarHandler.post(new Runnable() {
public void run() {
progressBar.setProgress(progressBarStatus);
```

```
}
});
}
// performing operation if file is downloaded,
if (progressBarStatus >= 100) {
// sleeping for 1 second after operation completed
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
// close the progress bar dialog
progressBar.dismiss();
}
}
}).start();
}//end of onClick method
});
}
// checking how much file is downloaded and updating the filesize
public int doOperation() {
//The range of ProgressDialog starts from 0 to 10000
while (fileSize <= 10000) {
fileSize++;
if (fileSize == 1000) {
return 10;
} else if (fileSize == 2000) {
return 20;
} else if (fileSize == 3000) {
return 30;
} else if (fileSize == 4000) {
return 40; // you can add more else if
}
/* else {
return 100;
}*/
}//end of while
```

```
return 100;
}//end of doOperation
}
```



- ## Working with Android Preferences

Android shared preference is used to store and retrieve primitive information. In android, string, integer, long, number etc. are considered as primitive data type.

Android Shared preferences are used to store data in key and value pair so that we can retrieve the value on the basis of key.

Shared Preferences is a mechanism for storing small amounts of data in key-value pairs. It is often used for saving application settings, user preferences, or other simple pieces of data that need to persist between application sessions.

Here's a brief explanation of how Shared Preferences work in Android with an example:

1. **Initialization**: First, you need to initialize the Shared Preferences object. You typically do this in your Activity or Application class.

```
SharedPreferences sharedPreferences =
getSharedPreferences("MyPrefs", Context.MODE_PRIVATE);
```

2. **Editing**: To edit the preferences, you use the SharedPreferences.Editor class.

```
SharedPreferences.Editor editor = sharedPreferences.edit();
```

3. **Adding Data**: Use the editor to put key-value pairs into the Shared Preferences.

```
editor.putString("username", "john_doe");

editor.putInt("age", 25);
```

4. **Committing Changes**: Once you have added the data you want to store, you need to commit the changes.

```
editor.apply(); // or editor.commit();
```

5. **Retrieving Data**: To retrieve data from Shared Preferences, you simply access them using the same key you used to store them.

```
String username = sharedPreferences.getString("username", "");

int age = sharedPreferences.getInt("age", 0);
```

In this Example

- We first initialize the Shared Preferences object with the name "MyPrefs" and the mode `MODE_PRIVATE`, which means only our application can access these preferences.
- **We then edit the preferences using the** `SharedPreferences.Editor`**.**
- **We add data using** `putString()` **and** `putInt()` **methods.**
- **Finally, we commit the changes using** `apply()` **or** `commit()` **methods.**
- To retrieve the data later, we use the same keys to get the values.

> It's important to note that Shared Preferences are meant for storing small amounts of data. If you need to store larger or more complex data, you might want to consider other options like a SQLite database or using files.