UNIT-4

Using Selection widgets and Debugging:

Using List view:

A ListView is used to display a list of vertically scrolling items, allowing users to select one or more of them.

List of Attributes Used to Configure the ListView Control:

Attribute	Description
`android:entries`	Used to refer to an `array resource` for displaying options in the `ListView`

`android:choiceMode`	Used to define the number of items that are selectable from the `ListView`. Valid values are
	`none`—Doesn't allow selection of any option from the `ListView`
	`singleChoice`—Allows selection of a single option from the `ListView`
	`multipleChoice`—Allows selection of more than one option from the `ListView`

`multipleChoiceModal`	Used to allow selection of more than one item in a custom selection mode
`android:drawSelectorOnTop`	When set to `true`, the `selector` (an orange bar) is drawn over the selected item. Otherwise, the `selector` is drawn behind the selected item. The default value is `false`.
`android:transcriptMode`	Used to set the `transcript mode` for the list. The `transcript mode` helps in deciding whether we want the list to automatically scroll to the bottom. The valid values are `disabled`—Disables the `transcript mode` (default value). The `ListView` does not scroll automatically. `normal`—The `ListView` automatically scrolls to the bottom when a new item is added to the adapter and a notification is generated. `alwaysScroll`—The `ListView` always automatically scrolls to the bottom.

A sample ListView control may appear as follows:

<ListView

```
android:id="@android:id/list"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:entries="@array/fruits"
android:choiceMode="singleChoice"
android:drawSelectorOnTop="false"
android:transcriptMode="normal" />
```

Creating a ListView with an Activity Base Class

When we are creating a ListView by extending the Activity class or the ListActivity class, the ListView can be populated by one of the following two methods:

- By ListView through a string resource
- By ListView through Adapter

Populating ListView Through String Resource:

The strings.xml File After Adding the Strings Array

```
<resources>
<string name="app_name">ListViewApp</string>
<string name="menu_settings">Settings</string>
<string
name="title_activity_list_view_app">ListViewAppActivity</string>
<string-array name="fruits">
<item>Apple</item>
<item>Mango</item>
<item>Orange</item>
<item>Grapes</item>
<item>Banana</item>
</string-array>
</resources>
```

The Layout File activity_list_view_app.xml After Defining the ListView and TextView Controls

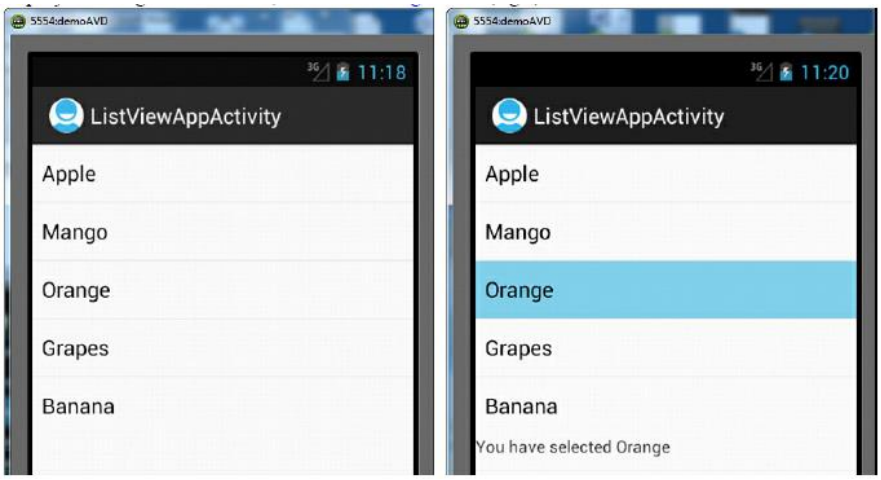
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="wrap_content">
<ListView android:id="@+id/fruits_list"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:entries="@array/fruits"
android:drawSelectorOnTop="false"/>
<TextView
android:id="@+id/selectedopt"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
```

</LinearLayout>

- To populate the ListView, the string-array fruits is assigned to the ListView through the android:entries attribute. That is, the android:entries attribute in the layout XML file is used for populating ListView from the string resource.
- The android:drawSelectorOnTop attribute is set to false, because we don't want the selector to be drawn over the selected item.
- To display the option selected from the ListView in the TextView control, we need to access the string-array TextView and attach an event listener to the ListView .

Code Written into the Java Activity File ListViewAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.ListView;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.view.View;
public class ListViewAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_list_view_app);
final String[] fruitsArray = getResources().getStringArray(R.array.fruits);
final TextView selectedOpt= (TextView)findViewById(R.id.selectedopt);
ListView fruitsList = (ListView)findViewById(R.id.fruits_list);
fruitsList.setOnItemClickListener(new
OnItemClickListener(){
@Override
public void onItemClick(AdapterView<?> parent, View v, int position,
long id)
{
selectedOpt.setText("You have selected "+fruitsArray[position]);
}
});
}
}
```



- To take an action when an item is selected from the ListView, the setOnItemClickListener() method is passed a new AdapterView.OnItemClickListener. the onItemClick() method, is executed when an item is selected from the ListView.

Using the Spinner control:

1. The Spinner control is like a drop-down menu. Allows you to select something from a short list of choices. The user only sees 1 value at rest.
2. Defining array in:
   1. Java
   String[]
   days={“Sunday”,”Monday”,”Tuesday”,”Wednesday”,Thursday”,”Friday”,”Saturday”};
   2. XML
   <String-array name=”days”>
   <item>Sunday</item>
   </String-array>
3. onItemSelectedListener is used to determine what happens when the user clicks on an item inside the Spinner.
4. It Overrides 2 methods, onItemSlected and onNothingSelected.
5. onItemSelected displays the TextView control.
6. Spinner manages its data through 2 methods. They are through String resources and ArrayAdapter.

Populating a Spinner Through string resources:

We define two resources, one to display a prompt in the Spinner control and the other to display a list of choices. To display a prompt in the Spinner control, we define a string resource. Open the strings.xml file from the res/values folder and define a string resource in it.

String Resource Defined in the strings.xml File

```
<resources>
<string name="app_name">SpinnerApp</string>
<string name="menu_settings">Settings</string>
<string name="title_activity_spinner_app">SpinnerAppActivity</string>
<string name="choose_msg">Choose a fruit</string>
</resources>
```

we need to define the resource for displaying options in the Spinner control. We use a string-array to do this. To add a new xml file to the res/values folder, right-click on the res/values folder in the Package Explorer window and select the New, Android XML File option. Call the file arrays (without the extension .xml) and then click the Finish button.

String Array Defined in the arrays.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="fruits">
<item>Apple</item>
<item>Mango</item>
<item>Orange</item>
<item>Grapes</item>
<item>Banana</item>
</string-array>
</resources>
```

To display the Spinner control in the application, let's define it in the layout file activity_spinner_app.xml

The activity_spinner_app.xml File with a Defined Spinner Control

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Spinner
android:id="@+id/spinner"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:prompt="@string/choose_msg"
android:entries="@array/fruits"/>
<TextView
android:id="@+id/selectedopt"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```

The Code Written into the Java Activity File SpinnerAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Spinner;
import android.widget.AdapterView;
import android.view.View;
import android.widget.AdapterView.OnItemSelectedListener;
public class SpinnerAppActivity extends Activity {
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_spinner_app);
final TextView selectedOpt= (TextView)findViewById(R.id.selectedopt);
Spinner spin=(Spinner)findViewById(R.id.spinner);
final String[] fruitsArray = getResources().getStringArray(R.array.fruits);
spin.setOnItemSelectedListener(new OnItemSelectedListener() {
public void onItemSelected(AdapterView<?> parent, View v, int position, long id) {
selectedOpt.setText("You have selected " +fruitsArray[position]);
}
public void onNothingSelected(AdapterView<?> parent)
{
selectedOpt.setText("");
}
});
}
}
```

Populating a Spinner through Array Adapter:

Code Written into SpinnerAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Spinner;
import android.widget.ArrayAdapter;
import android.widget.AdapterView;
import android.view.View;
import android.widget.AdapterView.OnItemSelectedListener;
public class SpinnerAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_spinner_app);
final TextView selectedOpt= (TextView)findViewById(R.id.selectedopt);
final String[] fruits={"Apple", "Mango", "Orange", "Grapes", "Banana"};
Spinner spin=(Spinner)findViewById(R.id.spinner);
ArrayAdapter<String> arrayAdpt=new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, fruits);
spin.setAdapter(arrayAdpt);
spin.setOnItemSelectedListener(new OnItemSelectedListener() {
public void onItemSelected(AdapterView<?> parent, View v, int position,long id) {
selectedOpt.setText("You have selected " +fruits[position]);
}
public void onNothingSelected(AdapterView<?> parent)
{
selectedOpt.setText("");
}
```

});
}
}

Using GridView control:

1. The GridView control is a ViewGroup used to display text and image data in the form of a rectangular, scrollable grid.
2. GridView is mainly useful when we want show data in grid layout like displaying images or icons.
3. This layout can be used to build applications like image viewer, audio or video players in order to show elements in grid manner.
4. To display data in the grid, we first define a GridView control in the XML layout, and then bind the data that we want to be displayed to it using the ArrayAdapter.

Attributes:

- **android:id** : This is the ID which uniquely identifies the layout
- **android:columnWidth** : This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm
- **android:gravity** : Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
- **android:horizontalSpacing** : Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm
- **android:numColumns** : Defines how many columns to show. May be an integer value, such as “100” or auto_fit which means display as many columns as possible to fill the available space
- **android:stretchMode** : Defines how columns should stretch to fill the available empty space, if any. This must be either of the values :
    - **none** : Stretching is disabled
    - **spacingWidth** : The spacing between each column is stretched
    - **columnWidth** : Each column is stretched equally
    - **spacingWidthUniform** : The spacing between each column is uniformly stretched

  android:verticalSpacing : Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm

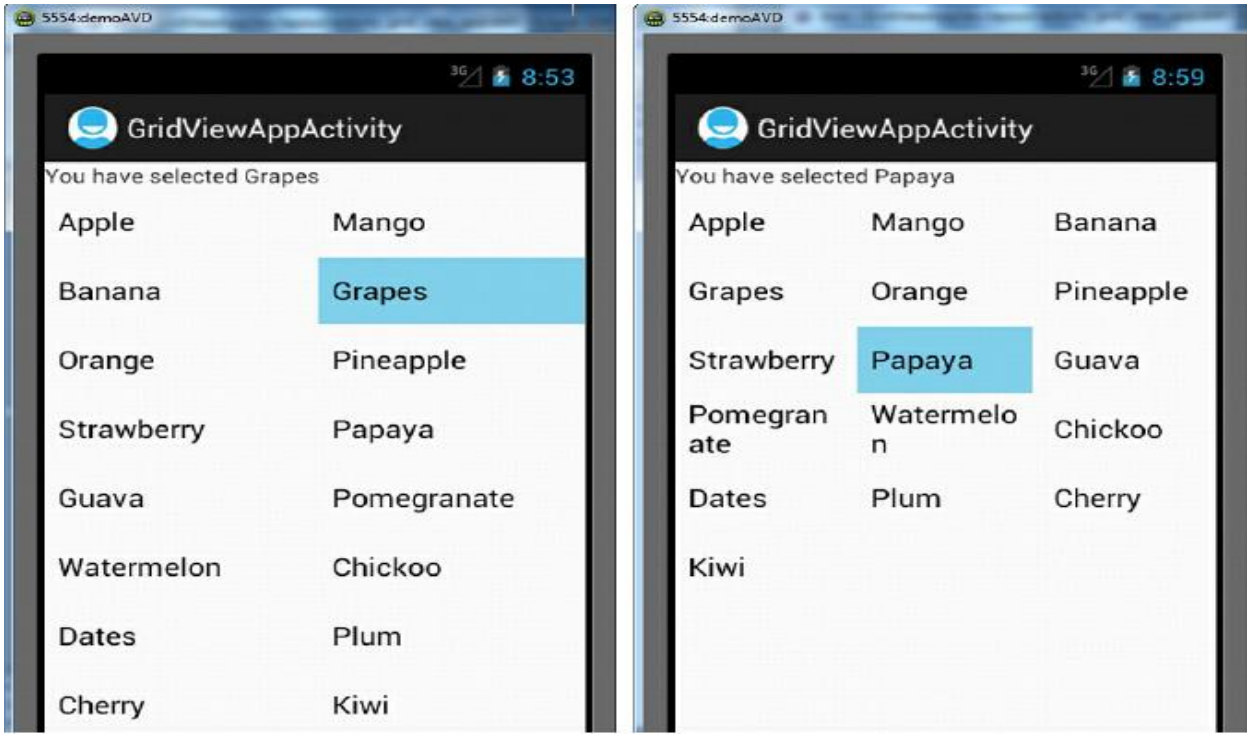
The Layout File activity_grid_view_app.xml After Defining theTextView and GridView Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView android:id="@+id/selectedopt"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Select a fruit " />
<GridView android:id="@+id/grid"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:verticalSpacing="2dip"
android:horizontalSpacing="5dip"
android:numColumns="auto_fit"
android:columnWidth="130dip"
android:stretchMode="columnWidth"
android:gravity="center" />
</LinearLayout>
```

Code Written into the Java Activity File GridViewAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.GridView;
import android.widget.ArrayAdapter;
import android.widget.AdapterView;
import android.view.View;
public class GridViewAppActivity extends Activity implements
AdapterView.OnItemClickListener {
TextView selectedOpt;
String[] fruits={"Apple", "Mango", "Banana", "Grapes", "Orange", "Pineapple", "Strawberry",
"Papaya", "Guava", "Pomegranate", "Watermelon", "Chickoo", "Dates", "Plum", "Cherry",
"Kiwi"};
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_grid_view_app);
selectedOpt=(TextView) findViewById(R.id.selectedopt);
GridView g=(GridView) findViewById(R.id.grid);
ArrayAdapter<String> arrayAdpt=new ArrayAdapter<String>
(this, android.R.layout.simple_list_item_1, fruits);
g.setAdapter(arrayAdpt);
g.setOnItemClickListener(this);
}
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
selectedOpt.setText("You have selected
"+fruits[position]);
}
public void onNothingSelected(AdapterView<?> parent) {
selectedOpt.setText("");
}
}
```



Displaying Images in GridView:

The GridView shows items in a two-dimensional scrolling grid. you can use the GridView together with an ImageView to show a series of pictures. the following example demonstrates how to Use GridView in Android application.

- create a new application called GridImageApp.
- Assuming the image filenames that we want to display through the GridView control are prod1.png, prod2.png, prod3.png, prod4.png, and prod5.png, copy them into the four res/drawable folders.
- TextView control for displaying the selected image number and a GridView control for displaying images in a grid.

The Layout File activity_grid_image_app.xml After Adding the TextView and GridView Controls

```
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:orientation="vertical" >
<TextView android:id="@+id/selectedopt"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="List of Products " />
<GridView android:id="@+id/grid"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:verticalSpacing="2dip"
android:horizontalSpacing="2dip"
```

```
android:numColumns="auto_fit"
android:columnWidth="100dip"
android:stretchMode="columnWidth"
android:gravity="center" />
</LinearLayout>
```

- To display images in the GridView control and also tell us which image is selected by the user, write the code shown below:

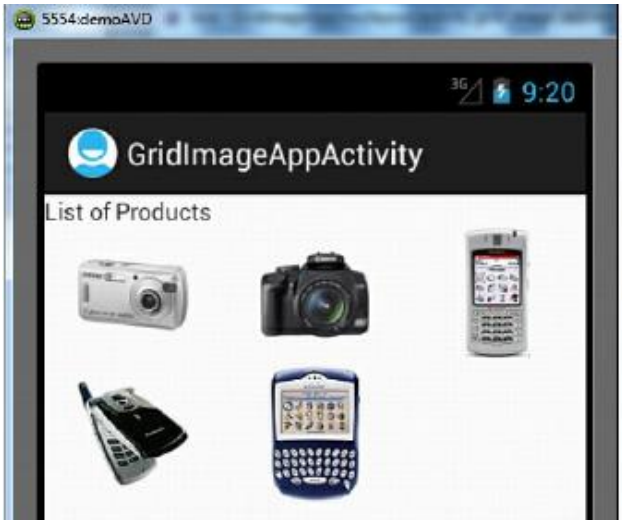
Code Written into the Java Activity File GridImageAppActivity.java

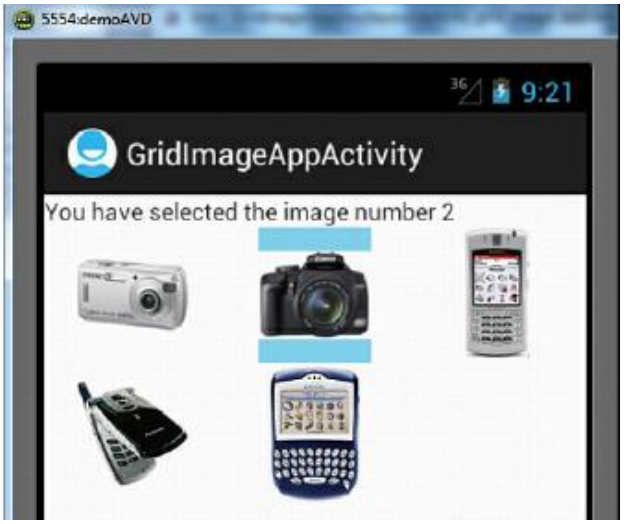
```
import android.app.Activity;
import android.os.Bundle;
import android.widget.GridView;
import android.view.View;
import android.widget.ImageView;
import android.content.Context;
import android.widget.BaseAdapter;
import android.widget.AdapterView;
import android.widget.TextView;
import android.view.ViewGroup;
public class GridImageAppActivity extends Activity implements
AdapterView.OnItemClickListener {
TextView selectedOpt;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_grid_image_app);
selectedOpt=(TextView) findViewById(R.id.selectedopt);
GridView g=(GridView) findViewById(R.id.grid);
g.setAdapter(new ImageAdapter(this));
g.setOnItemClickListener(this);
}
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
int p=position+1;
selectedOpt.setText("You have selected the image number "+p);
}
public class ImageAdapter extends BaseAdapter {
private Context contxt;
Integer[] images = {
R.drawable.prod1,
R.drawable.prod2,
R.drawable.prod3,
R.drawable.prod4,
R.drawable.prod5
};
public ImageAdapter(Context c) {
```

```
contxt = c;
}
public int getCount() {
return images.length;
}
public Object getItem(int position) {
return position;
}
public long getItemId(int position) {
return position;
}
public View getView(int position, View convertView, ViewGroup parent) {
ImageView imageView = new ImageView(contxt);
imageView.setImageResource(images[position]);
imageView.setLayoutParams(new GridView.LayoutParams(100, 120));
return imageView;
}
}
}
```

- To create our custom adapter, ImageAdapter, we extend the BaseAdapter abstract class that is provided by Android.
- The adapter's methods getCount(), getItem(), and getItemId() are used to determine the number of images to be displayed and the unique identifier of the specified image.
- The getView() method is used to retrieve the appropriate View, that is, the image at the specified position.
- In the getView() method, the member Context is used to create a new ImageView.
- An AdapterView.OnItemClickListener event handler is attached to the GridView, so that when any image displayed through GridView is clicked, the callback onItemClick() method is invoked.

On running the application, we see that all images are displayed in GridView, as shown in Figure (left). The image selected in the GridView is displayed through the TextView control at the top, as shown in Figure (right).





left right

Creating an Image Gallery Using the ViewPager Control:

- The ViewPager control (android.support.v4.view.ViewPager) helps in showing data, which may be text, image, and so on, in the form of pages with the horizontal swiping behavior. That is, the pages of data being displayed can be flipped left and right.
- The ViewPager needs a data adapter to define and load the data for each page.
- The data adapter that is used to define the data for each page to be displayed through the ViewPager control is the PagerAdapter (android.support.v4.view.PagerAdapter) class.

While implementing the PagerAdapter, we must override the following methods:

instantiateItem(View container, int position)—Used for creating and instantiating the page and adding it to the container.

Syntax:

public ObjectinstantiateItem(View container, int position)

container—Represents the container in which the page has to be displayed.

position—Represents the position of the page to be instantiated.

destroyItem(View container, int position, Object object)—Used for removing the page of the specified position from the container.

isViewFromObject(View view, Object object)—Determines whether the specified page is associated with a specific key object.

getCount()—Defines the size of the paging range, that is, the count of the number of the pages.

We can also set the initial position of the pager through the setCurrentItem() method.

SimpleOnPageChangeListener is used To listen to the change in state of the selected page, we need . onPageSelected() is called when a page from the ViewPager is selected.The instantiateItem(), getCount(), destroyItem(), and isViewFromObject() methods define the pages, determine the number of images

The Layout File activity_view_pager_app.xml After Defining the TextView and ViewPager Controls

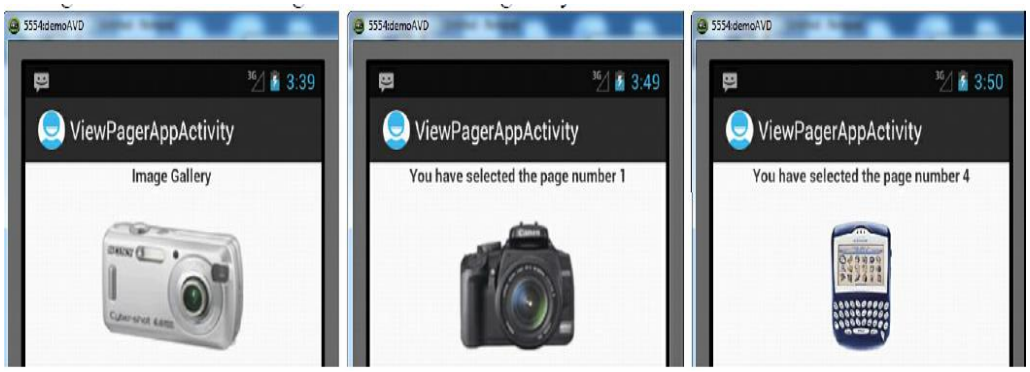
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<TextView android:id="@+id/selectedopt"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Image Gallery "
android:gravity="center"
android:textStyle="bold" />
<android.support.v4.view.ViewPager
android:id="@+id/viewpager"
android:layout_width="match_parent"
android:layout_height="100dip"
```

```
android:layout_marginTop="25dip" />
</LinearLayout>
```

Code Written into the Java Activity File ViewPagerAppActivity.java

```
import android.os.Bundle;
import android.app.Activity;
import android.support.v4.view.ViewPager;
import android.support.v4.view.PagerAdapter;
import android.widget.TextView;
import android.view.View;
import android.widget.ImageView;
import
android.support.v4.view.ViewPager.SimpleOnPageChangeListener;
public class ViewPagerAppActivity extends Activity {
public TextView selectedOpt;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_view_pager_app);
selectedOpt=(TextView) findViewById(R.id.selectedopt);
ViewPager viewPager = (ViewPager)
findViewById(R.id.viewpager);
viewPager.setAdapter(new ImageAdapter());
viewPager.setOnPageChangeListener(new PageListener());
}
public class ImageAdapter extends PagerAdapter {
Integer[] images = {
R.drawable.prod1,
R.drawable.prod2,
R.drawable.prod3,
R.drawable.prod4,
R.drawable.prod5
};
public Object instantiateItem(View container, int
position) {
ImageView view = new
ImageView(ViewPagerAppActivity.this);
view.setImageResource(images[position]);
((ViewPager) container).addView(view, 0);
return view;
}
@Override
public int getCount() {
return images.length;
}
```

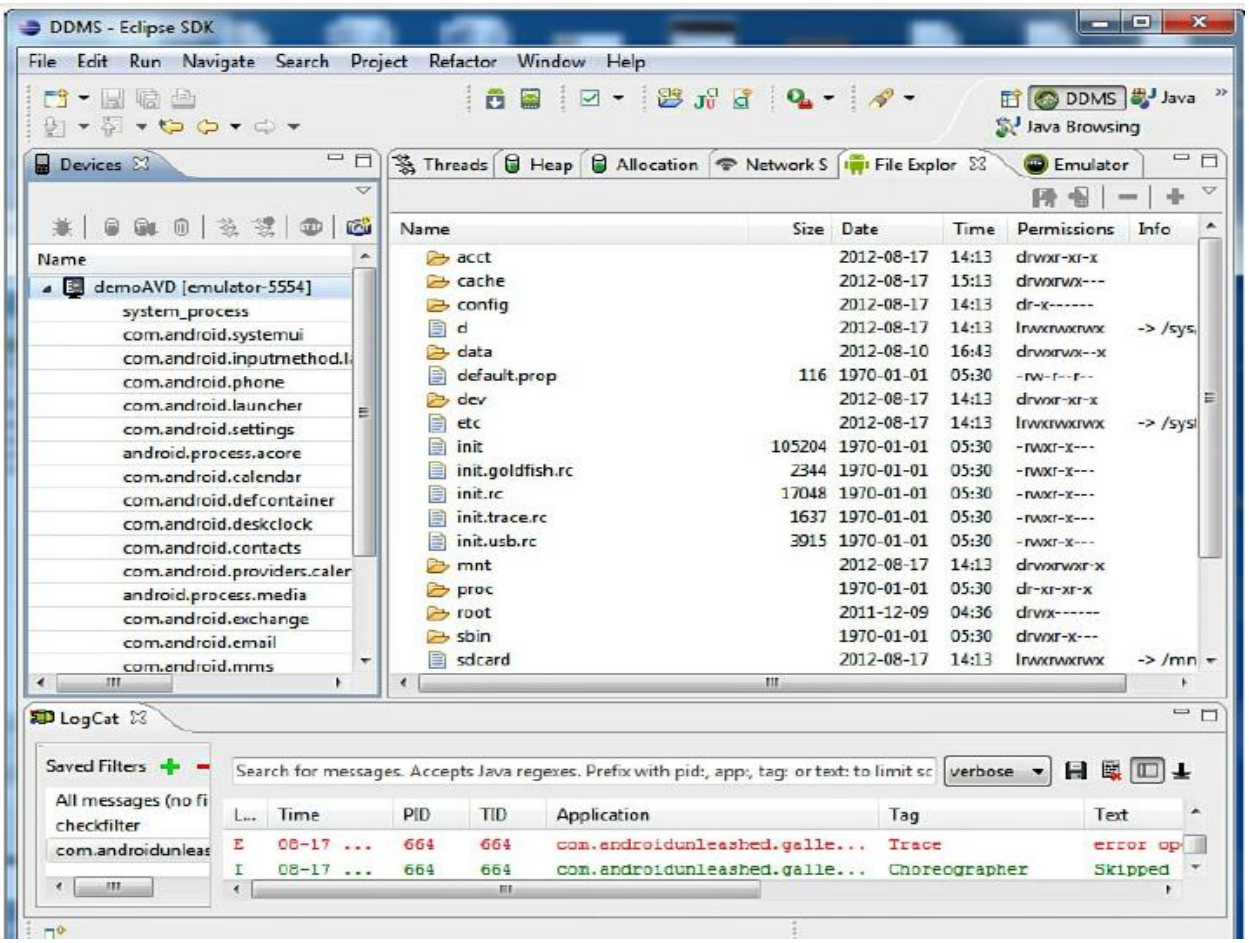
```
@Override
public void destroyItem(View arg0, int arg1, Object
arg2) {
((ViewPager) arg0).removeView((View) arg2);
}
@Override
public boolean isViewFromObject(View arg0, Object arg1)
{
return arg0 == ((View) arg1);
}
}
private class PageListener extends
SimpleOnPageChangeListener{
public void onPageSelected(int position) {
selectedOpt.setText("You have selected the page number "+position);
}
}
}
```



Using the Debugging Tool: Dalvik Debug Monitor Service (DDMS):

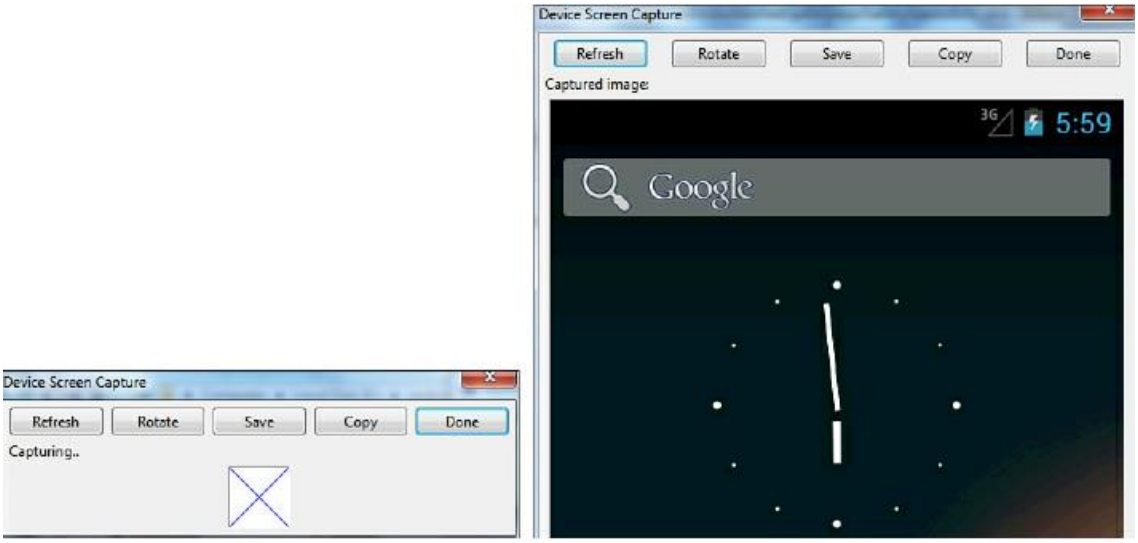
- The DDMS is a powerful debugging tool that is downloaded as part of the Android SDK.
- DDMS can be run either by selecting the DDMS icon on the top-right corner of the Eclipse IDE or by selecting the Window, Open Perspective, DDMS option.
- When we run DDMS, it automatically connects to the attached Android device or any running emulator. DDMS helps with a variety of tasks, including: port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more.

The DDMS tool window is shown below



In the upper-left pane of the DDMS window, we see a Devices tab that displays the list of Android devices connected to your PC. In Devices tab you see some icons. They are:

- Debug—Used to debug the selected process.
- Update Heap—Enables heap information of the process.
- Dump HPROF file—Shows the HPROF file that can be used for detecting memory leaks.
- Cause GC—Invokes Garbage Collection.
- Update Threads—Enables fetching the thread information of the selected process.
- Start Method Profiling—Used to find the number of times different methods are called in an application and the time consumed in each of them.
- Stop Process—Stops the selected process.
- Screen Capture—Captures our device/emulator **.** If the application is runnScreen ing and its output is being displayed through the device/emulator, clicking the Screen Capture icon displays the Device Screen Capture dialog box.



The meaning of the buttons shown at the top in the Device Screen Capture dialog box is shown here:

• **Refresh—**Updates the captured image.

• **Rotate—**With each click of this button, the captured image rotates 90 degrees in the counterclockwise direction.

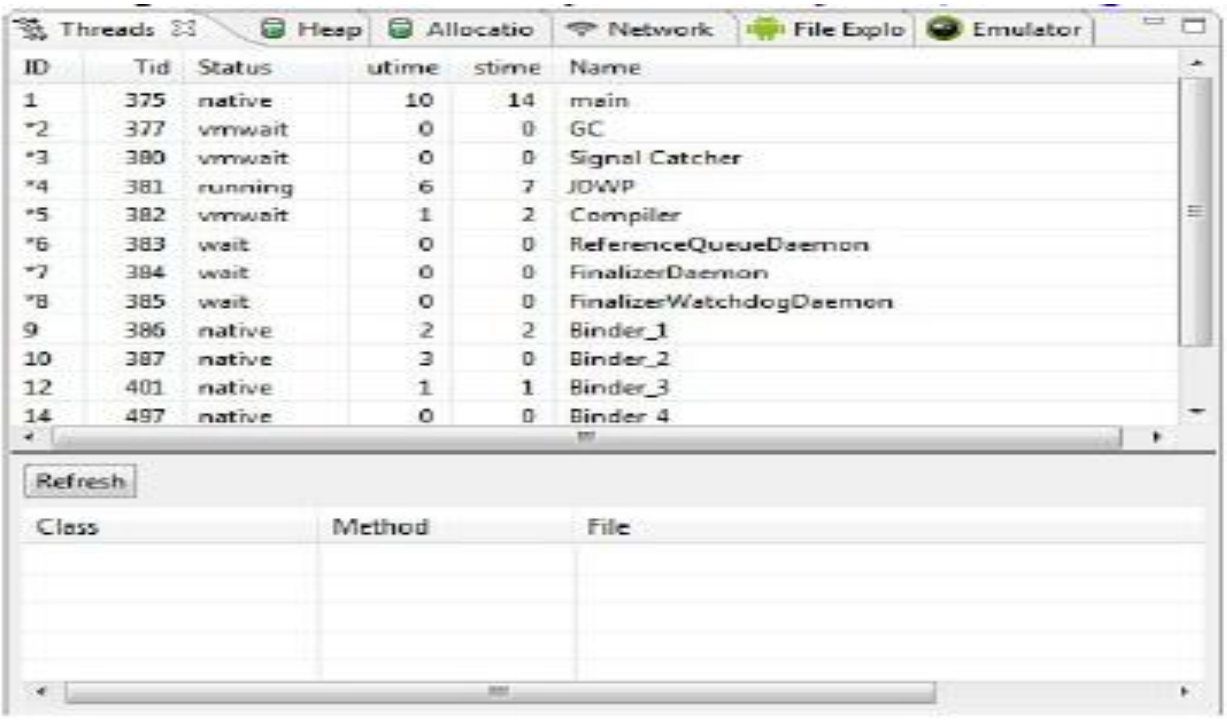
• **Save—**Saves the captured image as a .png file.

• **Copy—**Copies the captured image to the clipboard.

• **Done—**Closes the Device Screen Capture dialog.

Back to DDMS, on the right pane, we find the following tabs:

1. **Threads**-Displays information about the threads within each process.



The meaning of the buttons shown at the top in the Threads dialog box is shown here:

Thread ID—Displays the unique ID assigned to each thread

• **Status—**Displays the current status of the thread—whether it is in running, sleeping, starting,

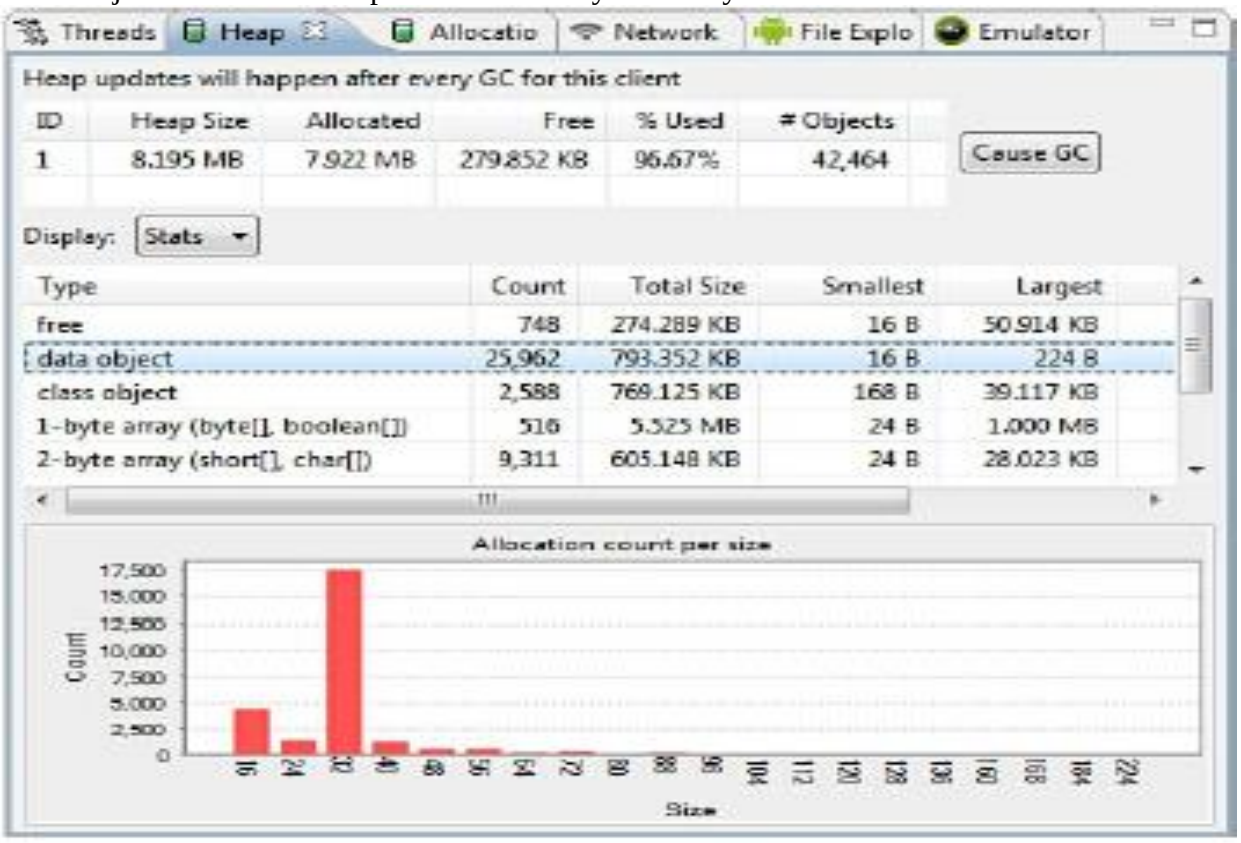
waiting, native, monitor, or zombie state

• **utime**—Indicates the cumulative time spent executing user code

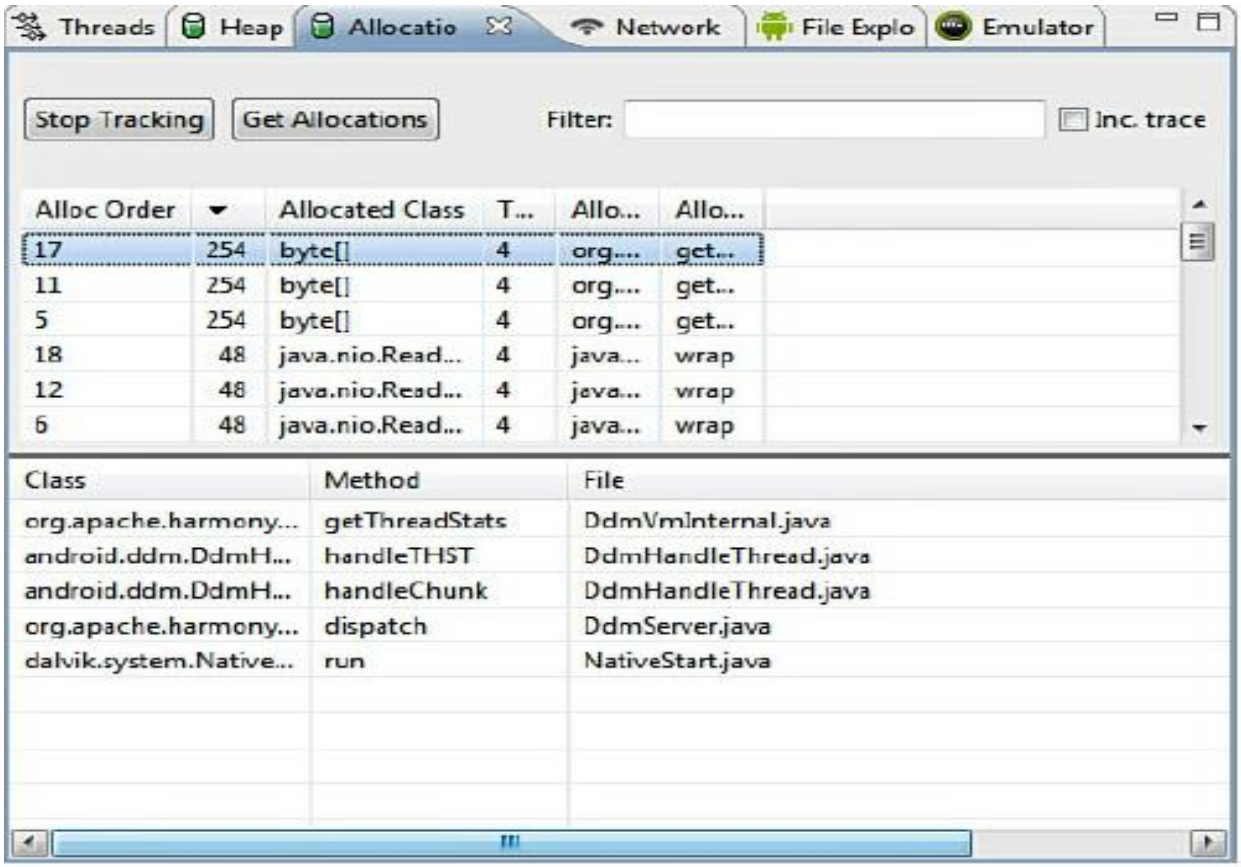
• **stime**—Indicates the cumulative time spent executing system code

• **Name**—Displays the name of the thread

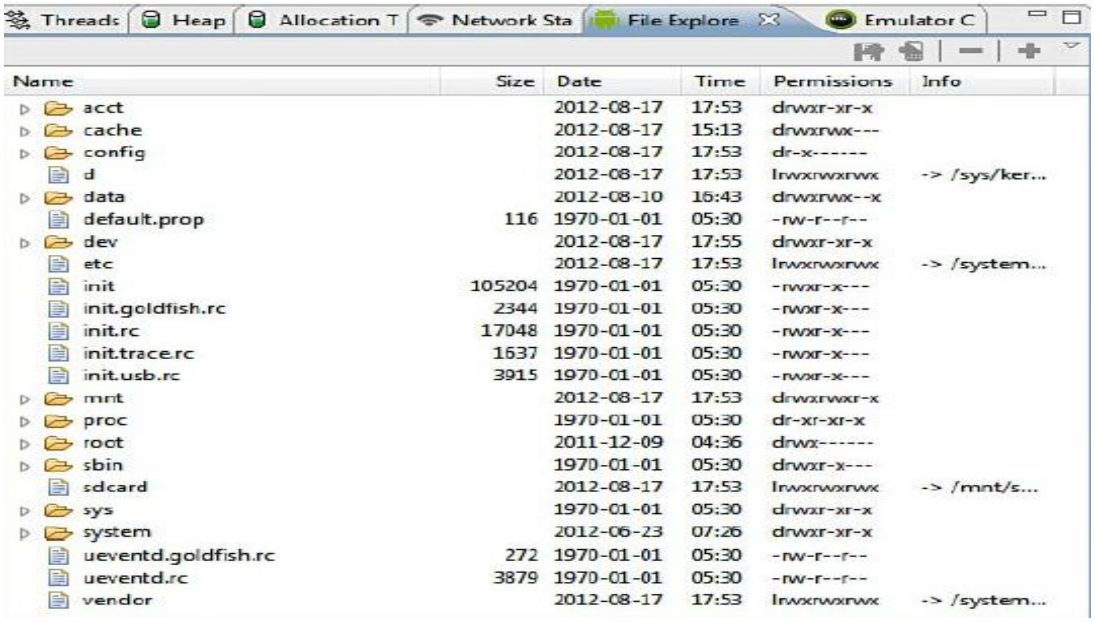
2. **Heap**-Displays the heap information of the process (provided the Update Heap button from the Devices tab has been clicked). Select the Cause GC button to begin the garbage collection process. The object types and the size of memory allocated to them are displayed. After we select an object type, a bar graph is displayed, showing the number of objects allocated for a particular memory size in bytes



3. **Allocation Tracker**—Tracks the objects allocated to an application. Click the Start Tracking button, interact with the application, and then click Get Allocations to see the list of objects allocated to the application . After we click the Get Allocations button again, the newly allocated objects are added to the earlier displayed list of allocated objects. We can also click the Stop Tracking button to clear the data and restart.

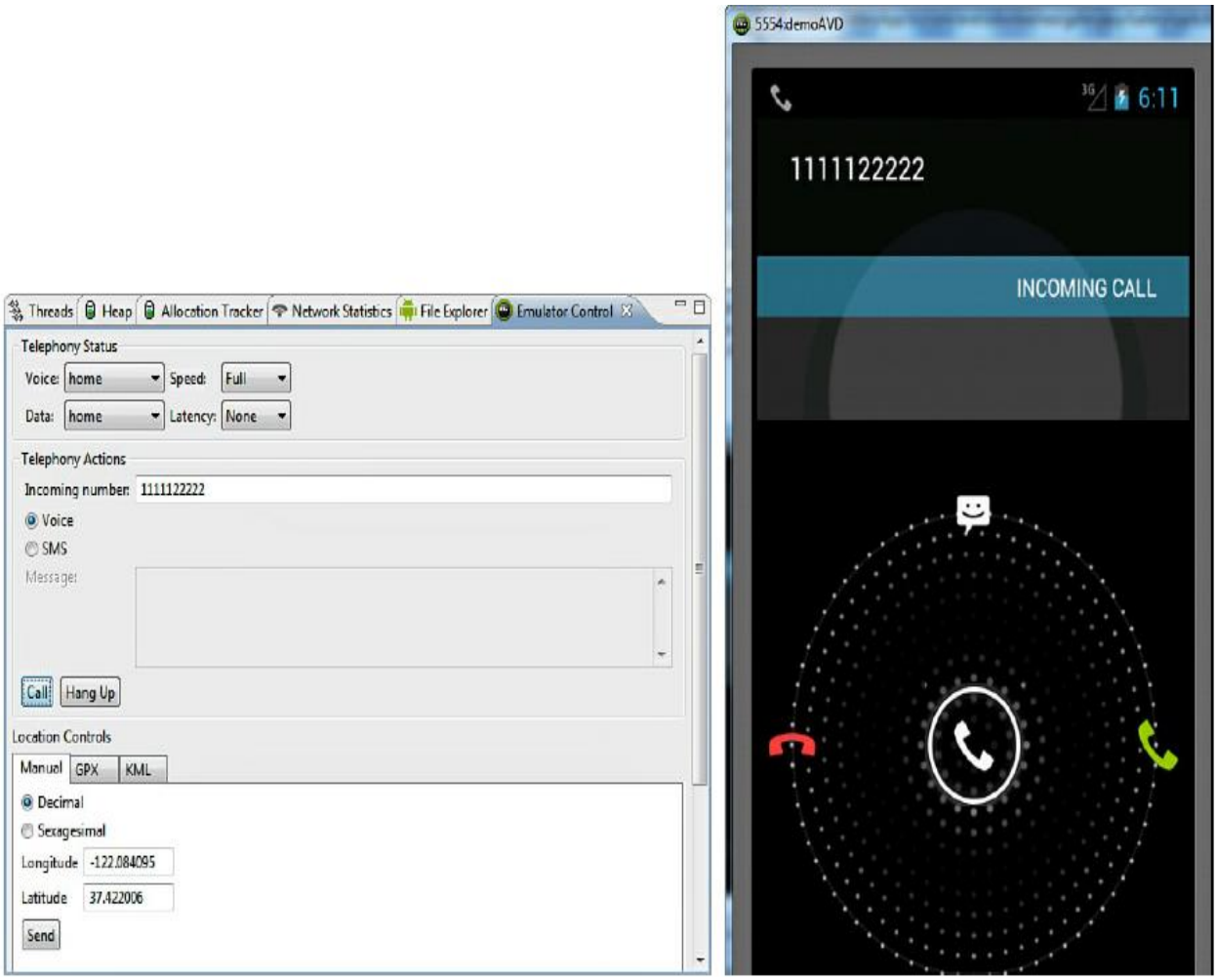


4. **Network Statistics**—Helps us in getting information regarding network usage of our application, that is, when our app made network requests, speed of data transfer—and other related information.



5. **File Explorer**—Displays the file system on the device, as shown in Figure (right). We can view and delete files on the device/emulator through this tab. We can even push or pull files from the device using the two icons, Pull a file from the device and Push a file onto the device, that are shown at the top. To copy a file from the device, select the file in the File Explorer and click the Pull a file from the device button. The Get Device File dialog box opens up, prompting us to specify the path and filename where we want to store the

pulled device file. Similarly, to copy a file to the device, click the Push file onto the device button in the File Explorer tab. The Put File on Device dialog box opens up, letting us browse the local disk drive. Select the file we want to copy to the device and click Open button to copy it to the device.



6. **Emulator Control-** Right of the File Explorer tab is the Emulator Control tab that can be used to simulate incoming phone calls, SMS messages, or GPS coordinates. To simulate an incoming phone call, select the Voice option, provide the incoming phone number, and click the Call button, as shown in Figure (left). In the emulator, an incoming call appears, prompting the user to answer the call in Figure (right). The incoming call can be ended either by clicking the End button in the emulator or by clicking the Hang Up button in the Emulator Control tab.
7. **LogCat tab-**The LogCat tab shows all messages of the device, including exceptions and those placed in the application to see the intermediate results. We can also set up filters to watch filtered log information. The Console tab displays the messages related to the starting of the activity.

Debugging Applications:

- The two most common ways of debugging an application and finding out what went wrong are placing breakpoints and displaying log messages.

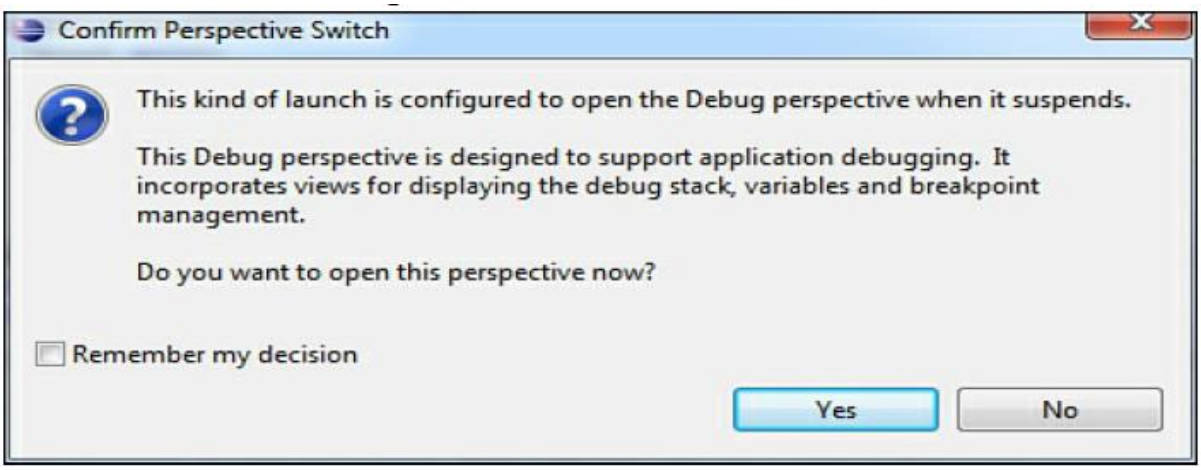
Placing Breakpoints in an Application

- To place a breakpoint in an application, select the line of code where you want to place a breakpoint and either press Ctrl+Shift+B, select Run, Toggle Breakpoint, or double-click in the marker bar to the left of the line in the Eclipse code editor.
- You can place as many breakpoints as you want in our application.

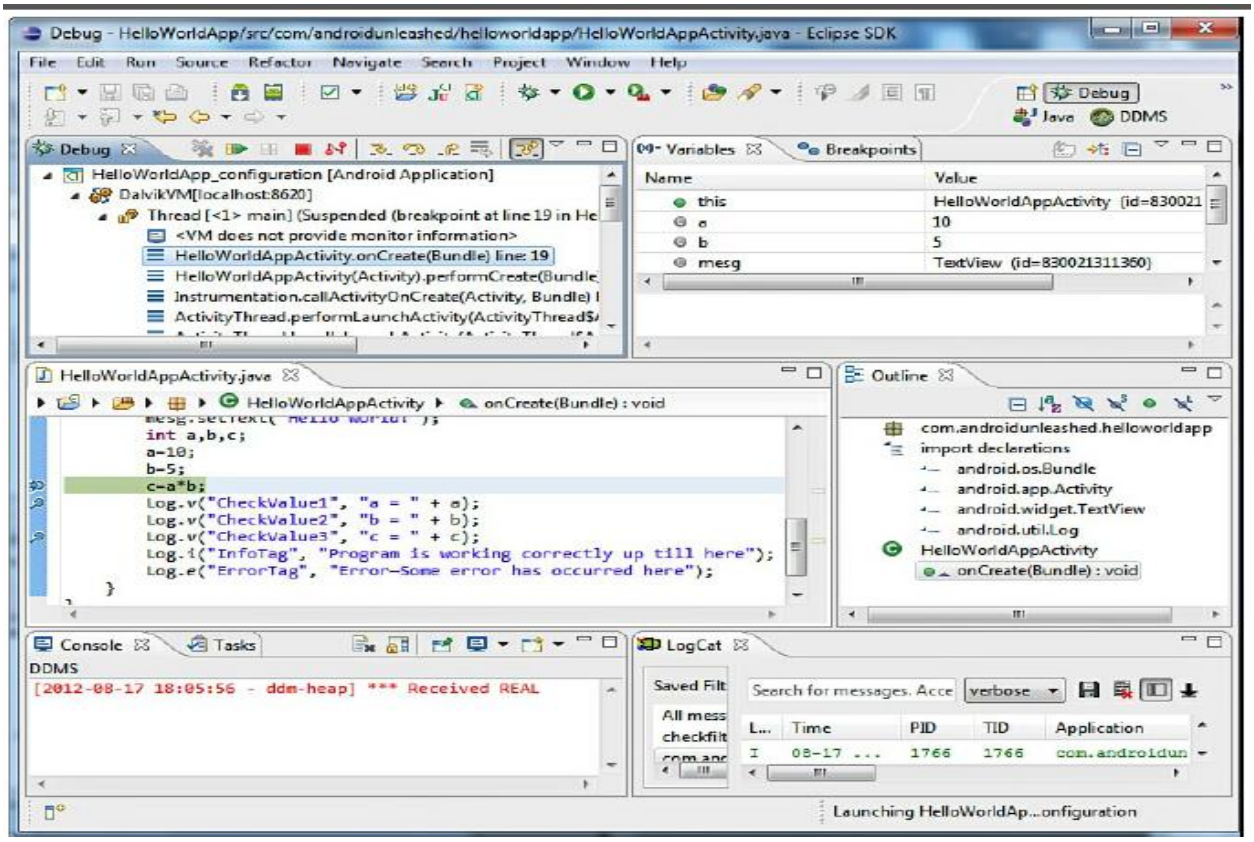
*HelloWorldAppActivity.java

```
package com.androidunleashed.helloworldapp;

import android.os.Bundle;
import android.app.Activity;
import android.widget.TextView;
import android.util.Log;

public class HelloWorldAppActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_world_app);
        TextView mesg = (TextView)findViewById(R.id.message);
        mesg.setText("Hello World!");
        int a,b,c;
        a=10;
        b=5;
        c=a*b;
        Log.v("CheckValue1", "a = " + a);
        Log.v("CheckValue2", "b = " + b);
        Log.v("CheckValue3", "c = " + c);
        Log.i("InfoTag", "Program is working correctly up till here");
        Log.e("ErrorTag", "Error-Some error has occurred here");
    }
}
```

- To stop execution at the breakpoints, debug it by either pressing F11, selecting Run, Debug, or right-clicking the project in Package Explorer and selecting Debug As, Android Application.



Using the Debug Perspective:



The following panes are visible by default in Debug perspective:

• **Debug—**On the top left, this pane displays the application being debugged, along with its currently running threads.

• **Variables—**Displays values of the variables at the specific breakpoints.

• **Breakpoints—**Lists all the breakpoints inserted in the code.

• **Editor—**At the middle left, this pane displays the application code pointing to the breakpoint where the application is currently suspended.

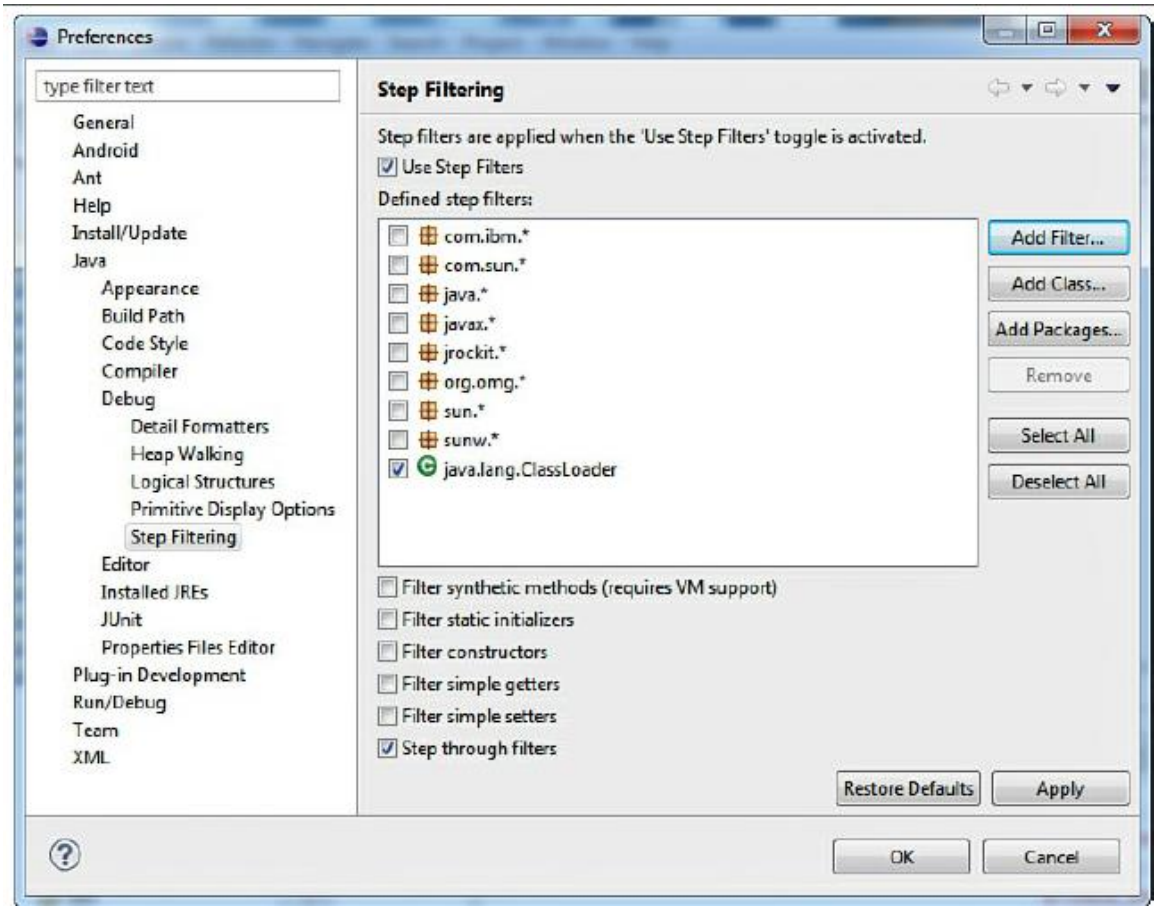
• **Outline—**At the center right, this pane lists the imports, classes, methods, and variables used in the application. When we select an item in the Outline pane, the matching source code in the Editor pane is highlighted.

• **Console—**At the bottom left, the pane displays the status of emulator/device activity, such as the launching of activities.

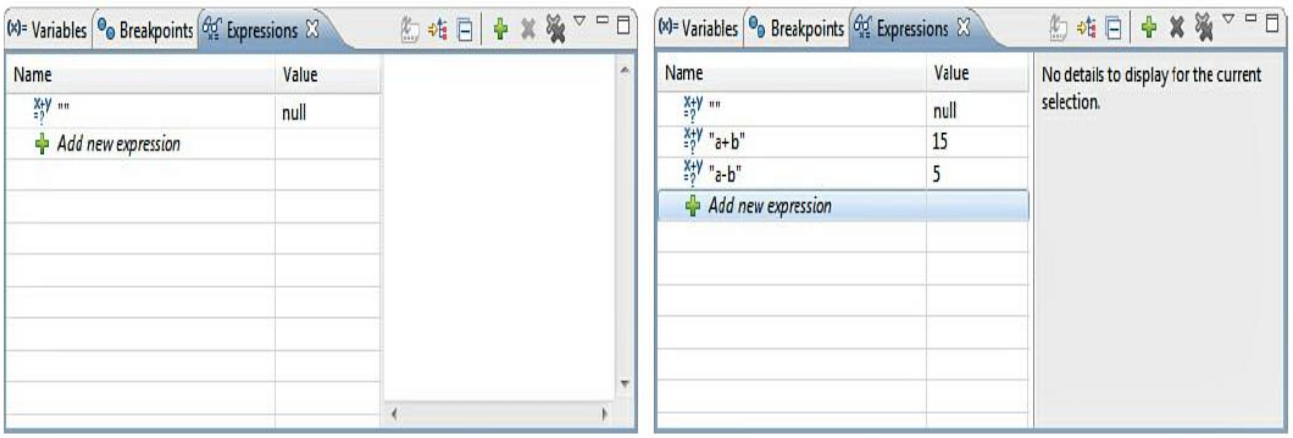
• **LogCat—**At the bottom right, this pane displays the system log messages.

- **Debug Pane:**
- Remove All Terminated Launches—Clears all terminated processes from the display.
- Resume (F8)—Resumes execution of the currently suspended debugging session.
- Suspend—Pauses the selected debugging session.
- Terminate (Ctrl+F2)—Ends the selected debugging session.
- Disconnect—Disconnects the debugger from the selected debug target. The disconnected process can be relaunched by right-clicking it in the Debug pane and selecting the Relaunch option.
- Step Into (F5)—Executes the current statement or method and proceeds to the next method call/statement.

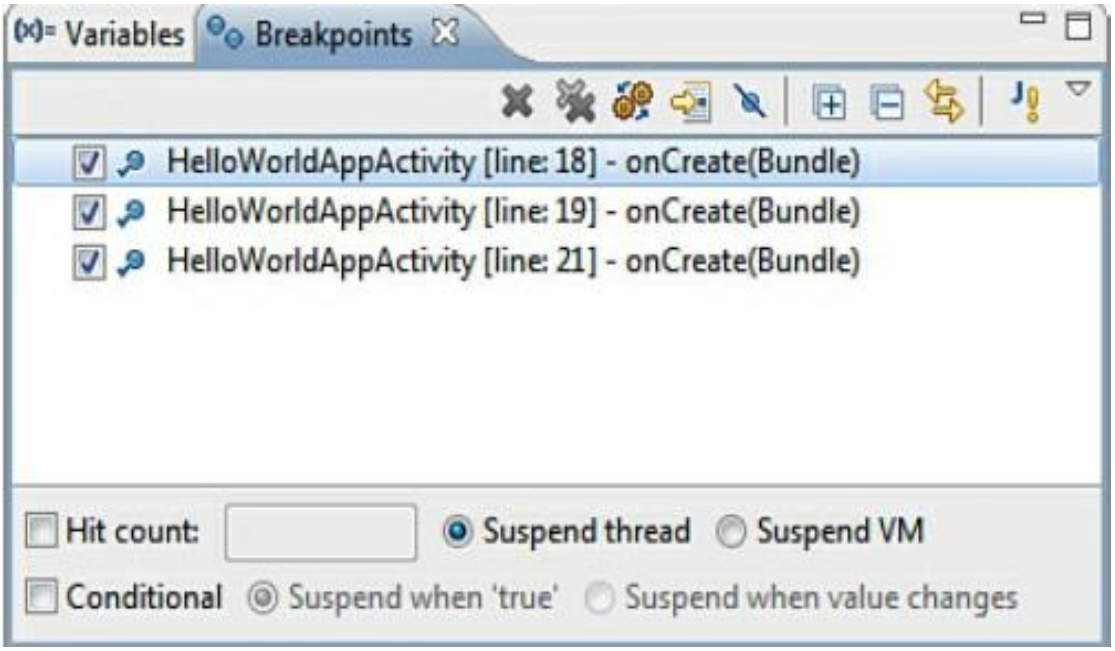
- Step Over (F6)—Steps over the next method call/statement. On every Step-Over, we can examine the values of the variables and output.
- Step Return (F7)—Returns from the method that has been stepped into.
- Drop To Frame—Used to rerun a part of an application. Select a stack frame in the call stack and select this option to return the application from there.
- Use Step Filters—Ensures that all step functions apply step filters. The step filters are used to filter out the classes that we don't want to step into. To specify the classes that we want to filter out while stepping, select Window, Preferences. In the Preferences window that opens, navigate to Java, Debug, Step Filtering.



- Expressions Pane:Expressions Pane is used to compute expressions with different variables of the application.The Expressions pane is not visible by default. To open it, select Window, Show View, Expressions.We can add our own expressions by clicking the + Add new expression button shown in the last row of the Expressions pane. Remember that the variables (if any) used in the expression must exist in the application. After writing an expression, press the Enter key to see the result, which is displayed in the Value column.

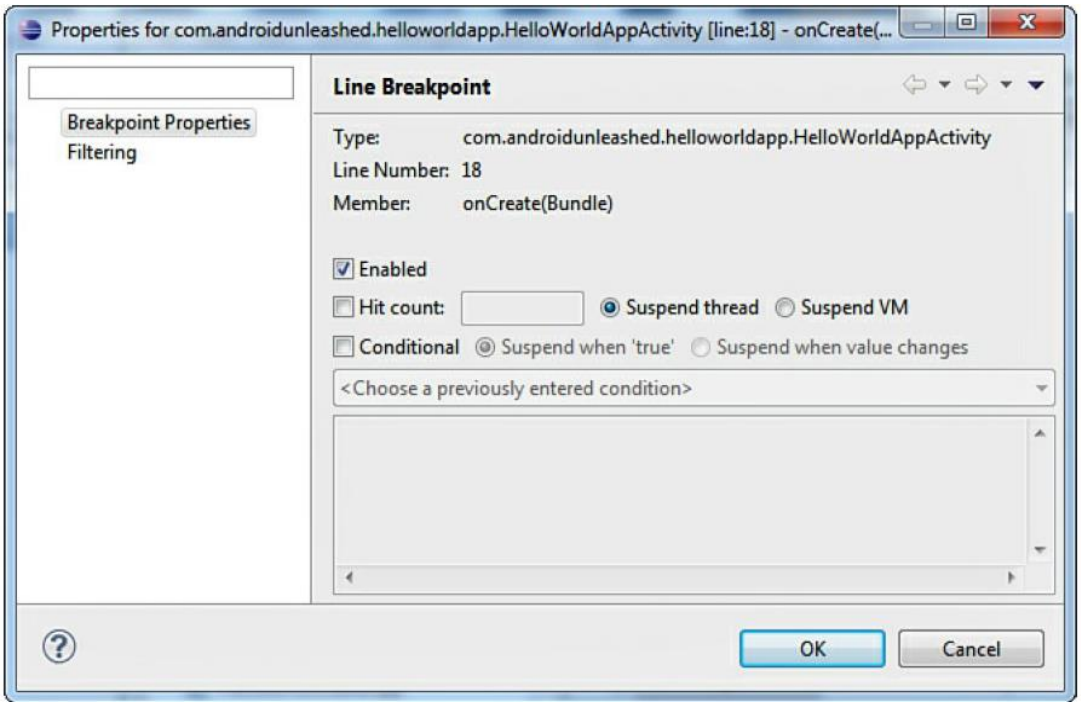


- **Breakpoints Pane-**The Breakpoints pane displays all the inserted breakpoints in the application, as shown in below figure.This pane helps in enabling, disabling, skipping, and removing breakpoints. It also helps in suspending the breakpoint on specific conditions.



At the bottom of the Breakpoints pane, we find two check boxes: Hit count and Conditional. The Hit count is for specifying the number of times we want the breakpoint to occur before suspending the thread execution. The Conditional check box is for specifying a condition when we want to suspend a thread.

From the Editor pane, right-click the breakpoint (blue dot) in the marker bar and select Breakpoint Properties from the context menu that opens. A dialog box appears, as shown in below figure



The following buttons appear at the top of the Breakpoints pane, from left to right:

• **Remove Selected Breakpoints—**Removes the selected breakpoint. You can also remove breakpoint from the Editor pane by double-clicking on the blue dot that represents the breakpoint in the marker bar. You also can right-click on the blue dot and select Toggle Breakpoint from the context menu that opens.

• **Remove All Breakpoints—**Removes all breakpoints from the application.

• **Show Breakpoints Supported by Selected Target**—Displays the breakpoints that are supported by the currently selected debug target.

• **Go to File for Breakpoint—**Highlights the statement in the Editor pane where the breakpoint is suspended.

• **Skip All Breakpoints—**Ignores all breakpoints.

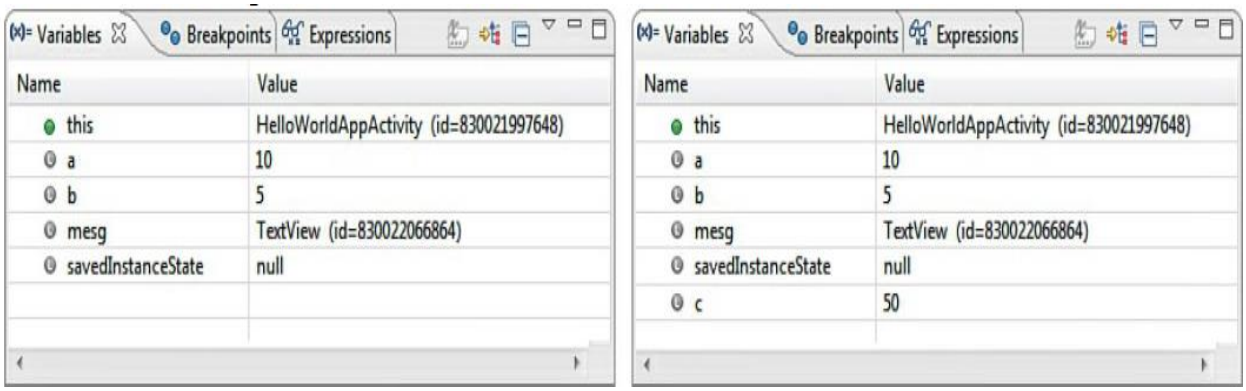
• **Expand All—**Expands all the collapsed elements in the view.

• **Collapse All—**Collapses all the expanded elements in the view.

• **Link with Debug View—**Highlights the selected breakpoint when the application moves from one breakpoint to another in the Debug view.

• **Add a Java Exception Breakpoint—**Lets you set breakpoints that are suspended when exceptions are thrown. A thread can be suspended when the specified exception occurs, whether it is uncaught, caught, or both.

- **variables pane:**The Variables pane displays values of the variables associated with the stack frame selected in the Debug pane.As we step into the application, new variables may be added and the values of the existing variables display their current value, as shown in Figure (right). To see the values of the variables up to a particular statement, you can place the cursor on a particular statement in the Editor pane and select Run-To Line from the Run menu or press Ctrl+R to continue execution up to the selected statement.



- **Adding Logging Support to Android Applications**

LogCat is commonly used for debugging an application. This utility is provided through the Log class of the android.util package and displays the log messages, exceptions, warnings, System.out.println, and intermediate results that occur during runtime. The methods in the android.util.Log class are shown in Table.

Method	Purpose
`Log.e()`	Log errors
`Log.w()`	Log warnings
`Log.i()`	Log informational messages
`Log.d()`	Log debug messages
`Log.v()`	Log verbose messages

To add logging support to our application, add the following import statement to the Activity file for the Log class:

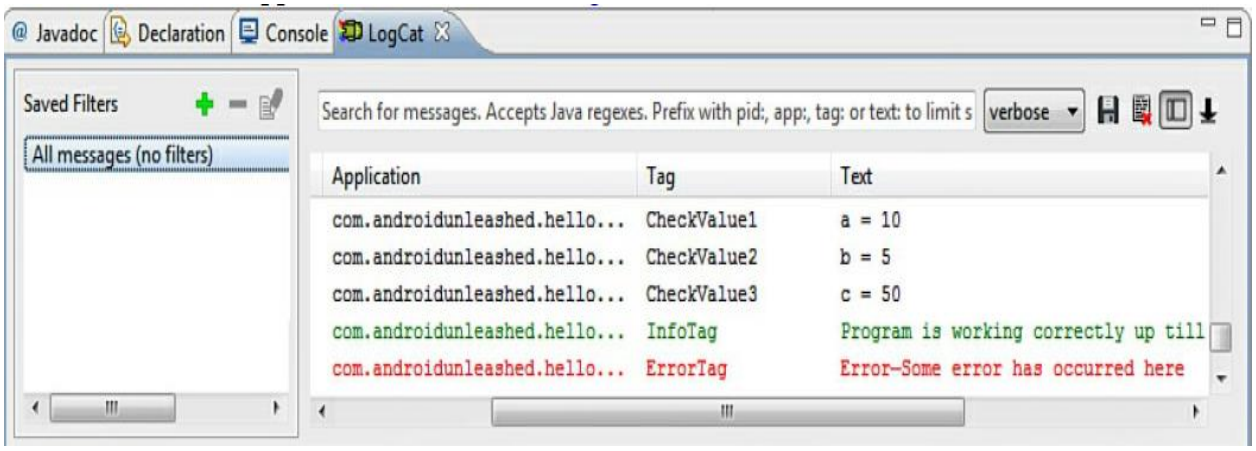
```
import android.util.Log;
```

The logging messages need to be tagged as follows:

```
Log.i("debug_tag", "Message to display");
```

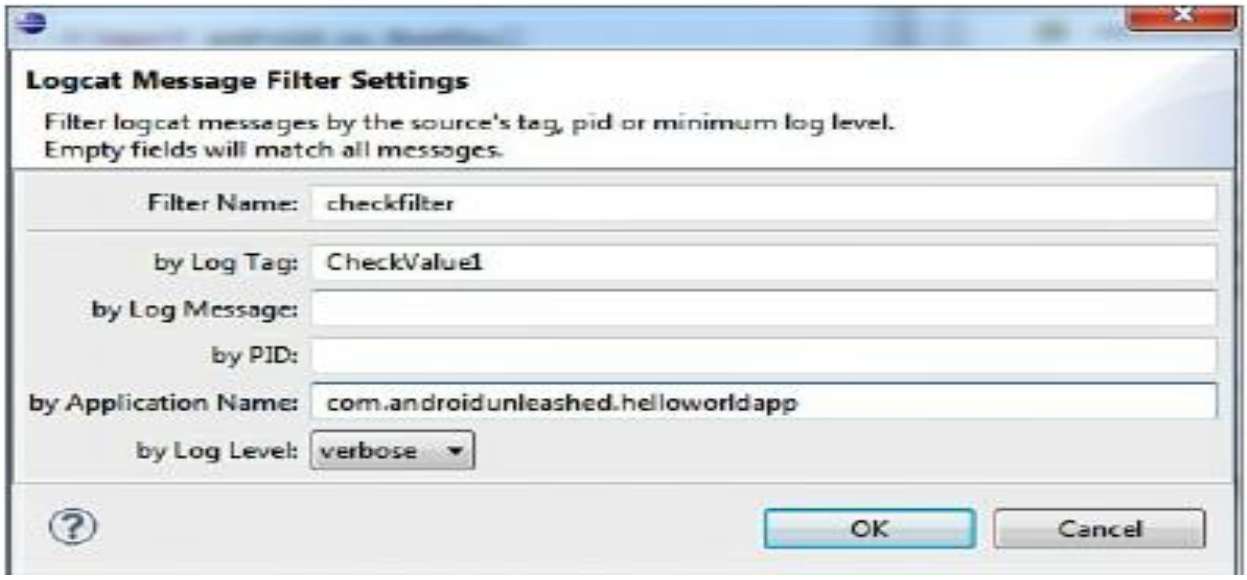
Example:

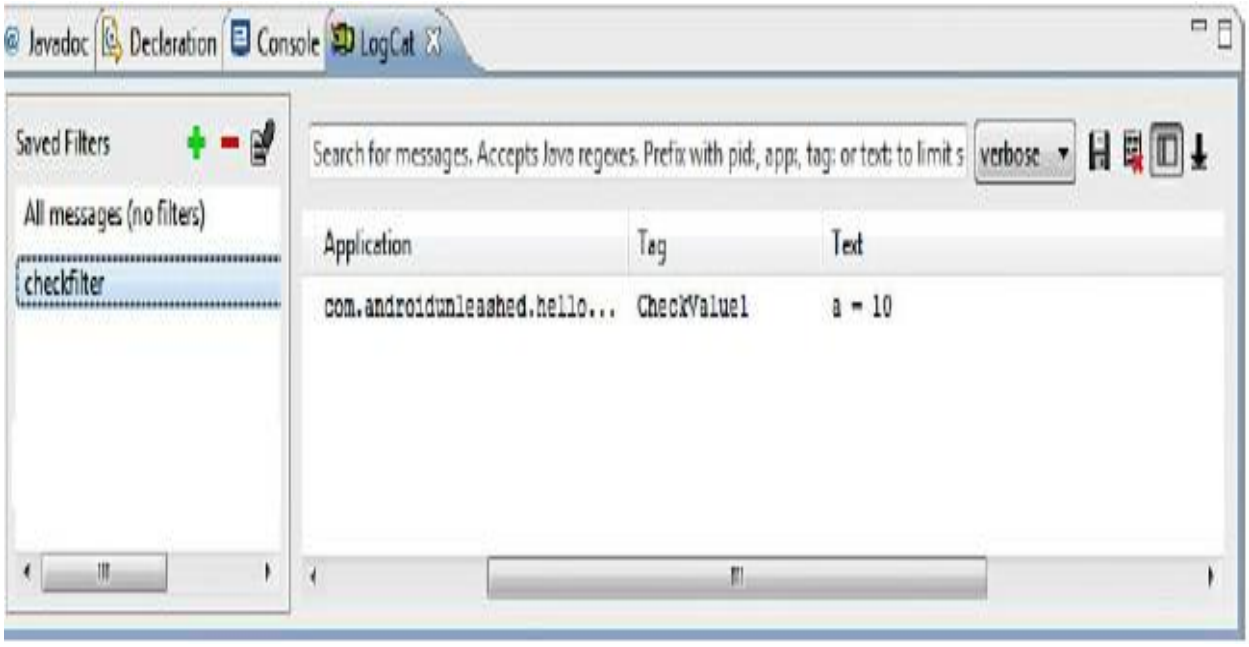
```
Log.i("InfoTag", "Program is working correctly up till here");
Log.e("ErrorTag", "Error--Some error has occurred here");
```



To see only the desired log messages, we can set up a filter by clicking the green plus sign in the LogCat pane. We see a dialog box, Logcat Message Filter Settings, as shown in below figure (left). Here we have to specify the Filter Name, the Log Tag whose messages we want to see, the

application name if we want to see log messages related to the entire application, and Log Level to filter the log messages on the basis of severity level.





Let's specify the filter name as checkfilter and Log Tag as CheckValue1, as we wish to see the log messages related to this tag only. Specify the application name as com.androidunleashed.helloworldapp to focus only on the application's logs. After we select OK, a filter called checkfilter is added and appears as a new tab in LogCat. After we select the checkfilter tab, only the log messages that satisfy the specified criteria are displayed, as shown in above figure (right)

What are Dialogs:

A Dialog is small window that prompts the user to a decision or enter additional information. Dialogs are also used to guide users in providing requested information, confirming certain actions, and displaying warnings or error messages.
Different dialog window types provided by android SDK is chown below:
Dialog—The basic class for all dialog types.
• **AlertDialog—**A dialog with one, two, or three Button controls.
• **CharacterPickerDialog—**A dialog that enables you to select an accented character associated with a regular character source.
• **DatePickerDialog—**A dialog that enables you to set and select a date with a DatePicker control.
• **ProgressDialog—**A dialog that displays a ProgressBar control showing the progress of a designated operation.
• **TimePickerDialog—**A dialog that enables you to set and select a time with a TimePicker control.Each dialog window is defined within the activity where it will be used.

The following is a list of the Activity class dialog methods:
• **showDialog()**—Displays a dialog and creates a dialog if one does not exist. Each dialog has a special dialog identifier that is passed to this method as a parameter.
• **onCreateDialog()**—The callback method that executes when the dialog is created for the first time. It returns the dialog of the specified type.
• **onPrepareDialog()**—The callback method used for updating a dialog.
• **dismissDialog()**—Closes the dialog whose dialog identifier is supplied to this method. The dialog can be displayed again through the showDialog() method.
• **removeDialog()**—The dismissDialog() method doesn't destroy a dialog. The dismissed dialog can be redisplayed from the cache. If we do not want to display a dialog, we can remove it from the activity dialog pool by passing its dialog identifier to the removeDialog() method.

In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below

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

Methods of the AlertDialog.Builder Subclass

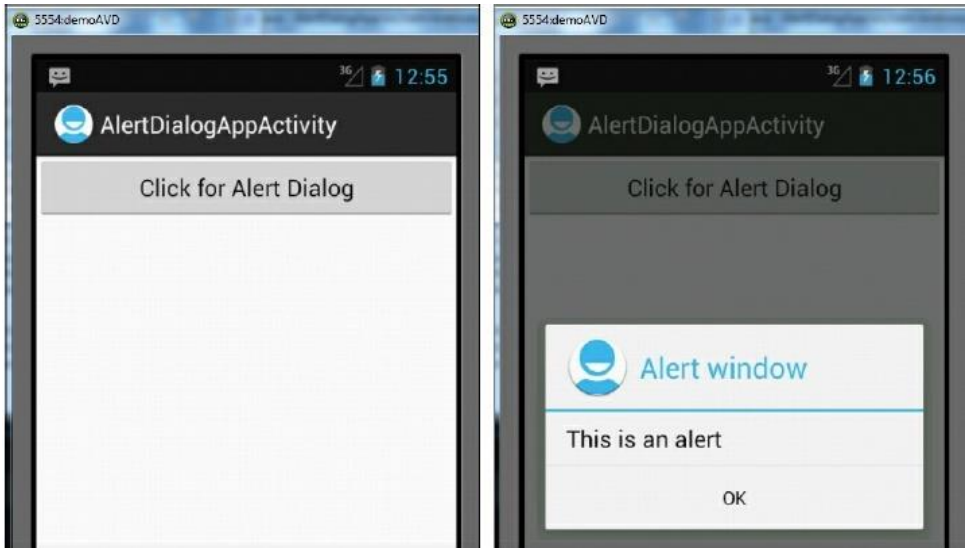
The methods of the AlertDialog.Builder subclass that we can use to configure the AlertDialog box are:
• **setTitle() and setIcon()—**For specifying the text and icon to appear in the title bar of the dialog box.
• **setMessage()—**For displaying a text message in the dialog box.
• **setPositiveButton(), setNeutralButton(), and setNegativeButton()—**For configuring the following three buttons:
• **Positive button—**Represents the OK button.
• **Negative button—**Represents the Cancel button.
• **Neutral button—**Represents a button to perform a function other than OK or Cancel.

The Layout File activity_alert_dialog_app.xml After Adding the Button Control

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Button
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/click_btn"
android:text="Click for Alert Dialog" />
</LinearLayout>
```

Code Written into the Java Activity File AlertDialogAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.view.View;
import android.app.AlertDialog;
import android.content.DialogInterface;
public class AlertDialogAppActivity extends Activity implements
OnClickListener {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_alert_dialog_app);
Button b = (Button)this.findViewById(R.id.click_btn);
b.setOnClickListener(this);
}
@Override
public void onClick(View v) {
AlertDialog.Builder alertDialog = new AlertDialog.Builder(this);
alertDialog.setTitle("Alert window");
alertDialog.setIcon(R.drawable.ic_launcher);
alertDialog.setMessage("This is an alert");
alertDialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int buttonId) {
return;
}
});
alertDialog.show();
}
```



DatePickerDialog:

DatePickerDialog allows the user to set,view and change the system date. The user can modify the year,month and date by using a constructor to display through this dialog.A calendar instance is used to initialize the values. The constructor includes a callback listener to inform the current Context when the date has been set or changed

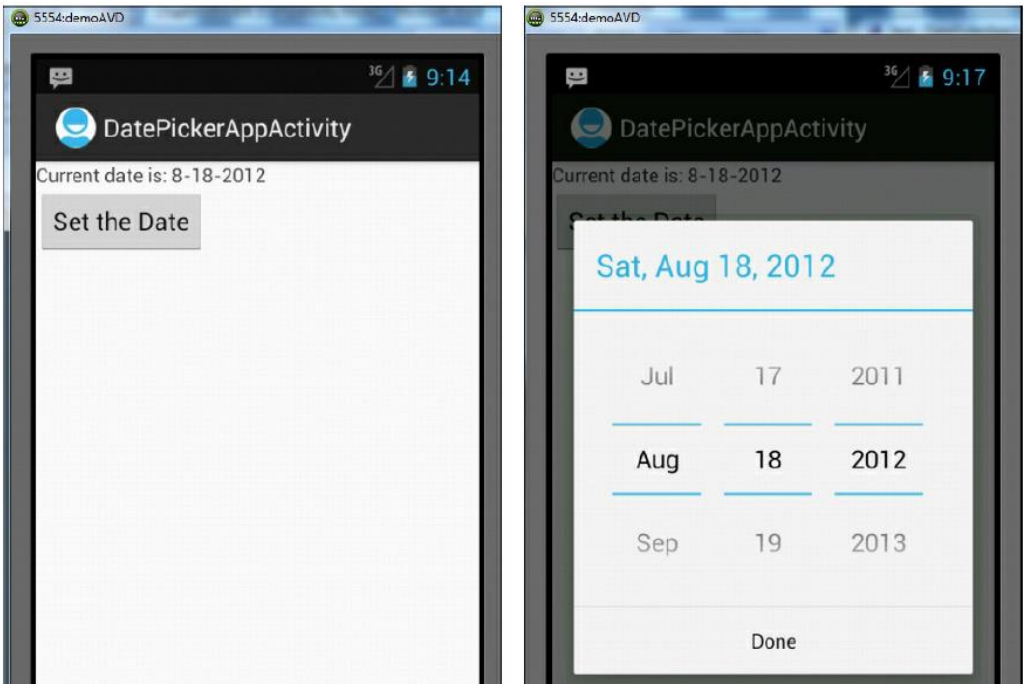
The Layout File activity_date_picker_app.xml After Adding the TextView and Button Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView android:id="@+id/datevw"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<Button android:id="@+id/date_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Set the Date" />
 </LinearLayout>
```

Code Written into the Java Activity File DatePickerAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
import java.util.Calendar;
import android.app.DatePickerDialog;
import android.view.View.OnClickListener;
```

```
import android.view.View;
import android.widget.DatePicker;
public class DatePickerAppActivity extends Activity {
private TextView dispDate;
private int yr, mon, dy;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_date_picker_app);
dispDate = (TextView) findViewById(R.id.datevw);
Button dateButton = (Button)
findViewById(R.id.date_button);
final Calendar c = Calendar.getInstance();
yr = c.get(Calendar.YEAR);
mon = c.get(Calendar.MONTH);
dy = c.get(Calendar.DAY_OF_MONTH);
dispDate.setText("Current date is: "+(mon+1)+"-"+dy+"-"+yr);
dateButton.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
new DatePickerDialog(DatePickerAppActivity.this,dateListener, yr,mon, dy).show();
}
});
}
private DatePickerDialog.OnDateSetListener dateListener =new
DatePickerDialog.OnDateSetListener() {
public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOf- Month){
yr = year;
mon = monthOfYear;
dy = dayOfMonth;
dispDate.setText("Current date is: "+(mon+1)+"-"+dy+"-"+yr);
}
};
 }
```



TimePickerDialog:

TimePickerDialog allows user to set or select time via TimePicker view that allows user to perform a task o selecting the time of the day.The format of the time might either be 24hours mode or AM/PM mode.

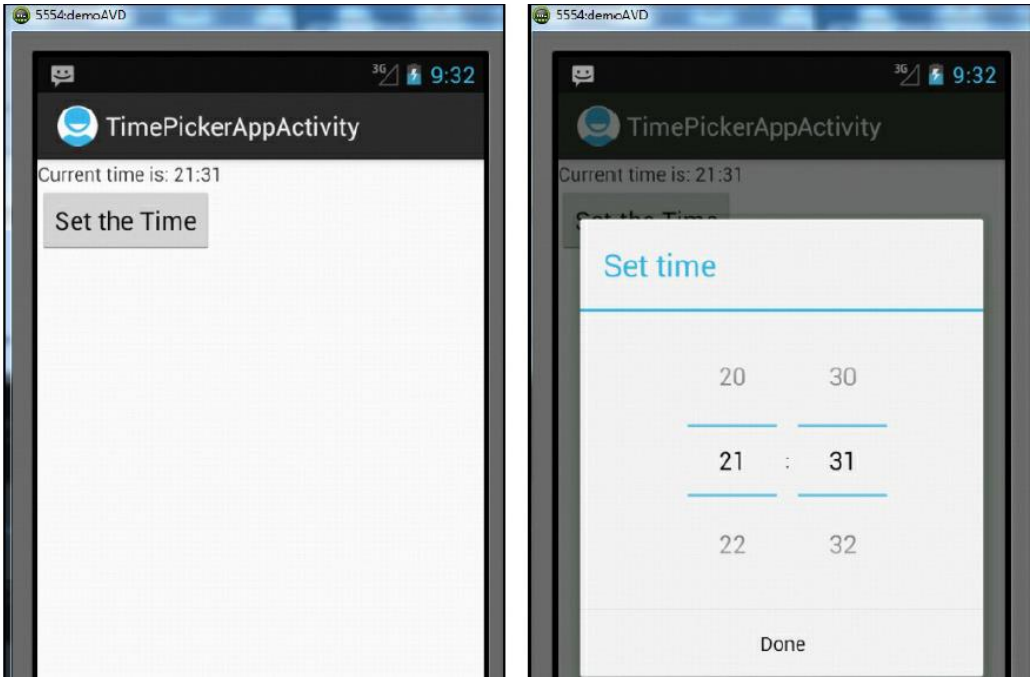
The Layout File activity_time_picker_app.xml After Adding the TextView and Button Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView android:id="@+id/timevw"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<Button android:id="@+id/time_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Set the Time" />
 </LinearLayout>
```

Code Written into the Java Activity File TimePickerAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
```

```
import java.util.Calendar;
import android.app.TimePickerDialog;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.TimePicker;
public class TimePickerAppActivity extends Activity {
private TextView dispTime;
private int h, m;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_time_picker_app);
dispTime = (TextView) findViewById(R.id.timevw);
Button timeButton = (Button)
findViewById(R.id.time_button);
final Calendar c = Calendar.getInstance();
h = c.get(Calendar.HOUR_OF_DAY);
m = c.get(Calendar.MINUTE);
dispTime.setText("Current time is: "+h+":"+m);
timeButton.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
new TimePickerDialog(TimePickerAppActivity.this,timeListener,h,m,true).show();
}
});
}
private TimePickerDialog.OnTimeSetListener timeListener =new
TimePickerDialog.OnTimeSetListener() {
public void onTimeSet(TimePicker view, int hour, int minute) {
h = hour;
m = minute;
dispTime.setText("Current time is: "+h+":"+m);
}
};
 }
```



Selecting Date and time in one application:

The Layout File activity_date_time_picker_app.xml After Adding the TextView and Button controls

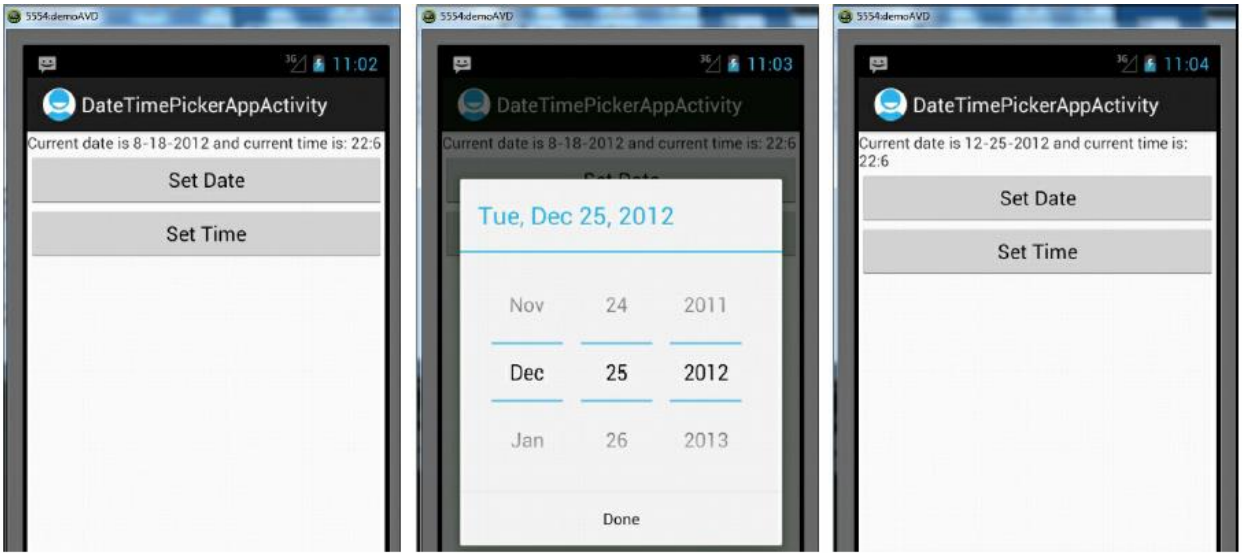
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent" >
<TextView android:id="@+id/datetimevw"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<Button android:id="@+id/date_button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Set Date" />
<Button android:id="@+id/time_button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Set Time" />
 </LinearLayout>
```

Code Written into the Java Activity File DateTimePickerAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
import java.util.Calendar;
```

```
import android.app.TimePickerDialog;
import android.app.DatePickerDialog;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.TimePicker;
import android.widget.DatePicker;
public class DateTimePickerAppActivity extends Activity {
private TextView dateTimeView;
private Calendar c;
private int h, m,yr,mon,dy;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_date_time_picker_app);
dateTimeView = (TextView) findViewById(R.id.datetimevw);
Button timeButton = (Button)findViewById(R.id.time_button);
Button dateButton = (Button)findViewById(R.id.date_button);
c = Calendar.getInstance();
h = c.get(Calendar.HOUR_OF_DAY);
m = c.get(Calendar.MINUTE);
yr = c.get(Calendar.YEAR);
mon = c.get(Calendar.MONTH);
dy = c.get(Calendar.DAY_OF_MONTH);
dateTimeView.setText("Current date is "+ (mon+1)+"-"+dy+"-"+yr+" and current time is:
"+h+":"+m);
dateButton.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
new
DatePickerDialog(DateTimePickerAppActivity.this, dateListener,yr, mon, dy).show();
}
});
timeButton.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
newTimePickerDialog(DateTimePickerAppActivity.this, timeListener,h,m,true).show();
}
});
}
private DatePickerDialog.OnDateSetListener dateListener =new DatePickerDialog.
OnDateSetListener() {
public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOf-Month)
{
yr = year;
mon = monthOfYear;
dy = dayOfMonth;
```

```
dateTimeView.setText("Current date is "+ (mon+1)+"-"+dy+"-"+yr+" and current time is: "+h+":"+m);
}
};
private TimePickerDialog.OnTimeSetListener timeListener = new TimePickerDialog.OnTimeSetListener() {
public void onTimeSet(TimePicker view, int hour, int minute) {
h = hour;
m = minute;
dateTimeView.setText("Current date is "+ (mon+1)+"-"+dy+"-"+yr+" and current time is:"+h+":"+m);
}
};
 }
```

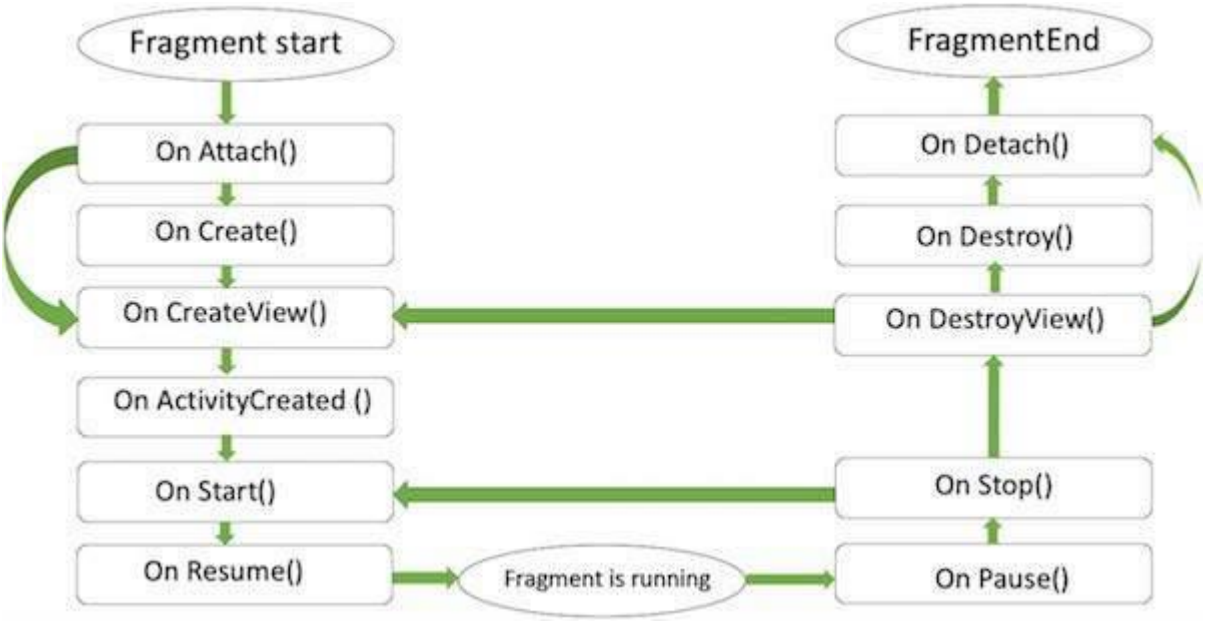


Fragments:

In Android, Fragment is a part 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. It represents a behaviour or a portion of user interface in an Activity. We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities. You can add or remove fragments in an activity while the activity is running. 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.

Fragment Life Cycle:

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



FRAGMENT LIFECYCLE

Here is the list of methods which you can to override in your fragment class −

- **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. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
- **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.

How to use Fragments?

This involves number of simple steps to create Fragments.

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the Fragment class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Creating Fragments through XML:

Code Written into the XML File fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_height="match_parent"
android:background="#FFBB00" >
<TextView
android:id="@+id/textview"
android:layout_width="wrap_parent"
android:layout_height="wrap_parent"
android:layout_margin="40dp"
android:layout_gravity=”left/center_vertical”/>
</LinearLayout>
```

Code Written into the XML File fragment2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="Wrap_parent"
android:layout_height="Wrap_parent"
android:backgrount="#00BBFF" >
<Fragment
android:id="@+id/fragment"
android:layout_width="wrap_parent"
android:layout_height="wrap_content"
android:name="My fragment1"
```

```
android:layout_alignParenttop=”true”
android:layout_alignParentleft=”true” />
 </LinearLayout>
```

Code Written into the Java Class

```
package com.androidunleashed.fragmentsapp;
import android.app.Fragment;
import android.os.Bundle;
import android.view.ViewGroup;
import android.view.View;
import android.view.LayoutInflater;
public class MyFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState)
 {
return inflater.inflate(R.layout.My_Fragment_layout, container,false);
}
 }
```

Creating Fragments through Java code:

FragmentManager:

- The FragmentManager is used to manage fragments in an activity.
- It provides the methods to access the fragments that are available in the activity.
- It also enables us to perform the FragmentTransaction required to add, remove, and replace fragments.
- To access the FragmentManager, the method used is getFragmentManager(), as shown here:

```
FragmentManager fragmentManager = getFragmentManager();
```

- To perform fragment transactions, we use the instance of the FragmentTransaction as shown here:

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

- A new FragmentTransaction is created using the beginTransaction() method of the FragmentManager.

The following code shows how to add a fragment:

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
Fragment1Activity fragment = new Fragment1Activity();
fragmentTransaction.add(R.id.fragment_container, fragment,"TAG1");
 fragmentTransaction.commit();
```

Navigating to previous fragments:

The fragment use the concept o activity stack that track the prvious activities.when the user clicks back button,then the activities in the stack will pop up and also make their view visible to the user.similarly , a fragment Transaction can be added to the back stack by calling the addToBackStack()method beore the commit() method is called.

In the statement #7,f2 replaces the f1 view in the fragment container of a layout file.Then,f1 is added back to the stack to make view invisible.when the user clicks back button.Then, it pop up from the stack and returns its views.

Retrieving the content passed through bundle:

The information passed to a fragment through a bundle object can be accessed by using getArgument() method.The code to access the bundle object i.e, passed to a fragment and the information that is passed to a string item 'Text' is as follows:

```
String Text=" ";

@override

Public void oncreate (Bundle state)

{

    Super.oncreate(state)

      if(null==state)

          State=getArgument( );

      Else

       Text=state.getString("item");

}
```

Code in the Layout File activity_fragment_code_app.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal" >
<LinearLayout
android:id="@+id/fragment1"
android:layout_weight="1"
android:layout_width="wrap_content"
android:layout_height="match_parent" />
<LinearLayout
android:id="@+id/fragment2"
android:layout_weight="0"
android:layout_width="wrap_content"
```

```
android:layout_height="match_parent" />
</LinearLayout>
```

Code Written into the Java Class File Fragment1Activity.java

```
package com.androidunleashed.fragmentcodeapp;
import android.view.View;
import android.view.LayoutInflater;
import android.app.Fragment;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.ListView;
import android.widget.ArrayAdapter;
import android.content.Context;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.TextView;
import android.content.Intent;
import android.app.FragmentManager;
public class Fragment1Activity extends Fragment{
protected static final String FRAG2 = "2";
public View onCreateView(LayoutInflater inflater, ViewGroup container,
BundlesavedInstanceState) {
Context c = getActivity().getApplicationContext();
View vw = inflater.inflate(R.layout.fragment1,container, false);
final String[] fruits={"Apple", "Mango", "Orange", "Grapes", "Banana"};
ListView fruitsList = (ListView)
vw.findViewById(R.id.fruits_list);
ArrayAdapter<String> arrayAdpt= new ArrayAdapter<String>
(c,android.R.layout.simple_list_item_1, fruits);
fruitsList.setAdapter(arrayAdpt);
final FragmentManager fragmentManager =getFragmentManager();
fruitsList.setOnItemClickListener(new
OnItemClickListener(){
@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long id){
if(null!=fragmentManager.findFragmentByTag(FRAG2))
{
TextView selectedOpt = (TextView)
getActivity().findViewById(R. id.selectedopt);
selectedOpt.setText("You have selected "+((TextView) v).getText().toString());
} else {
Intent intent = new Intent(getActivity().getApplicationContext(), ShowItemActivity.class);
intent.putExtra("item", ((TextView) v).getText().toString());
startActivity(intent);
```

```
}
}
});
return vw;
 }

 }

 }
```

Code Written into the Java Activity File FragmentCodeAppActivity.java

```
package com.androidunleashed.fragmentcodeapp;
import android.app.Activity;
import android.os.Bundle;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.res.Configuration;
public class FragmentCodeAppActivity extends Activity {
private static final String FRAG1 = "1";
private static final String FRAG2 = "2";
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_fragment_code_app);
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction=fragmentManager.beginTransaction();
if(getResources().getConfiguration().orientation==Configuration.ORIENTATION_LANDSCAP
E)
{
fragmentTransaction.add(R.id.fragment1, new Fragment1Activity(), FRAG1);
fragmentTransaction.add(R.id.fragment2, new Fragment2Activity(), FRAG2);
}
else
{
if(null!=fragmentManager.findFragmentByTag(FRAG2))
fragmentTransaction.remove(fragmentManager.findFragmentByTag(fragmentTransaction.add(R
.id.fragment1, new Fragment1Activity(), FRAG1);
}
fragmentTransaction.commit();
 }

 }
```

Creating Special Fragments:

The different types of Special Fragments are ListFragment, DialogFragment, PreferenceFragment

Creating List Fragment: A ListFragment is a fragment that contains a built-in ListView that can be set to display items from a specified data source. The item selected from the ListView is displayed through a TextView.

Code in the XML File Fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<TextView
android:id="@android:id/empty"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="New text"
android:layout_gravity=”left/center_vertical” />
<ListView
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:id=”@android:id/list”
android:layout_gravity=”lef/center_vertical” />
</LinearLayout>
```

Code in the XML File Fragment2.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:Paddingleft="128dp"
android:PaddingRight=”128dp”
android:PaddingTop=”16dp”
android:PaddingBottom=”16dp”
android:background=”#FFee00” >
<fragment
android:name="MyListFragment"
android:id="@+id/fragment"
android:layout_alignParentTop="true”
android:layout_alignParentLeft=”true”
```

android:layout_alignParentRight=”true”/>

Code in Strings.xml

```
<resources>
<string name="app_name">ListViewApp</string>
<string name="action_settings">Settings</string>
<string
name="Hello World">Helloworld</string>
<string-array name="heros">
<item>Batman</item>
<item>Superman</item>
<item>Green lantern</item>
<item>Arrow</item>
</string-array>
</resources>
```

Code in the Java Activity File ListFragAppActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.view.layoutInflater;
import android.view.View;
import android.widget.Adapter view;
import android.widget.ArrayAdapter;
import android.widget.Toast;
public class MyListFragAppActivity extends List Fragment implements Adapter
View.OnItemClickListener  {
@Override
public void onCreateView(layout Inflater,viewGroup container,Bundle savedInstanceState) {
return inflater.inflate(R.layout.my_list_fragment,container,false);
}
@Override
public void onActivitycreated(BundleSavedInstanceStae)
 {
 Super.onActivitycreated(SavedInstanceState);


 ArrayAdapter adapter =
ArrayAdapter.createFromResource(getActivity(
),R.array.heros,android.R.layout.Simple_list_item_1);
 SetListAdapter(adapter);
 getListView ( ).SetonItemClickListener(this);
 }
```

@override

Public void onItemClick(Adapter View<?> adapter view, view View, int I, long l)

{

Toast.makeText(getActivity( ),”Item”+I,Toast.LENGTH_SHORT).show( );

}

}

Dialog Fragment: DialogFragment is a utility class which extends the Fragment class. It is a part of the v4 support library and is used to display an overlay modal window within an activity that floats on top of the rest of the content.Essentially a DialogFragment displays a Dialog but inside a Fragment.

You can create Dialogs using DialogFragment in two ways:

- onCreateDialog – Here you can create the AlertDialog using the AlertDialog.Builder class.
- onCreateView – Here you can create a Dialog using a custom view defined.

In order to create a DialogFragment that shows a Dialog, we need to call the method show() on the DialogFragment instance as:

```
MyDialogFragment dialogFragment = new MyDialogFragment();
FragmentTranscation ft = getSupportFragmentManager().beginTransaction();
Fragment prev = getSupportFragmentManager().findFragmentByTag("dialog");
if (prev != null) {
ft.remove(prev);
}
ft.addToBackStack(null);
dialogFragment.show(ft, "dialog");
```

When a DialogFragment class is instantiated. Methods are called in the following order:
onCreate
onCreateDialog
onCreateView
onViewCreated
onDestroy

The code for the activity_main.xml class is given below:

```
<?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:orientation="vertical"
tools:context=".MainActivity">
<FrameLayout
android:id="@+id/frameLayout"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_above="@+id/btnEmbedDialogFragment"
android:layout_alignParentTop="true" />
<Button
android:id="@+id/btnEmbedDialogFragment"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_above="@+id/btnDialogFragment"
android:text="EMBED DIALOG FRAGMENT" />
<Button
android:id="@+id/btnDialogFragment"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:layout_marginTop="8dp"
android:text="SIMPLE DIALOG FRAGMENT" />
<Button
  android:id="@+id/btnDialogFragmentFullScreen"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/btnDialogFragment"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="8dp"
  android:text="DIALOG FRAGMENT FULL SCREEN" />
    <Button
       android:id="@+id/btnAlertDialogFragment"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_below="@+id/btnDialogFragmentFullScreen"
       android:layout_centerHorizontal="true"
       android:layout_marginTop="8dp"
       android:text="Alert Dialog Fragment" />
 <TextView
       android:id="@+id/textView"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_below="@+id/btnAlertDialogFragment"
       android:layout_centerHorizontal="true" />
</RelativeLayout>
```

The xml layout for the custom view for a DialogFragment is defined in fragment_sample_dialog.xml file as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="10dp">
   <TextView
      android:id="@+id/title"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_margin="16dp"
      android:text="Please enter your username and password" />
   <EditText
      android:id="@+id/inEmail"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Email Address"
      android:inputType="textEmailAddress" />
   <EditText
      android:id="@+id/inPassword"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Password"
      android:inputType="textPassword" />
   <Button
      android:id="@+id/btnDone"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Done" />
</LinearLayout>
```

The code for the MainActivity.java is given below:

```
package com.journaldev.androiddialogfragment;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity implements View.OnClickListener,
MyDialogFragment.DialogListener {
Button btnEmbedDialogFragment, btnDialogFragment, btnDialogFragmentFullScreen,
btnAlertDialogFragment;
```

```
    TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = findViewById(R.id.textView);
        btnEmbedDialogFragment = findViewById(R.id.btnEmbedDialogFragment);
        btnDialogFragment = findViewById(R.id.btnDialogFragment);
        btnDialogFragmentFullScreen = findViewById(R.id.btnDialogFragmentFullScreen);
        btnAlertDialogFragment = findViewById(R.id.btnAlertDialogFragment);
        btnEmbedDialogFragment.setOnClickListener(this);
        btnDialogFragment.setOnClickListener(this);
        btnDialogFragmentFullScreen.setOnClickListener(this);
        btnAlertDialogFragment.setOnClickListener(this);

    }

    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.btnEmbedDialogFragment:
                MyDialogFragment dialogFragment = new MyDialogFragment();
                FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
                ft.replace(R.id.frameLayout, dialogFragment);
                ft.commit();
                break;
            case R.id.btnDialogFragment:
                dialogFragment = new MyDialogFragment();
                Bundle bundle = new Bundle();
                bundle.putBoolean("notAlertDialog", true);
               dialogFragment.setArguments(bundle);
                ft = getSupportFragmentManager().beginTransaction();
                Fragment prev = getSupportFragmentManager().findFragmentByTag("dialog");
                if (prev != null) {
                    ft.remove(prev);
                }
                ft.addToBackStack(null);
                dialogFragment.show(ft, "dialog");
     break;
case R.id.btnDialogFragmentFullScreen:
                dialogFragment = new MyDialogFragment();
                bundle = new Bundle();
                bundle.putString("email", "xyz@gmail.com");
                bundle.putBoolean("fullScreen", true);
```

```
            bundle.putBoolean("notAlertDialog", true);
            dialogFragment.setArguments(bundle);
            ft = getSupportFragmentManager().beginTransaction();
            prev = getSupportFragmentManager().findFragmentByTag("dialog");
            if (prev != null) {
                ft.remove(prev);
            }
            ft.addToBackStack(null);
            dialogFragment.show(ft, "dialog");
            break;
        case R.id.btnAlertDialogFragment:
            dialogFragment = new MyDialogFragment();
            ft = getSupportFragmentManager().beginTransaction();
            prev = getSupportFragmentManager().findFragmentByTag("dialog");
            if (prev != null) {
                ft.remove(prev);
            }
            ft.addToBackStack(null);
            dialogFragment.show(ft, "dialog");
            break;
        }
    }

    @Override
    public void onFinishEditDialog(String inputText) {
        if (TextUtils.isEmpty(inputText)) {
            textView.setText("Email was not entered");
        } else
            textView.setText("Email entered: " + inputText);
    }
}
```

The code for the MyDialogFragment.java class is given below:

```
package com.journaldev.androiddialogfragment;

import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.DialogFragment;
import android.support.v7.app.AlertDialog;
import android.text.TextUtils;
import android.util.Log;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
public class MyDialogFragment extends DialogFragment {
    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
        if (getArguments() != null) {
            if (getArguments().getBoolean("notAlertDialog")) {
                return super.onCreateDialog(savedInstanceState);
            }
        }
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Alert Dialog");
        builder.setMessage("Alert Dialog inside DialogFragment");
        builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dismiss();
            }
        });

        builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dismiss();
            }
        });

        return builder.create();

    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container, @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_sample_dialog, container, false);

    }

    @Override
```

```
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        final EditText editText = view.findViewById(R.id.inEmail);
        if (getArguments() != null && !TextUtils.isEmpty(getArguments().getString("email")))
            editText.setText(getArguments().getString("email"));

        Button btnDone = view.findViewById(R.id.btnDone);
        btnDone.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                DialogListener dialogListener = (DialogListener) getActivity();
                dialogListener.onFinishEditDialog(editText.getText().toString());
                dismiss();
            }
        });
    }

    @Override
    public void onResume() {
        super.onResume();

    }

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("API123", "onCreate");
        boolean setFullScreen = false;
        if (getArguments() != null) {
            setFullScreen = getArguments().getBoolean("fullScreen");
        }
        if (setFullScreen)
            setStyle(DialogFragment.STYLE_NORMAL,
android.R.style.Theme_Black_NoTitleBar_Fullscreen);
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
    }
    public interface DialogListener {
        void onFinishEditDialog(String inputText);
    }
}
```

Preference Fragment

PreferenceFragment is a fragment that enables users to configure and personalize an application. The PreferenceFragment can contain several Preference Viewsference Fragment:

Preference View	Description
`PreferenceScreen`	The root element of the XML used to define a preference screen
`CheckBoxPreference`	Displays a simple check box that returns `true` when checkedh otherwise returns `false`
`ListPreference`	Displays a list of radio buttons allowing the user to select one
`EditTextPreference`	Displays a dialog with an `EditText` control allowing the user to enter text
`RingtonePreference`	Displays radio buttons indicating the ringtones available for selection
`PreferenceCategory`	Used in grouping related preferences in categories
`Preference`	A custom preference that acts like a `Button` control

Code Written into XML File preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android" >
<PreferenceCategory android:title="Category 1">
<CheckBoxPreference
android:title="Pizza"
android:defaultValue="false"
android:key="Pizzakey" />
<EditTextPreference android:key="Namekey"
android:title="Enter your name: "
android:dialogTitle="Enter your information">
</EditTextPreference>
</PreferenceCategory>
<PreferenceCategory android:title="Category 2">
<RingtonePreference android:showDefault="true"
android:key="Audio" android:title="Select sound"
android:ringtoneType="notification">
</RingtonePreference>
<ListPreference android:title="Fruits List "
android:key="fruits_list"
android:entries="@array/fruits"
android:entryValues="@array/fruitselected"
android:dialogTitle="Choose a fruit">
</ListPreference>
</PreferenceCategory>
```

```
<Preference
android:title="Submit"
android:key="submitPref" />
</PreferenceScreen>
```

The Strings Resource File strings.xml After Defining the Two Arrays

```
<resources>
<string name="app_name">PreferenceFragApp</string>
<string name="menu_settings">Settings</string>
<string
name="title_activity_preference_frag_app">PreferenceFragAppActivity</<string-array
name="fruits">
<item>Apple</item>
<item>Mango</item>
<item>Orange</item>
<item>Grapes</item>
<item>Banana</item>
</string-array>
<string-array name="fruitselected">
<item>You have selected Apple</item>
<item>You have selected Mango</item>
<item>You have selected Orange</item>
<item>You have selected Grapes</item>
<item>You have selected Banana</item>
</string-array>
</resources>
```

Code Written into PreferenceFragment PrefActivity.java

```
package com.androidunleashed.preferencefragapp;
import android.os.Bundle;
import android.app.Activity;
import android.preference.Preference;
import android.preference.Preference.OnPreferenceClickListener;
import android.preference.PreferenceFragment;
public class PrefActivity extends Activity {
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
getFragmentManager().beginTransaction().replace(android.R.new
PrefsFragment()).commit();
}
public static class PrefsFragment extends PreferenceFragment
{
@Override
public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
addPreferencesFromResource(R.xml.preferences);
Preference submitPref = (Preference)
findPreference("submitPref");
submitPref.setOnPreferenceClickListener(new
OnPreferenceClickListener()
{
public boolean onPreferenceClick(Preference
preference) {
getActivity().finish();
return true;
}
});
}
}
}
```

The Layout File activity_preference_frag_app.xml After Adding the Four TextView Controls

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/pizza"/>
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/name"/>
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/ringtone"/>
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/fruit"/>
</LinearLayout>
```

Code Written into the Main Activity File PreferenceFragAppActivity.java

```
package com.androidunleashed.preferencefragapp;
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
```

```
import android.preference.PreferenceManager;
import android.content.SharedPreferences;
import android.widget.TextView;
public class PreferenceFragAppActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_preference_frag_app);
startActivity(new Intent(this, PrefActivity.class));
}
@Override
public void onResume() {
super.onResume();
SharedPreferences
prefs=PreferenceManager.getDefaultSharedPreferences(this);
TextView pizza=(TextView)findViewById(R.id.pizza);
TextView name=(TextView)findViewById(R.id.name);
TextView ringtone=(TextView)findViewById(R.id.ringtone);
TextView fruit=(TextView)findViewById(R.id.fruit);
if(Boolean.valueOf(prefs.getBoolean("Pizzakey", false)))
pizza.setText("You have selected Pizza");
else
pizza.setText("");
ringtone.setText("The ringtone selected is"+prefs.getString("Audio","Silent"));
name.setText("The name entered is "+prefs.getString("Namekey",""));
String selectedFruit = prefs.getString("fruits_list","Apple");
fruit.setText(selectedFruit);
}
}
```