UNIT 4

Using Selection widgets and Debugging

Syllabus: Using List View, Using the Spinner control, Using the GridView Control, Creating anImage Gallery Using the ViewPager Control, Using the Debugging Tool: Dalvik Debug Monitor Service(DDMS), Debugging Application, Using the Debug Perspective.

Displaying And Fetching Information Using Dialogs and Fragments: What Are Dialogs?, Selecting the Date and Time in One Application, Fragments, Creating Fragments with java Code, Creating Special Fragments

Using List View

ListViews in Android are one way to display a scrolling list of information—like a list of news items, a list of recipes, a list of delicious biscuits, whatever! They're one of the basic building blocks of a lot of apps.List of scrollable items can be displayed in Android using ListView. It helps you to displaying the data in the form of a scrollable list. Users can then select any list item by clicking on it. ListView is default scrollable so we do not need to use scroll View or anything else with ListView.

The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database. It's one of the basic and most used UI components of android. The most common usages include displaying data in the form of a vertical scrolling list.ListView is widely used in android applications. A very common example of ListView is your phone contact book, where you have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.

Adapters use in ListView

To fill the data in a ListView we simply use adapters. List items are automatically inserted to a list using an Adapter that pulls the content from a source such as an arraylist, array or database.Adapter holds the data and send the data to adapter view, the view can take the data from adapter view and shows the data on different views like as spinner, listview, gridview etc.

The adapter pulls the items out of a data source, an array for example, and then converts each item into a view which it then inserts into the ListView.ListView is a subclass of AdapterView and it can be populated by binding to an Adapter, which retrieves the data from an external source and creates a View that represents each data entry.

In android commonly used adapters are:

1. Array Adapter
2. Base Adapter

Now we explain these two adapter in detail:

1.Array Adapter: Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.By default, ArrayAdapter expects a Layout with a single TextView, If you want to use more complex views means more customization in list items, please avoid ArrayAdapter and use custom adapters.

2.Base Adapter: BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView. Whenever you need a customized list you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying a custom list item. ArrayAdapter is also an implementation of BaseAdapter.

ListView attributes:

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.

Example:

```
<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" />
```

This sample defines a ListView with the ID list, whose items are populated by the string array fruits. Only one item is selectable from the ListView. No selector appears on the top of the ListView, and it also scrolls to the bottom when a new item is added to the data source.

For creating ListView controls, we can use either of the following methods:

1. The regular Activity base class
2. An activity that extends android.app.ListActivity

1. Creating a ListView with an Activity Base Class

In both cases, whether 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:

a. By ListView through a string resource
b. By ListView through Adapter

a. Populating ListViewThrough String Resource

To understand how a ListView is populated through string resources, let's create a new Android project called ListViewApp. In this application, we use two controls: ListView and TextView. ListView is populated through string resources to display a list of items for the user to select from. The item or option selected from the ListView is displayed via the TextView control.

Open the string resource file /res/values/strings.xml, and add a string-array structure called fruits that lists various fruits that we want to display via the ListView control.

```
<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 string resource app_name is the default string resource meant for displaying the application name in the title bar while running the application. The string-array fruits is the one that defines the array elements that we use to populate the ListView control.

activity_main.xml

```
<LinearLayoutxmlns: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.

MainActivity.java

```
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
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 TextViewselectedOpt=(TextView)findViewById(R.id.selectedopt);
      ListViewfruitsList = (ListView)findViewById(R.id.fruits_list);
      fruitsList.setOnItemClickListener(new OnItemClickListener(){
         @Override
         public void onItemClick(AdapterView<?> parent, View v, intposition,long id)
         {
            selectedOpt.setText("You have selected "+fruitsArray[position]);
         }
      });
   }
}
```

The string-array fruits from the resource file strings.xml is accessed and assigned to the string array fruitsArray. Similarly, the TextViewselectedopt and the ListViewfruits_list are captured from the layout file and assigned to the objects selectedOpt and fruitsList, respectively. To take an action when an item is selected from the ListView, the setOnItemClickListener() method is passed a new AdapterView.OnItemClickListener. This anonymous instance implements the onItemClick() callback method, which is executed when an item is selected from the ListView. In the onItemClick() method, we find the index location (position) of the selected item, use it to access the array element from the string array fruitsArray, and display it through the TextView object selectedOpt.

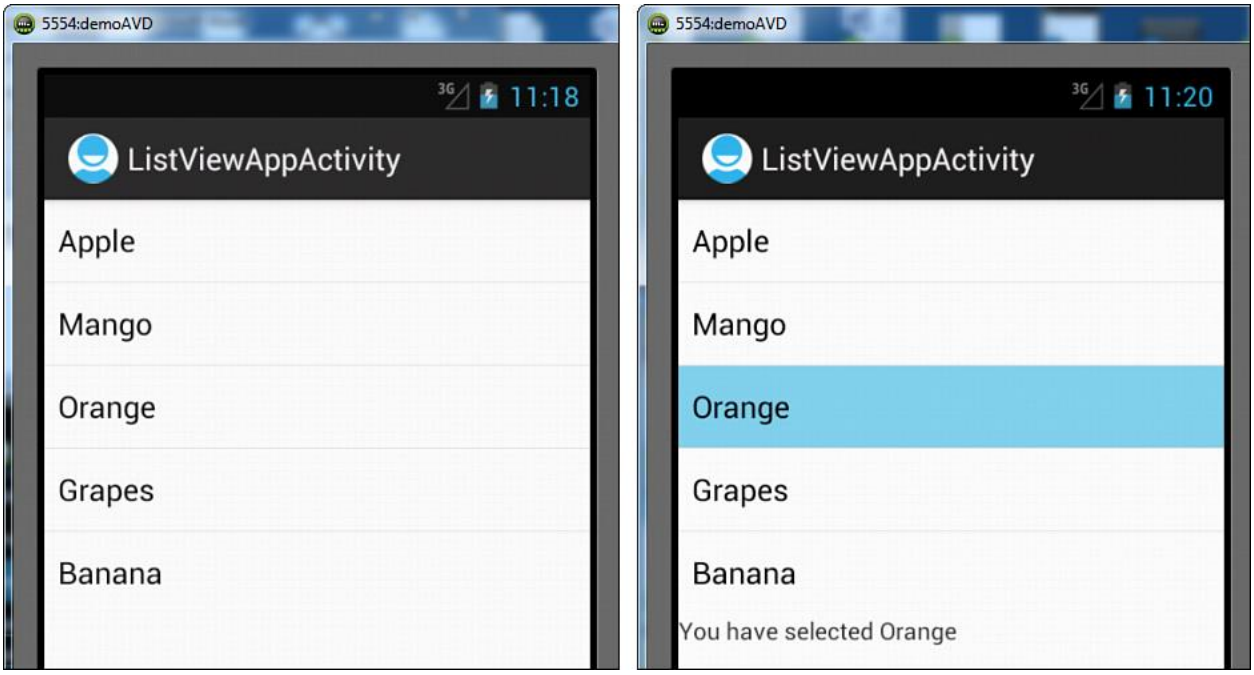


Figure Options displayed via the string array (string resource) in the ListView control (left) and the selected option from ListView (right) displayed via the TextView control

b.PopulatingListView Through the ArrayAdapter

The ArrayAdapter is one of the adapters provided by Android that provides data sources (child elements) to selection widgets and also casts the data into specific view(s) to be displayed inside the selection widgets. An ArrayAdapter can be created through string resource, as well as through string arrays defined in Java code.

Create a new Android project called ListViewDemo1. Again, in this application, we use two controls, ListView and TextView, where ListView displays the items assigned to it through ArrayAdapter and the TextView displays the item that is selected by the user from the ListView.

activity_main.xml.

```
<LinearLayoutxmlns: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:drawSelectorOnTop="false"/>
   <TextView
      android:id="@+id/selectedopt"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
</LinearLayout>
```

Next, we need to write code into the Java activity file MainActivity.java to serve the following purposes:
• Create an ArrayAdapter through a string array and assign it to the ListView for displaying items
• Display the item selected from the ListView in the TextView

MainActivity.java

```
import android.widget.ArrayAdapter;
import android.widget.AdapterView;
public class ListViewDemo1Activity extends Activity {
      public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_list_view_demo1);
       final String[] fruits={"Apple", "Mango", "Orange", "Grapes", "Banana"};
       final TextViewselectedOpt=(TextView)findViewById(R.id.selectedopt);
       ListViewfruitsList = (ListView)findViewById(R.id.fruits_list);
final ArrayAdapter<String>arrayAdpt= new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,
fruits);
       fruitsList.setAdapter(arrayAdpt);
       fruitsList.setOnItemClickListener(new OnItemClickListener(){
            public void onItemClick(AdapterView<?> parent, View v, intposition,long id){
             selectedOpt.setText("You have selected "+fruits[position]);
          }
       });              }              }
```

An ArrayAdapter is the simplest of the adapters and acts as the data source for the selection widgets ListView, GridView, and so on. An ArrayAdapter makes use of the TextView control to represent the child Views in a View.

ArrayAdapter<String>arrayadpt=new ArrayAdapter<String>
(this,android.R.layout.simple_list_item_1, fruits);

This constructor creates an ArrayAdapter called arrayAdpt that can display the elements of the specified array, fruits, via the TextView control.

The ArrayAdapter constructor consists of the following:

• **this (the current context)**: use the current instance as the context.

• **android.R.layout.simple_list_item_1:** Points to a TextView defined by the Android SDK that will be used for displaying each item in the ListView. The elements of the array that is specified next needs to be wrapped or cast in a view before being assigned to any selection widget for display. So, the android.R.layout.simple_list_item_1 simply turns the strings defined in the string array into a TextView for displaying them in a ListView.

• **array:**The data source—an array of strings for the ListView.

We can see that the ListView and TextView controls from the layout files are accessed and mapped to the objects fruitsList and selectedOpt, respectively. The arrayAdptArrayAdapter containing the elements of the fruits array in TextView form is assigned to the ListView control for displaying choices to the user. The OnItemClickListener interface is implemented via an anonymous class that implements a callback method, onItemClick(). The reference of an anonymous class is passed to the fruitsListListView to invoke the callback method onItemClick() when any of the items in ListView is clicked. In the onItemClick() method, the item selected in the ListView is displayed via the TextView control selectedOpt. When we run the application, the

list of items is displayed via ListView, and the item selected from the ListView is displayed via the TextView control.

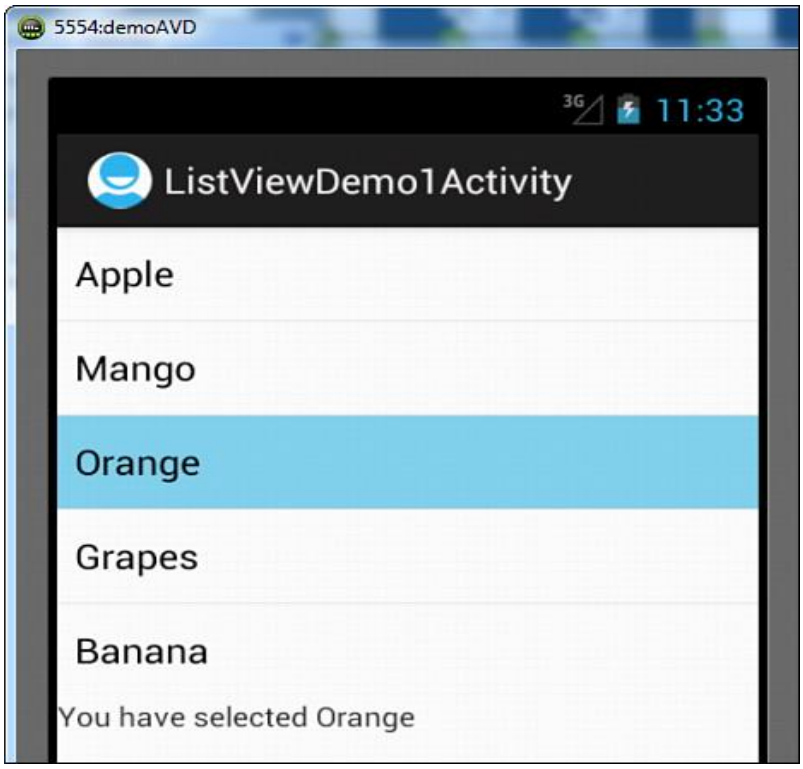


Figure: Options displayed through Java code in the ListView control and the selected option from ListView displayed through the TextView control

2. Creating ListView by Extending ListActivity

The ListActivity already contains a ListView, and we just need to populate it by calling the setListAdapter() method. If we just want to display a ListView and no other control in our application, then there is no need of defining any View in the layout file main.xml, as ListActivity automatically constructs a full-screen list for us. Also there is no need of using the onCreate() method for defining the content view of the activity for the simple reason that ListActivity already contains a ListView control. If we want to display some other controls along with ListView, we need to define the ListView and other desired controls in the layout file main.xml.

Let's create a new application to examine the creation of ListView by extending the ListActivity class.

activity_list_view_demo2.xml

```
<LinearLayoutxmlns: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="@android:id/list"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:drawSelectorOnTop="false"  />
```

```
        <TextView
            android:id="@+id/selectedopt"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </LinearLayout>
```

Note that the ID assigned to the ListView control defined in activity_list_view_demo2.xml must be @android:id/list; otherwise, ListActivity will not be able to identify it. To populate ListView and to display the item selected from it through the TextView control, write the code into ListViewDemo2Activity.java

```
import android.app.ListActivity;
import android.widget.ArrayAdapter;
public class ListViewDemo2Activity extends ListActivity {
   TextViewselectedOpt;
   String[] fruits={"Apple", "Mango", "Orange", "Grapes", "Banana"};
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_list_view_demo2);
      selectedOpt=(TextView)findViewById(R.id.selectedopt);
         ArrayAdapter<String>arrayAdpt = new
         ArrayAdapter<String>(this,android.R.layout.simple_list_item_single_choice,fruits);    #1
               getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);       #2
               setListAdapter(arrayAdpt);
            }

            @Override
            public void onListItemClick(ListView parent, View v, int position, long id)
            {
               super.onListItemClick(parent, v, position, id);
               selectedOpt.setText("You have selected "+fruits[position]);
            }
         }
```

We can see that an ArrayAdapter, arrayAdpt, is wrapping an array of strings. The ListView is populated by assigning arrayAdpt to it via the setListAdapter() method. The onListItemClick() method is executed when an item from the ListView is selected. In the onListItemClick() method, the item selected by the user from the ListView is displayed through the TextView selected-Opt. The simple_list_item_single_choice term in statement #1 and the ListView.CHOICE_MODE_SINGLE in statement #2 allow us to select a single item from the ListView. On running the application, we see that items in the ListView control appear in the form of a RadioButton control, allowing us to select only a single item at a time The chosen item is displayed through the TextView control.

Using the Spinner Control

In Android, Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list seen in other programming languages. In a default state, a spinner shows its currently selected value. It provides a easy way to select a value from a list of values.To populate the Spinnercontrol, we use two methods: one via the string resource and the other via the ArrayAdapter that acts as a data source.

Create a new Android project and name it as SpinnerApp. First, we populate the Spinner control via the string resource and then by ArrayAdapter.

Populating a SpinnerThrough Resources

we need to define the resource for displaying options in the Spinner control. We use a string-array to do this. add a new xml file to the res/values folder.

arrays.xml

```
<?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>
```

We can see that a string-array called fruits is defined, consisting of five elements, Apple, Mango, Orange, Grapes, and Banana. These array elements are used to display options in the Spinner control.

```
<LinearLayoutxmlns: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="Select a fruit"
      android:entries="@array/fruits"/>
   <TextView
      android:id="@+id/selectedopt"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
</LinearLayout>
```

We can see that a Spinner control and a TextView control are defined. The Spinner control displays a list of choices, and the TextView displays the choice selected by the user from the Spinner control. The prompt

attribute is a string that appears at the top of the Spinner control to guide the user. The choose_msg string resource, representing the string Choose a fruit is set to appear as a Spinner control prompt. The entries attribute is used to specify the data source to populate the Spinner control. We set the entries attribute to the string array fruits that we just defined in arrays.xml.

We want the item selected from the Spinner control by the user to appear in the TextView.

MainActivity.java

```
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 TextViewselectedOpt=(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("");
         }
      });
   }
}
```

The Spinner control is captured from the layout file and is mapped to the Spinner object spin. Similarly, the TextView is captured and mapped to the TextView object selectedOpt. The selectedOpt object is used to display the item selected from the Spinner control. An event listener, setOnItemSelectedListener, is attached to the Spinner control. When an item from the Spinner is selected, the onItemSelected() callback method that was specified in the anonymous class passed to the Spinner's setOnItemSelectedListener is called.

The onItemSelected() callback method retrieves the item that was selected in the Spinner and displays it through the TextView. The item selected in the Spinner is retrieved from the fruits array by specifying its index location—the position of the item selected from the Spinner control.

The onNothingSelected() method is implemented to make the selectedOptTextView blank; that is, it removes any previous message being displayed through TextView control that was selected earlier from the Spinner control.

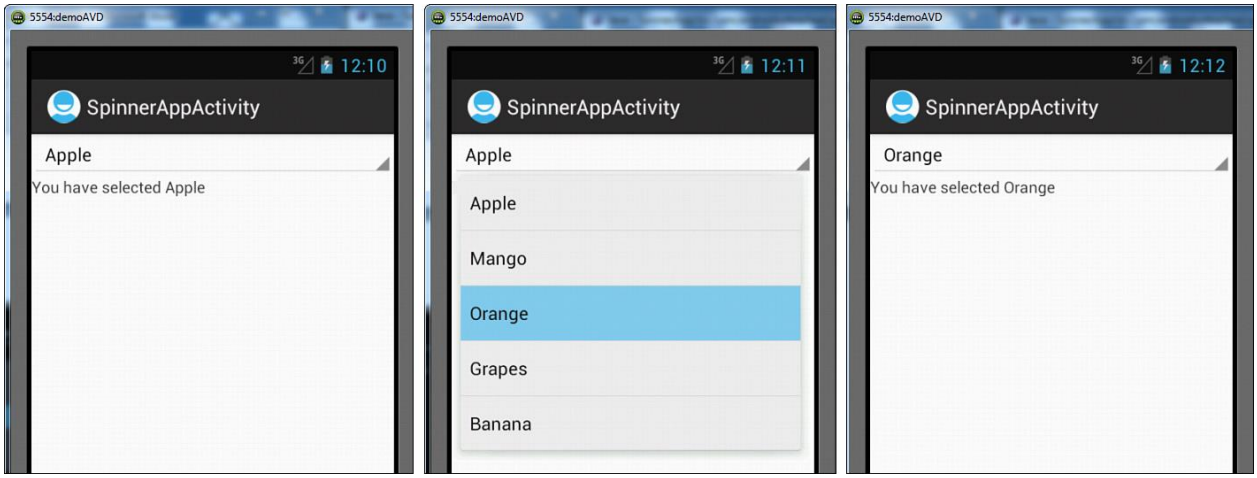


Figure:Spinner control with a drop-down arrow (left), options displayed through an Arrays Resource in the Spinner control (middle), and the option selected in the Spinner displayed through TextView (right)

Populating a SpinnerThroughArrayAdapter

Before populating the Spinner control with ArrayAdapter, we make it empty by removing the android:entries="@array/fruits" attribute from the XML definition of the Spinner control in the layout file main.xml. After we remove the entries attribute, the elements of the fruits string array no longer are displayed in the Spinner control.

```
<LinearLayoutxmlns: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="Select a fruit"/>
  <TextView
    android:id="@+id/selectedopt"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

There is no need to make any changes to strings.xml, the resource file, and we don't need the arrays.xml file either.

SpinnerAppActivity.java

```
package com.androidunleashed.spinnerapp;
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 TextViewselectedOpt=(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("");
         }
      });
   }
}
```

An ArrayAdapter<String> object called arrayAdpt is created from the string array fruits, and a standard simple_spinner_item view is used to display each bound element in the Spinner control. The arrayAdptArrayAdapter is assigned to the Spinner control for populating it. After we select an item in the Spinner control, the onItemSelected() callback method is executed to display the selected item through the TextView control.

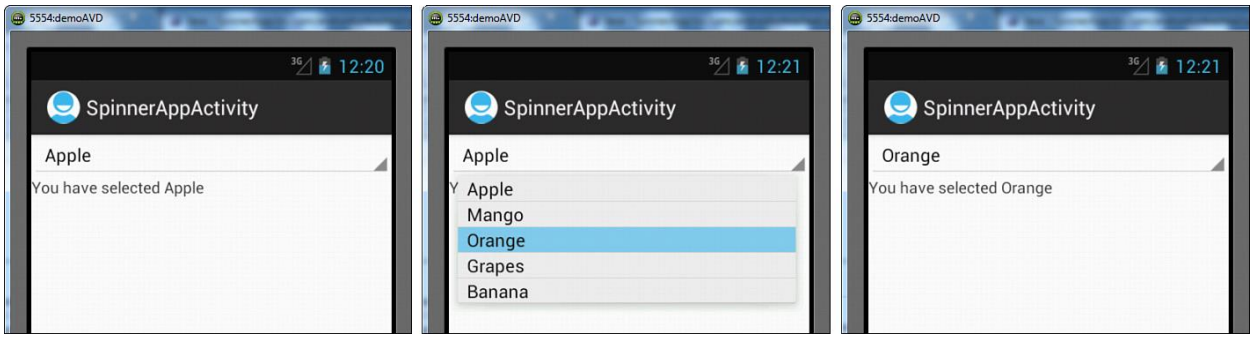


Figure: Spinner control with a drop-down arrow (left), options displayed in the Spinner control via Java code (middle), and the selected option displayed through the TextView (right)

Using the GridView Control

The GridView control is a ViewGroup used to display text and image data in the form of a rectangular, scrollable grid. 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.

GridView Attributes

The number of rows displayed through GridView is dependent on the number of elements supplied by the attached adapter. The size and number of columns is controlled through the following attributes:

• **android:numColumns:**Defines the number of columns. If we supply a value, auto_fit, Android computes the number of columns based on available space.

• **android:verticalSpacing and android:horizontalSpacing**: Define the amount of whitespace between the items in the grid.

• **android:columnWidth:** Defines the width of each column.

• **android:stretchMode:** The attribute determines whether the column can stretch or expand to take up the available space. The valid values for this attribute are

a. none—Does not allow columns to stretch or expand
b. columnWidth—Makes the columns take up all available space
c. spacingWidth—Makes the whitespace between columns take up all available space

Let's create a new Android project called GridViewApp. In this application, we display certain strings arranged in a rectangular grid. When a user selects any of the strings, its name is displayed. That is, we require two controls in this application: a GridView control for arranging strings and a TextView control for displaying the string selected by the user.

```
<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>
```

The GridView displays items or data in a rectangular grid, and TextView displays the item selected by the user from the GridView. The horizontal and vertical spacing among items in the GridView is set to 5dip and 2dip. The width of a column in GridView is set to 130dip. The number of columns in the GridView is determined by the number of columns of 130dip that can be accommodated in the available space. The columns are set to stretch to take up the available space, if any. The GridView appears at the center of the LinearLayout container.

```
package com.androidunleashed.gridviewapp;

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("");
   }
}
```

We access the TextView with the selectedopt ID from the layout and map it to the selectedOpt TextView object. We use the selectedOpt object to display the item selected by the user in the GridView. An array of strings called fruits is created. It is the strings in this array that we want to display in the GridView. We create an ArrayAdapter called arrayAdpt that makes the elements in the string array fruits appear in the TextView form. The ArrayAdapter is set to the GridView via the setAdapter() method to display its content via GridView. By attaching ItemClickListener to the GridView, we are assured that when any item displayed through GridView is clicked, the onItemClick() callback method is invoked. Through the onItemClick() method, we display the item selected by the user in the GridView via the TextView selectedOpt.

On running the application, we find that all items are displayed in the GridView,

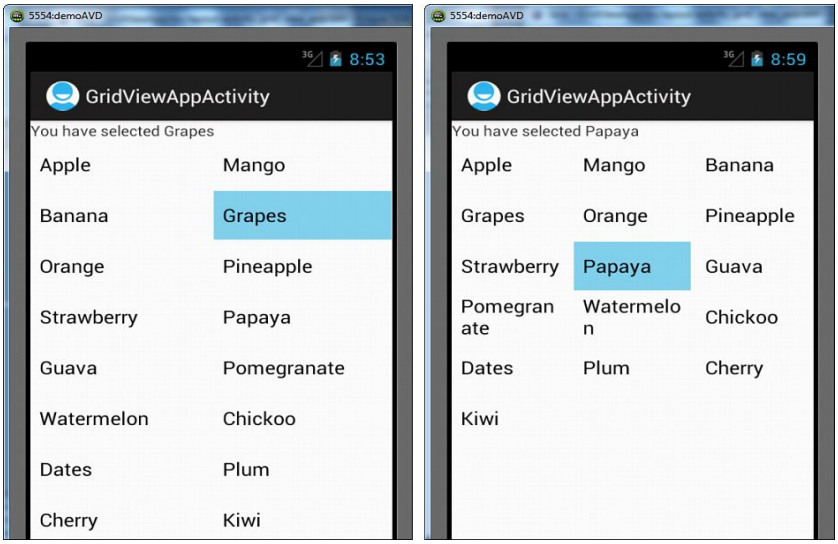


Figure: Items displayed in two columns in GridView (left), and items displayed in three columns in GridView (right)

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 behaviour. ViewPager in Android is a class that allows the user to flip left and right through pages of data. This class provides the functionality to flip pages in app. It is a widget found in the support library. It also has the ability to dynamically add and remove pages (or tabs) at anytime. ViewPager is most often used along with fragment which is convenient way to manage the life cycle of each page. ViewPager class works with PagerAdapter which provides pages.

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.

When you implement a PagerAdapter, you must override the following methods at minimum:

1. instantiateItem(ViewGroup, int): This method is used for creating and instantiating the page and adding it to the container. We inflate() our layout resource to create the hierarchy of view objects and then set resource for the ImageView in it. Finally, the inflated view is added to the container (which should be the ViewPager) and return it as well.

Syntax: public Object instantiateItem(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.

2.destroyItem(ViewGroup, int, Object): Removes the page from the container for the given position. We simply remove object using removeView() but could've also used removeViewAt() by passing it the position.

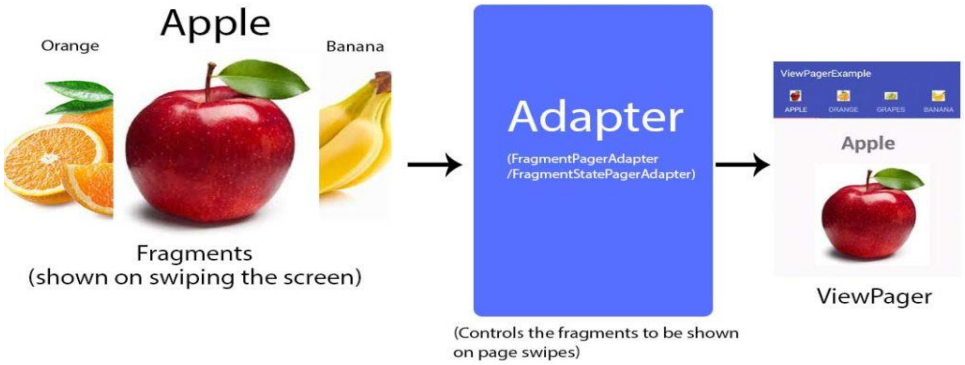
3. isViewFromObject(View viewT Orbject object): Determines whether the specified page is associated with a specific key object.

4. getCount(): Defines the size of the paging range, that is, the count of the number of the pages. The position of the pages is zero based by default; that is, the first page to the left is in position 0, the next page to the right is position 1, and so on. We can also set the initial position of the pager through the **setCurrentItem()** method.

To listen to the change in state of the selected page, we need to define a class that extends SimpleOnPageChangeListener. When a page from the ViewPager is selected, the callback method onPageSelected() is called.

There are three important steps to implement ViewPager:

1) A layout(that contains ViewPager) for the MainActivity.
2) FragmentPagerAdapter/FragmentStatePagerAdapter class which controls the fragments to be shown on page swipes.
3) Fragments to be shown on swiping the screen.

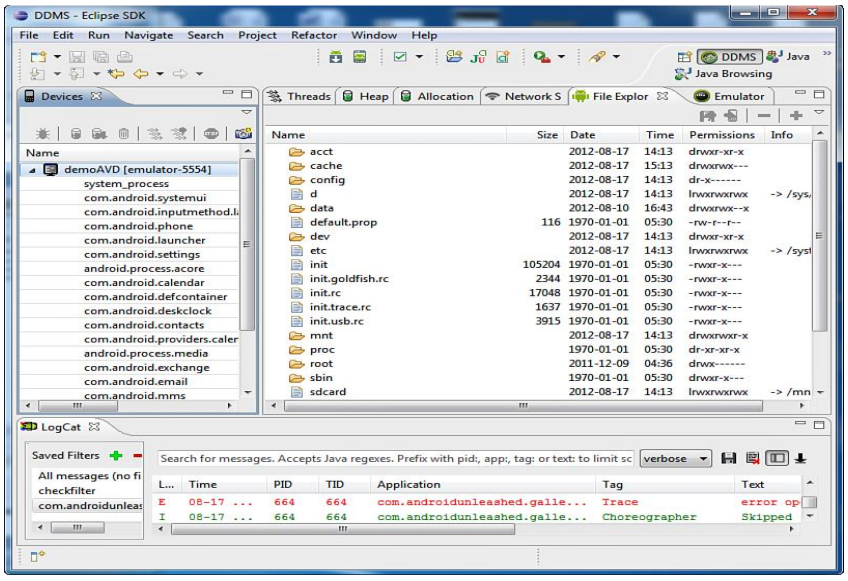


Using the Debugging Tool: Dalvik Debug Monitor Service (DDMS)

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. For example, by using the DDMS' LogCat feature, developers can view log messages regarding the state of the application and the device. LogCat can pinpoint the exact line number on which an error occurred. DDMS helps with a variety of tasks, including

- Finding bugs in applications running either on an emulator or on the physical device.
- Providing several services such as port forwarding, on-device screen capture, incoming call, SMS, and location data spoofing.
- Showing the status of active processes, viewing the stack and heap, viewing the status of active threads, and exploring the file system of any active emulator.
- Providing the logs generated by LogCat, so we can see log messages about the state of the application and the device. LogCat displays the line number on which the error(s) occurred.
- Simulating different types of networks, such as GPRS and EDGE.

Below figure the DDMS tool window.



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, along with the running AVDs (if any). The VMs associated with each device or AVD also is displayed. Selecting a VM displays its information in the right pane. In the Devices tab, you see some icons, described here:

• **Debug—**Used to debug the selected process.

• **Update Heap—**Enables heap information of the process. After clicking this icon, use the Heap icon on the right pane to get heap information.

• **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. After clicking this icon, we need to click the Threads icon in the right pane to display information about the threads that are created and destroyed in 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. Click the Start Method Profiling icon, interact with the application, and click the Stop Method Profiling icon to obtain information related to the different methods called in the application.

• **Stop Process—**Stops the selected process.

• **Screen Capture—**Captures our device/emulator screen. If the application is running and its output is being displayed through the device/emulator, clicking the Screen Capture icon displays the Device Screen Capture dialog box

DEBUGGING APPLICATIONS

The two most common ways of debugging an application and finding out what went wrong are

1. Placing breakpoints
2. Displaying log messages

Breakpoints are used to temporarily pause the execution of the application, allowing us to examine the content of variables and objects. 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. Let's open the Android project, HelloWorldApp, add a few statements to its activity file, as shown below. The statements just perform simple multiplication and display log messages.

```
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"T "a = " + a);
Log.v("CheckValue2"T "b = " + b);
Log.v("CheckValue3"T "c = " + c);
Log.i("InfoTag"T "Program is working correctly up till here");
```

```
Log.e("ErrorTag"T "Error--Some error has occurred here");
}        }
```

Let's place breakpoints at the following three statements in the activity file:

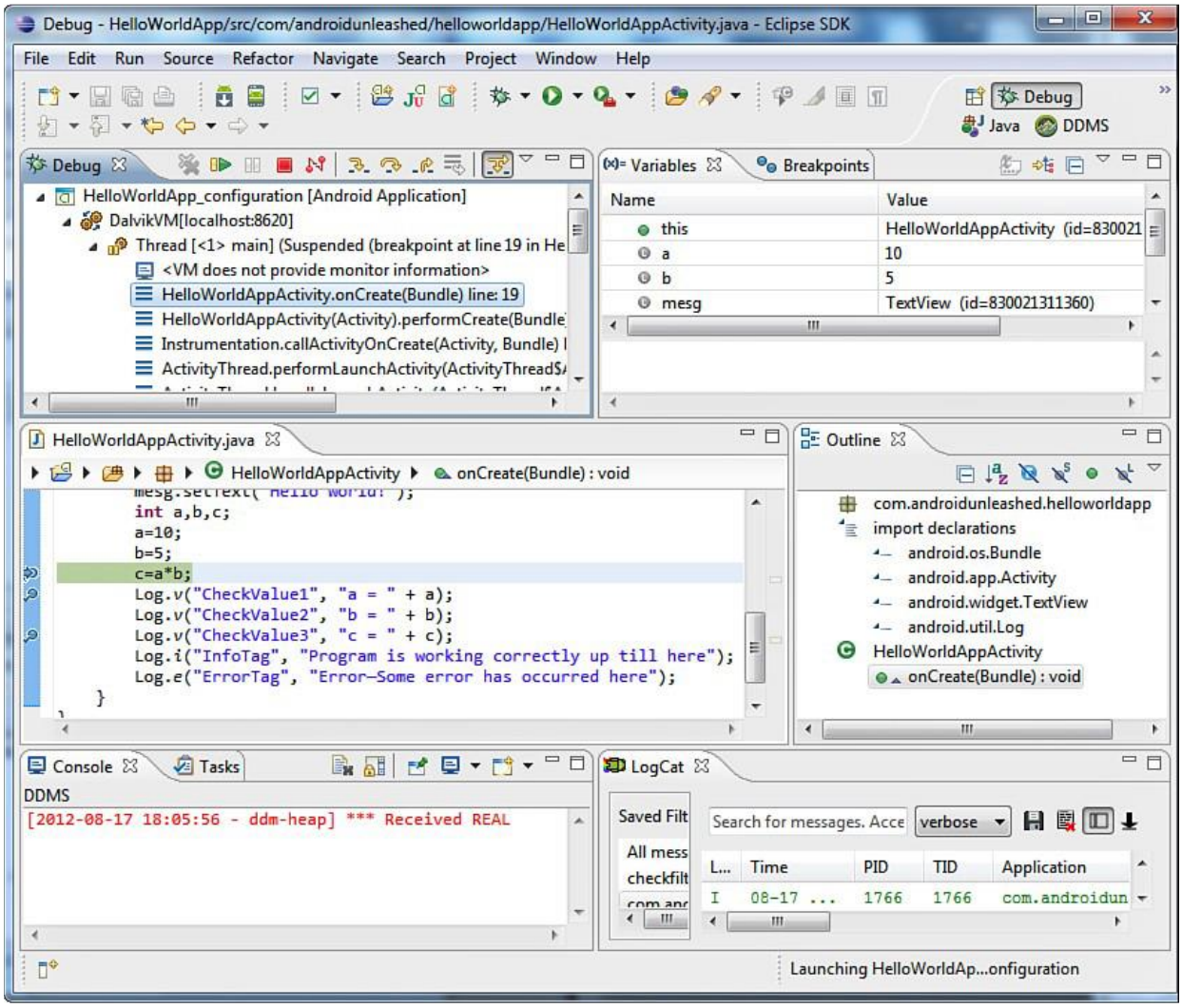
```
c=a*b;
Log.v("CheckValue1"T "a = " + a);
Log.v("CheckValue3"T "c = " + c);
```

When we place these breakpoints, a blue dot appears on the left, indicating that the breakpoints were successfully inserted

To stop execution at the breakpoints, don't run the application; instead debug it by either pressing F11, selecting Run, Debug, or right-clicking the project in Package Explorer and selecting Debug As, Android Application. During debugging, the application pauses when the first breakpoint is reached. At the breakpoints, we can highlight variables to see their values and execute certain expressions. When the application reaches a breakpoint for the first time, a window pops up asking whether we want to switch to the Debug perspective. To prevent this window from appearing again, check the Remember my decision check box and click Yes.

USING THE DEBUG PERSPECTIVE

When the application switches from the Java to the Debug perspective, you see the callback stack, console, code view, and variables, as shown in Figure below.



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. The Debug pane displays debug session information in a tree hierarchy
- **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. 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

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

WHAT ARE DIALOGS?

A dialog is a smaller window that pops up to interact with the user. It can display important messages and can even prompt for some data. Once the interaction with the dialog is over, the dialog disappears, allowing the user to continue with the application. We usually create a new activity or screen for interacting with users, but when we want only a little information, or want to display an essential message, dialogs are preferred. Dialogs are also used to guide users in providing requested information, confirming certain actions, and displaying warnings or error messages. The following is an outline of different dialog window types provided by the Android SDK:

- **Dialog:** The basic class for all dialog types.
- **AlertDialog**: An alert dialog box supports 0 to 3 buttons and a list of selectable elements, including check boxes and radio buttons. Among the other dialog boxes, the most suggested dialog box is the alert dialog box.
- **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. A dialog is created by creating an instance of the Dialog class.

The Dialog class creates a dialog in the form of a floating window containing messages and controls for user interaction. In Android, the dialogs are called asynchronously; that is, the dialogs are displayed and the main thread that invokes the dialogs returns and continues executing the rest of the application. The rest of the code continues to execute in the background and also allows users to simultaneously interact with the dialog. That means the dialogs in Android are modal in nature. If the dialog is open, users can interact only with the options and controls in the dialog until it is closed. While the user interacts with the dialog, the parent activity resumes its normal execution for efficiency. Each dialog window is defined within the activity where it will be used. A dialog window can be created once and displayed several times. It can also be updated dynamically.
The following is a list of the Activity class dialog methods:

1. **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.
2. **onCreateDialog()**—The callback method that executes when the dialog is created for the first time. It returns the dialog of the specified type.
3. **onPrepareDialog()**—The callback method used for updating a dialog.
4. **dismissDialog()**—Closes the dialog whose dialog identifier is supplied to this method. The dialog can be displayed again through the showDialog() method.
5. **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.

All these methods are deprecated, with the new preferred way being to use the DialogFragment with the FragmentManager. Older platforms should use the compatibility library to use DialogFragment and the FragmentManager.

The onCreateDialog() method is called only once while creating the dialog for the first time, whereas the onPrepareDialog() method is called each time the showDialog() method is called, allowing the activity to update the dialog before displaying it to the user. Basically, instead of creating new instances of a dialog each time, onCreateDialog() and onPrepareDialog() persist and manage dialog box instances. When these methods persist the state information within dialogs, any option selected or data entered in any of its text fields will remain and will be lost while displaying different dialog instances.

By overriding the onCreateDialog() method, we specify dialogs that will be created when showDialog() is called. Several dialog window types are available in the Android SDK, such as AlertDialog, DatePickerDialog, and TimePickerDialog, that we can readily use in an application. All the dialog windows are created by extending the Dialog class.

AlertDialog

An AlertDialog is a popular method of getting feedback from the user. This pop-up dialog remains there until closed by the user and hence is used for showing critical messages that need immediate attention or to get essential feedback before proceeding further. The simplest way to construct an AlertDialog is to use the static inner class AlertDialog.Builder that offers a series of methods to configure an AlertDialog. This example creates a new AlertDialog.Builder object called alertDialog:

AlertDialog.Builder alertDialog = new AlertDialog.Builder(this);

We can add a title, icon, and message to the alertDialog object that we want to display in the dialog. We can define buttons and controls for user interaction to display in the dialog. We can also register event listeners with the dialog buttons for handling events.

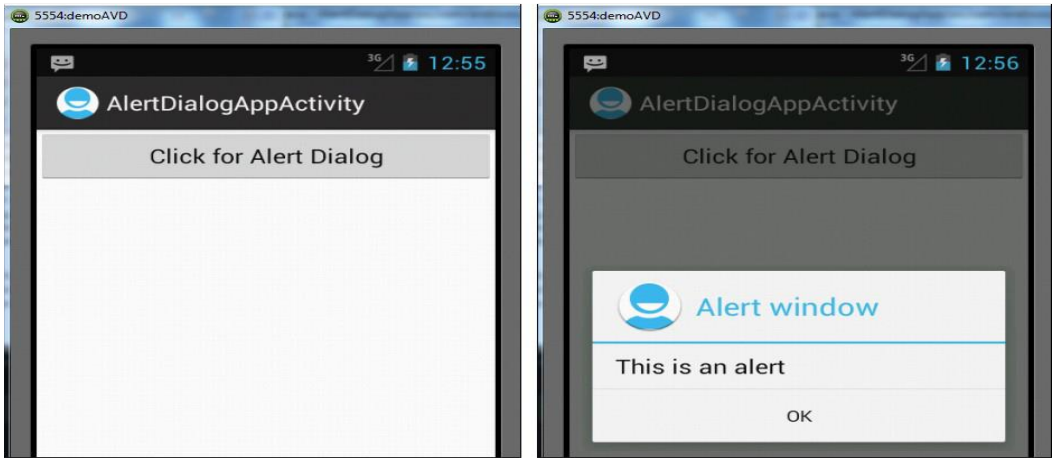
Let's create an Android application to see how AlertDialog is displayed. Name the project AlertDialogApp. In this application, we want to display a Button control that, when clicked, displays the AlertDialog. So, frst we need to defne a Button control in the layout

```
<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>
```

To display an AlertDialog, we use the AlertDialog.Builder subclass to create a Builderobject. Thereafter, we confgure the dialog with a title, message, and buttons with the Builder object.

```
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();
}
}
```

After running the application, we see a Button control with the caption Click for Alert Dialog, as shown in Figure(left). When we select the Button control, an AlertDialog is displayed with the title Alert window showing the message **This is an alert.**



Getting Input via the Dialog Box

We modify our current Android project AlertDialogApp to get input from the user. We make the following changes to the application:

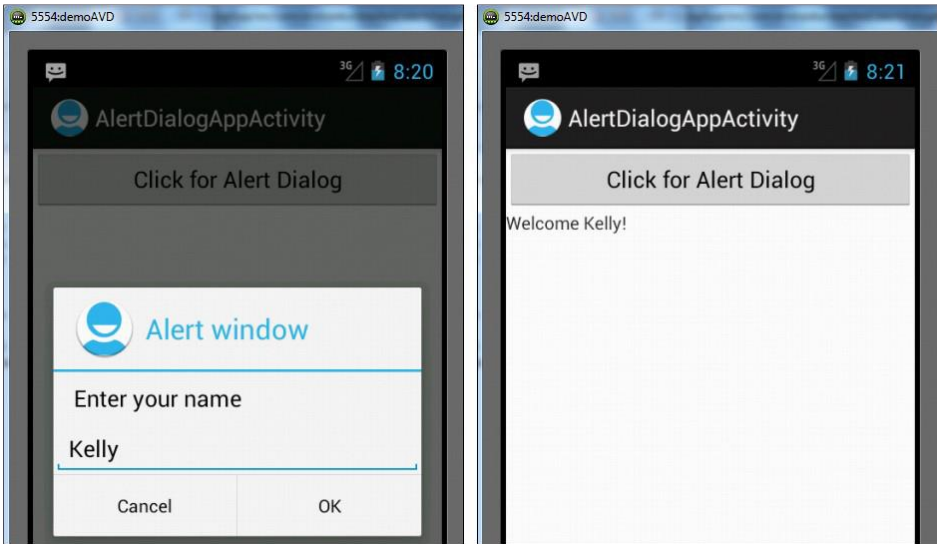
• Dynamically create an EditText control and set it as part of the AlertDialog to prompt the user for input.

• Add a TextView control to the layout file to display the data entered by the user in AlertDialog.

To make it more specific, our application asks the user to input a name through AlertDialog, and when the user selects the OK button after entering a name, a welcome message is displayed through the TextView control defined in the layout file. We also add a Cancel button to the AlertDialog, allowing the user to cancel the operation, which terminates the dialog. We don't have to worry about defining the EditText control in the layout file, as it will be created dynamically with Java code in the activity file. The only thing that we need to define in activity_main.xml is a TextView control that will be used for displaying a Welcome message on the screen.

```
import android.app.AlertDialog;
import android.content.DialogInterface;
public class AlertDialogAppActivity extends Activity   implements OnClickListener {
TextView resp;
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_alert_dialog_app);
        resp = (TextView)this.findViewById(R.id.response);
        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("Enter your name ");
fnal EditText username = new EditText(this);
alertDialog.setView(username);
alertDialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int buttonId) {
String str = username.getText().toString();
resp.setText("Welcome "+str+ "!");
return;
}
});
alertDialog.setNegativeButton("Cancel", new  DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int  buttonId) {
return;
}
});
alertDialog.show();
}
}
```



Selecting the Date and Time in One Application

To select the system date and time in an application we use two components:

1. DatePicker
2. TimePicker

DatePickerDialog:

In Android, DatePicker is a widget used to select a date. Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality android provides DatePicker and DatePickerDialog components. DatePickerDialog is used to see and modify the date. We can supply the day, month, and year values to its constructor to initialize the date initially displayed through this dialog.

```
<DatePicker
android:id="@+id/simpleDatePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"/>
```

Methods:

1. **setSpinnersShown(boolean shown):** This method is used to set whether the spinner of the date picker in shown or not. In this method you have to set a Boolean value either true or false. True indicates spinner is shown, false value indicates spinner is not shown. Default value for this function is true.
2. **getDayOfMonth():** This method is used to get the selected day of the month from a date picker. This method returns an integer value
3. **getMonth():** This method is used to get the selected month from a date picker. This method returns an integer value.
4. **getYear():** This method is used to get the selected year from a date picker. This method returns an integer value.
5. **getFirstDayOfWeek():** This method is used to get the first day of the week. This method returns an integer value

TimePicker

In Android, TimePicker is a widget used for selecting the time of the day in either AM/PM mode or 24 hours mode. The displayed time consist of hours, minutes and clock format. If we need to show this view as a Dialog then we have to use a TimePickerDialog class.

The TimePickerDialog allows us to set or select time through the built-in Android TimePickerview. We can set the values of hour and minute with values of hours ranging from 0 through 23 and minutes from 0 through 59.

Example

```
<TimePicker
android:id="@+id/simpleTimePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:timePickerMode="spinner"/>
```

Methods

1. **setCurrentHour(Integer currentHour):**This method is used to set the current hours in a time picker.
2. **setCurrentMinute(Integer currentMinute):** This method is used to set the current minutes in a time picker.
3. **getCurrentHour():** This method is used to get the current hours from a time picker.
4. **getCurrentMinute():** This method is used to get the current minutes from a time picker.
5. **setIs24HourView(Boolean is24HourView):** This method is used to set the mode of the Time picker either 24 hour mode or AM/PM mode. In this method we set a Boolean value either true or false. True value indicate 24 hour mode and false value indicate AM/PM mode.
6. **is24HourView():** This method is used to check the current mode of the time picker. This method returns true if its 24 hour mode or false if AM/PM mode is set

To see how the system date and time can be set in an application, let's create a new Android application and name it DateTimePickerApp. In this application, we use a TextView and two Button controls. The TextView control displays the current system date and time, and the two Button controls, Set Date and Set Time, are used to invoke the respective dialogs. When the Set Date button is selected, the DatePickerDialog is invoked, and when the Set Time button is selected, the TimePickerDialog is invoked.

```
<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>
```

We can see here that the TextView and the two Button controls are defined with the IDs datetimevw, date_button, and time_button, respectively. The captions for the two Buttoncontrols are Set Date and Set Time, respectively.

After defining the controls in the layout file, we write Java code into the DateTimePickerAppActivity.java activity file to perform the following tasks:

• Display the current system date and time in the TextView control.

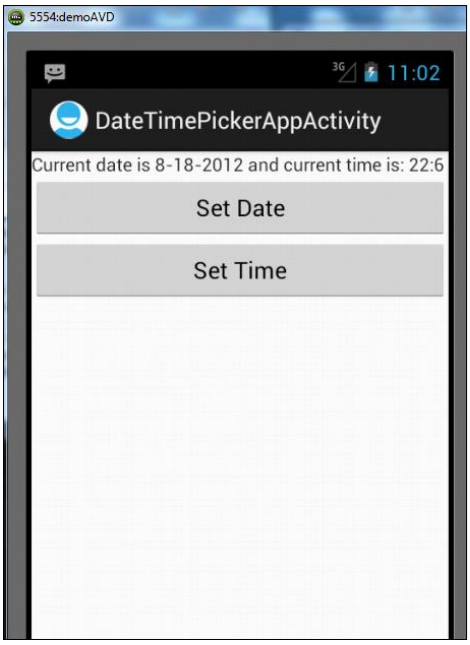
• Invoke DatePickerDialog and TimePickerDialog when the Set Date and Set Time Button controls are clicked.

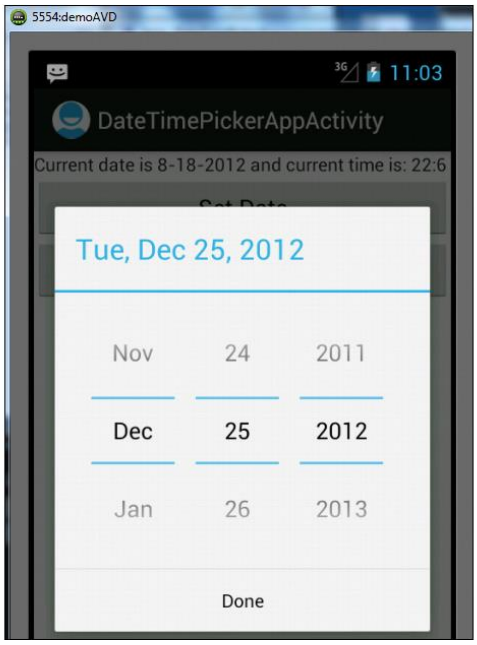
• Initialize DatePickerDialog and TimePickerDialog to display the current system date and time via the Calendar instance.

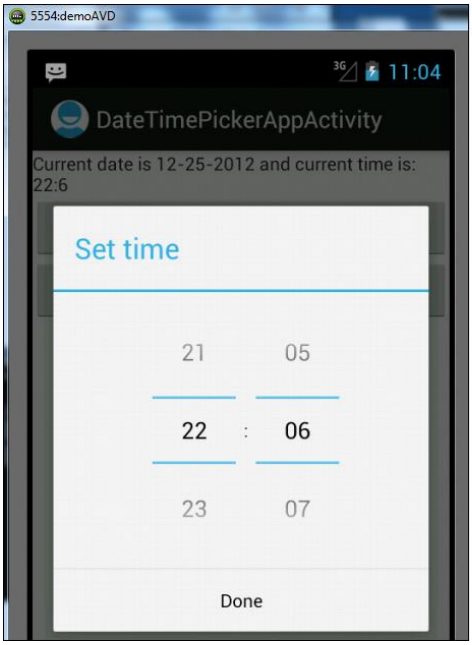
• Display the modifed date and time set by the user via the DatePickerDialog and TimePickerDialog through the TextView control.

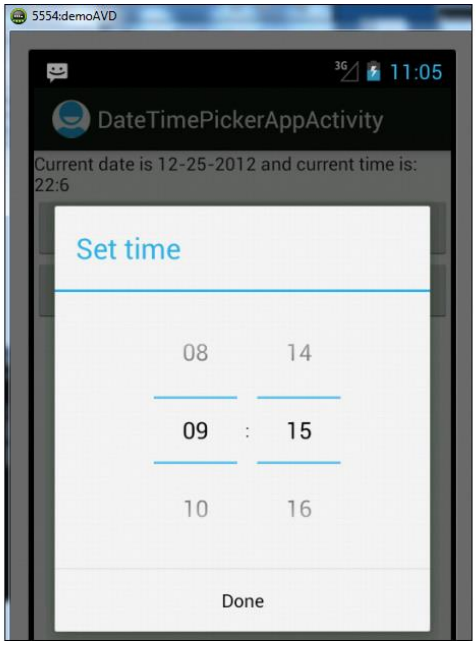
```
import java.util.Calendar;
import android.app.TimePickerDialog;
import android.app.DatePickerDialog;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.TimePicker;
import android.widget.DatePicker;
public class DateTimePickerAppActivity extends Activity
{
private 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) {
new TimePickerDialog(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);
}
};
}
```









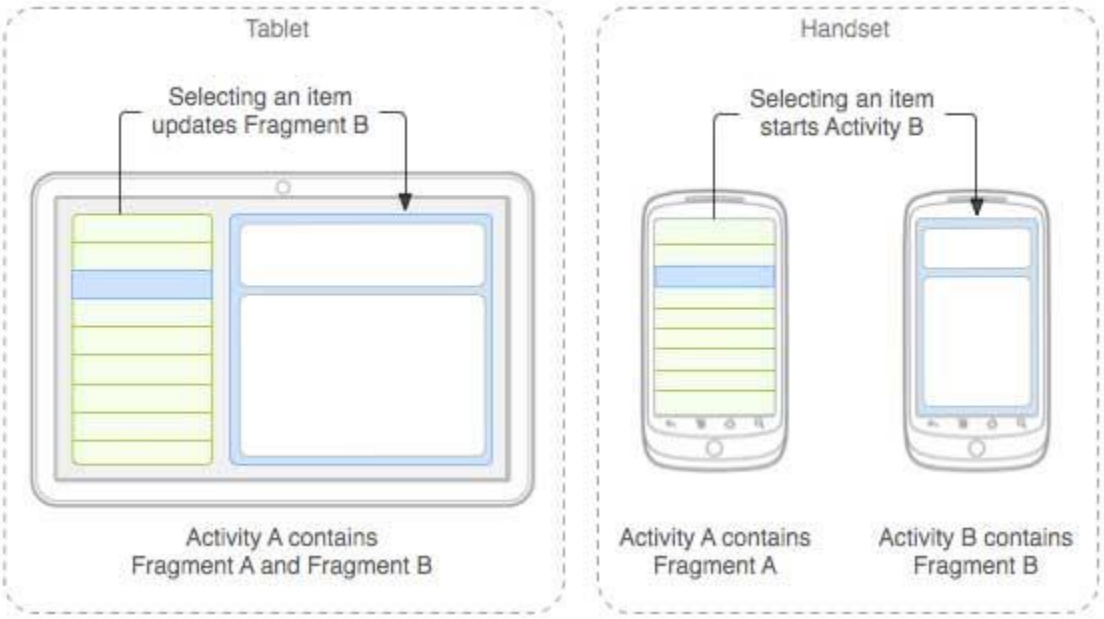
FRAGMENTS

The size of the screen changes when a device is oriented from portrait to landscape mode. In landscape mode, the screen becomes wider and shows empty space on the right. The height becomes smaller and hides the controls on the bottom of the display. There is a difference in screen sizes between the Android phone and Android tablet, as well. Android tablets have a 7–10 inch display, whereas Android phones are in the range of 3–5 inches. When developing an application, we need to arrange Views in such a way that the user can view everything in both landscape and portrait mode. If we don't organize the Views with this in mind, problems arise if the user switches modes while running an application. One solution to this problem is one we have already seen—designing an individual layout for each device or screen mode. This solution is time consuming. Another solution is implementing fragments in the application.

The Structure of a Fragment

In Android, Fragment is a part of an activity which enables more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents a behavior 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. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle.

One or more fragments can be embedded in the activity to fll up the blank space that appears on the right when switching from portrait to landscape. Similarly, the fragments can be dynamically removed if the screen size is unable to accommodate the Views. That is, the fragments make it possible for us to manage the Views depending on the target device.



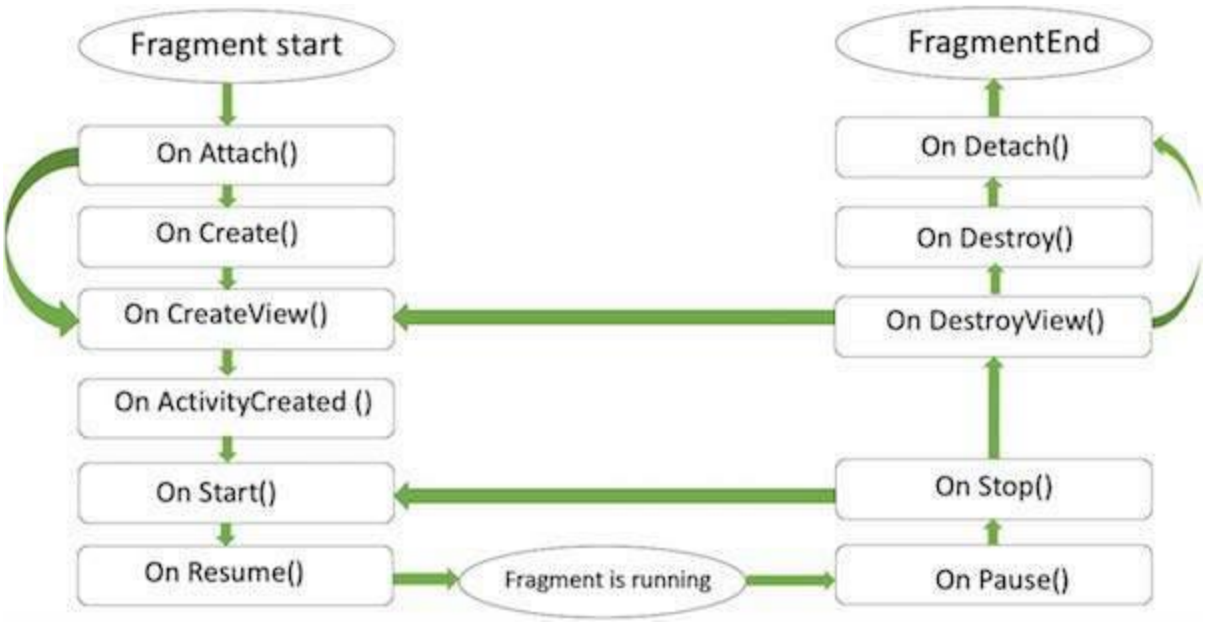
A fragment is like a subactivity with its own life cycle and view hierarchy. We can add or remove fragments while the activity is running. Remember that the fragments exist within the context of an activity, and so cannot be used without one. To create a fragment, we need to extend the Fragment class and implement several life cycle callback methods, similar to an activity.

Some Important Points About Fragment In Android:

1. Fragments were added in Honeycomb version of Android i.e API version 11.
2. We can add, replace or remove Fragment's in an Activity while the activity is running. For performing these operations we need a Layout(Relative Layout, FrameLayout or any other layout) in xml file and then replace that layout with the required Fragment.
3. Fragments has its own layout and its own behaviour with its own life cycle callbacks.
4. Fragment can be used in multiple activities.
5. We can also combine multiple Fragments in a single activity to build a multi-plane UI.

The Life Cycle of a Fragment

The life cycle of a fragment is affected by the activity's life cycle in which it is embedded. That is, when the activity is paused, all the fragments in it are paused. Similarly, if an activity is destroyed, all of its fragments are destroyed, as well.



The life cycle of a fragment includes several callback methods

1. **onAttach()**—Called when the fragment is attached to the activity.
2. **onCreate()**—Called when creating the fragment. The method is used to initialize the items of the fragment that we want to retain when the fragment is resumed after it is paused or stopped. For example, a fragment can save the state into a Bundle object that the activity can use in the onCreate() callback while re-creating the fragment.
3. **onCreateView()**—Called to create the view for the fragment.
4. **onActivityCreated()**—Called when the activity's onCreate() method is returned.
5. **onStart()**—Called when the fragment is visible to the user. This method is associated with the activity's onStart().
6. **onResume()**—Called when the fragment is visible and is running. The method is associated with the activity's onResume().
7. **onPause()**—Called when the fragment is visible but does not have focus. The method is attached to the activity's onPause().
8. **onStop()**—Called when fragment is not visible. The method is associated with the activity's onStop().
9. **onDestroyView()**—Called when the fragment is supposed to be saved or destroyed. The view hierarchy is removed from the fragment.
10. **onDestroy()**—Called when the fragment is no longer in use. No view hierarchy is associated with the fragment, but the fragment is still attached to the activity.
11. **onDetach()**—Called when the fragment is detached from the activity and resources allocated to the fragment are released.

A fragment also has a bundle associated with it that serves as its initialization arguments. Like an activity, a fragment can be saved and later automatically restored by the system.

Example

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context="example.javatpoint.com.fragmentexample.MainActivity">
    <fragment
      android:id="@+id/fragment1"
      android:name="example.javatpoint.com.fragmentexample.Fragment1"
      android:layout_width="0px"
      android:layout_height="match_parent"
      android:layout_weight="1"   />
   <fragment
      android:id="@+id/fragment2"
      android:name="example.javatpoint.com.fragmentexample.Fragment2"
      android:layout_width="0px"
      android:layout_height="match_parent"
      android:layout_weight="1"   />
  </LinearLayout>
```

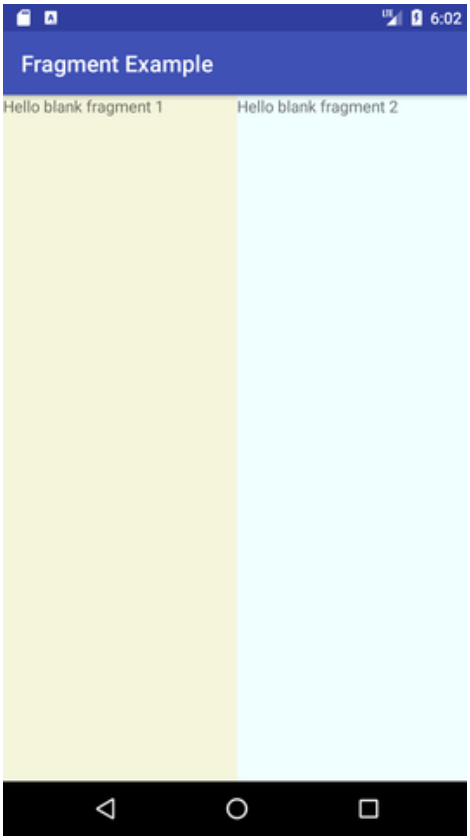
fragment_fragment1.xml

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

fragment_fragment2.xml

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

MainActivity.java




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

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

Fragment1.java

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment1 extends Fragment {
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
    }
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle saved) {
      // Inflate the layout for this fragment
      return inflater.inflate(R.layout.fragment_fragment1, container, false);
    }
  }
```

Fragment2.java

```
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
public class Fragment2 extends Fragment {
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
   }
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle saved) {
     // Inflate the layout for this fragment
     return inflater.inflate(R.layout.fragment_fragment2, container, false);
   }          }
```

CREATING FRAGMENTS WITH JAVA CODE

Until now, we have been defning fragments statically by using <fragment> elements in the layout file of the application. For creating, adding, and replacing fragments to an activity dynamically, we use the FragmentManager.

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();
```

Here the Fragment1Activity is the Java class of the fragment, which is also used to load the UI of the fragment from its XML file. We assume that the fragment_container is the ID of the container that exists in the layout file where we want to put our fragment. Usually LinearLayout or FrameLayout is used as the fragment_container. The TAG1 refers to the unique ID to identify and access the fragment. The commit() method is used to apply the changes.

CREATING SPECIAL FRAGMENTS

After understanding the procedure to create simple fragments, we learn to create specialized fragments such as list fragments, dialog fragments, and preference fragments. To create these, we extend from the following subclasses of the Fragment base class:

• **ListFragment:** The basic implementation of list fragment is for creating list of items in fragments. A ListFragment is a fragment that contains a built-in ListView that can be set to display items from a specified data source. The data source can be an array or a cursor.

• **DialogFragment:** This fragment displays a dialog on top of its owner activity. A fragment that displays a dialog window, floating on top of its activity's window. This fragment contains a Dialog object, which it displays as appropriate based on the fragment's state. Control of the dialog (deciding when to show, hide, dismiss it) should be done through the API here, not with direct calls on the dialog.

• **PreferenceFragment:** In Android apps, there are often settings pages that contain different options the user can tweak. In Android apps, there are often settings pages that contain different options the user can tweak. PreferenceFragment is a fragment that enables users to confgure and personalize an application. The PreferenceFragment can contain several Preference Views that help in uniformly setting application preferences with minimum effort. The PreferenceFragment and PreferenceFragmentCompat contain a hierarchy of preference objects displayed on screen in a list. These preferences will automatically save to SharedPreferences as the user interacts with them.

UNIT-IV

Short Answer Questions

1. What is an adapter? Why it is used in android?
2. Define ListView. Explain its attributes
3. What are the different ways to load data in to listview, gridview and spinner
4. Explain the use of spinner control and gridview
5. Write about gridview attributes
6. Explain about viewpager control
7. What is DDMS. What are the ways to debug an application?
8. Define Dialog. List different types of dialogs
9. List the activity class dialog methods
10. Write about DatePicker and TimePicker controls
11. Define fragment. What is the use of it
12. Draw the life cycle of Fragment
13. Write about special fragments

(Long Answer Questions)

1. Explain in detail about ListView control with an example program
2. What is an adapter? Explain about different types of adapters
3. Describe about spinner with a program
4. What is a GridView. Explain about it with an example program
5. Explain how to debug an application with dalvik debug
6. What are dialogs? Explain in detail about different dialogs
7. Develop an application for Selecting the Date and Time in One Application
8. What are fragments, explain their importance while designing an activity
9. Describe about fragment lifecycle and special fragments