Activities, Views and Layouts

By

Srikanth Pragada

- **MCTS for .NET 4.0(Web Applications)**
- **SCJP 5.0**
- **SCWCD 1.4**
- **SCBCD 5.0**
- **Oracle Certified SQL Expert**

www.srikanthtechnologies.com

304, 2nd Floor,
Eswar Paradise,
Dwarkanagar Main Road,
Vizag – 16.

Phone : 2541948, 9059057000

Activity

- An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
- Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.
- An application usually consists of multiple activities that are loosely bound to each other.
- Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.
- Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack").
- When a new activity starts, it is pushed onto the top of back stack and takes user focus.
- When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods.
- There are several callback methods that an activity might receive, due to a change in its state.
- You must declare your activity in the manifest file in order for it to be accessible to the system
- An <**activity**> element can also specify various intent filters—using the <**intent-filter**> element—in order to declare how other application components may activate it.

Creating an Activity

- To create an activity, you must create a subclass of **Activity**.
- This subclass needs to implement callback methods that the system calls when the activity transitions between various states of its lifecycle
- The user interface for an activity is provided by a hierarchy of **views**.
- Each view controls a particular rectangular space within the activity's window and can respond to user interaction. For example, a view might be a button that initiates an action when the user touches it.
- Widgets are views that provide a visual (and interactive) elements for the screen, such as a button, text field, checkbox, or just an image.
- Layouts are views derived from **ViewGroup** that provide a unique layout model for its child views, such as a linear layout, a grid layout, or relative layout.
- The most common way to define a layout using views is with an XML layout file saved in your application resources. This way, you can maintain the design of your user interface separately from the source code that defines the activity's behavior.
- You can set the layout as the UI for your activity with **setContentView**(), passing the resource ID for the layout.
- You must declare your activity in the manifest file using <**activity**> element, in order for it to be accessible to the system.

Main Activity

- Main activity is the one that starts when you start your application.
- It is declared in manifest file like any other activity, but has an intent-filter that specifies it should respond to main action and should be placed in launcher category.

```
<application ... >
    <activity
        android:name=".HelloActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Programmatically Starting an Activity

- You can start another activity by calling **startActivity**(), passing it an Intent that describes the activity you want to start.
- An intent can also carry small amounts of data to be used by the activity that is started.

```
Intent intent = new Intent(this, InputActivity.class);
startActivity(intent);
```

Passing data between Activities

Data can be sent from one activity to another using either through extras or a bundle, which contains a collection of keys, using Intent.

The following example shows how **FirstActivity** sends name to **SecondActivity** using extras in Intent.

FirstActivity.java

```
01: package com.st.first;
02: import android.app.Activity;
03: import android.content.Intent;
04: import android.os.Bundle;
05: import android.view.View;
06: import android.widget.EditText;
07: public class FirstActivity extends Activity {
08:   @Override
09:   protected void onCreate(Bundle savedInstanceState) {
10:         super.onCreate(savedInstanceState);
11:         setContentView(R.layout.activity_first);
12:   }
13:   public void invokeSecond(View v) {
14:         EditText  editName = (EditText) findViewById(R.id.editName);
15:
16:         Intent intent  = new Intent(this, SecondActivity.class);
17:         intent.putExtra("name", editName.getText().toString());
18:         startActivity(intent);
19:   }
20: }
```

activity_first.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".FirstActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter your name : " />
    <EditText
        android:id="@+id/editName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" >
        <requestFocus />
    </EditText>
    <Button
        android:id="@+id/button1"
        android:onClick="invokeSecond"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ok" />
</LinearLayout>
```

SecondActivity.java

```
01: package com.st.first;
02: import android.app.Activity;
03: import android.os.Bundle;
04: import android.widget.TextView;
```

```
05:
06: public class SecondActivity extends Activity {
07:   @Override
08:   protected void onCreate(Bundle savedInstanceState) {
09:         super.onCreate(savedInstanceState);
10:         setContentView(R.layout.activity_second);
11:         TextView textName = (TextView)  findViewById( R.id.textName);
12:         String name = this.getIntent().getStringExtra("name");
13:         textName.setText(name);
14:      }
15: }
```

activity_second.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity" >
    <TextView
        android:id="@+id/textName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />
</RelativeLayout>
```

The above can also be done using Bundle, which contains data in the form of keys.

FirstActivity.java

```
Intent intent  = new Intent(this, SecondActivity.class);
Bundle bundle = new Bundle();
bundle.putString("name",editName.getText().toString());
intent.putExtras(bundle);
startActivity(intent);
```

SecondActivity.java

```
Bundle bundle = this.getIntent().getExtras();
textName.setText( bundle.getString("name"));
```

Starting an activity for a result

- Sometimes, you might want to receive a result from the activity that you start. In that case, start the activity by calling **startActivityForResult**()
- The caller must implement **onActivityResult**() callback method.
- When the called activity is done, it returns a result in an Intent to your **onActivityResult**() method.

CallerActivity.java

```
01: import android.app.Activity;
02: import android.content.Intent;
03: import android.os.Bundle;
04: import android.view.View;
05: import android.widget.TextView;
06:
07: public class CallerActivity extends Activity {
08:   @Override
09:   protected void onCreate(Bundle savedInstanceState) {
10:         super.onCreate(savedInstanceState);
11:         setContentView(R.layout.activity_caller);
12:   }
13:     public void getMessage(View v) {
14:       Intent intent = new Intent(this,CalledActivity.class);
15:       startActivityForResult(intent,1);
16:   }
17:   @Override
```

```
18:   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
19:          if ( requestCode == 1 &&  resultCode == 1) {
20:            String message = data.getStringExtra("message");
21:            TextView textMessage = (TextView) findViewById(R.id.textMessage);
22:            textMessage.setText(message);
23:          }
24:   }
25: }
```

activity_caller.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".CallerActivity" >
    <TextView
        android:id="@+id/textMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="getMessage"
        android:text="Get Message" />
</LinearLayout>
```

CalledActivity.java

```
01: import android.app.Activity;
02: import android.content.Intent;
03: import android.os.Bundle;
04: import android.view.View;
05: import android.widget.EditText;
06: public class CalledActivity extends Activity {
07:   @Override
08:   protected void onCreate(Bundle savedInstanceState) {
09:         super.onCreate(savedInstanceState);
10:         setContentView(R.layout.activity_called);
11:   }
12:   public void sendMessage(View v) {
13:         EditText editMessage  = (EditText) findViewById(R.id.editMessage);
14:         Intent intent = getIntent();
15:         intent.putExtra("message", editMessage.getText().toString());
16:         this.setResult(1,intent);
17:         this.finish();  // terminate activity
18:   }
19: }
```

activity_called.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".CalledActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter Message : " />
    <EditText
        android:id="@+id/editMessage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10" >

        <requestFocus />
    </EditText>
    <Button
        android:id="@+id/button1"
        android:onClick="sendMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ok" />

</LinearLayout>
```

Shutting Down an Activity

- You can shut down an activity by calling its **finish**() method.
- You can also shut down a separate activity that you previously started by calling **finishActivity**().

***Note**: In most cases, you should not explicitly finish an activity using these methods. The Android system manages the life of an activity for you, so you do not need to finish your own activities. Calling these methods could adversely affect the expected user experience and should only be used when you absolutely do not want the user to return to this instance of the activity.*

Activity Lifecycle

Managing the lifecycle of your activities by implementing callback methods is crucial to developing a strong and flexible application.

Resumed	The activity is in the foreground of the screen and has user focus. (This state is also sometimes referred to as "running".)
Paused	Another activity is in the foreground and has focus, but this one is still visible. That is, another activity is visible on top of this one and that activity is partially transparent or doesn't cover the entire screen. A paused activity is completely alive, but can be killed by the system in extremely low memory situations.
Stopped	The activity is completely obscured by another activity (the activity is now in the "background"). A stopped activity is also still alive but not attached to Window Manager.

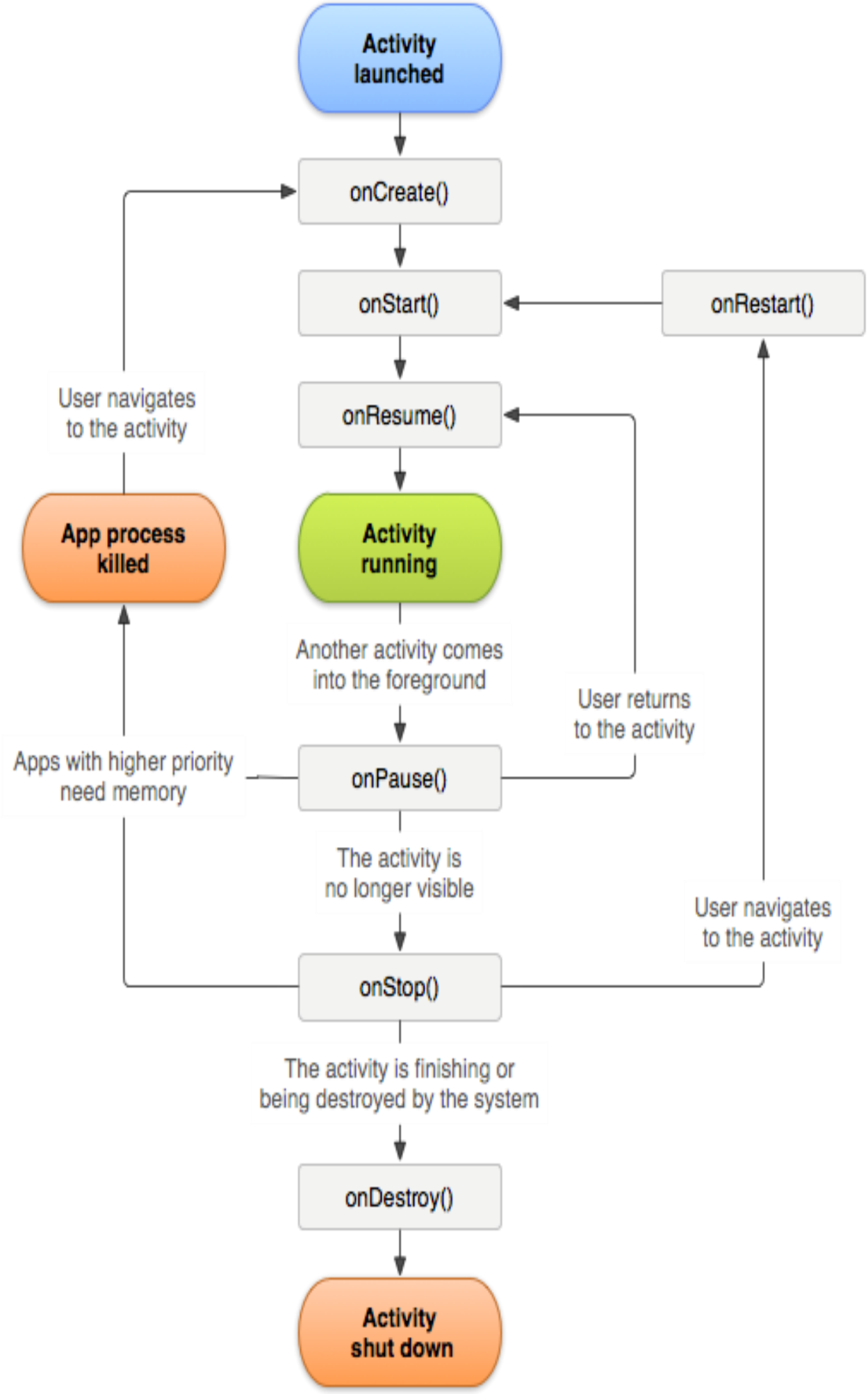
Implementing the lifecycle callbacks

When an activity transitions into and out of the different states described above, it is notified through various callback methods:

```
01: public class ExampleActivity extends Activity {
02:     @Override
03:     public void onCreate(Bundle savedInstanceState) {
04:         super.onCreate(savedInstanceState);
05:         // The activity is being created.
06:     }
07:     @Override
08:     protected void onStart() {
```

```
09:         super.onStart();
10:         // The activity is about to become visible.
11:     }
12:     @Override
13:     protected void onResume() {
14:         super.onResume();
15:         // The activity has become visible (it is now "resumed").
16:     }
17:     @Override
18:     protected void onPause() {
19:         super.onPause();
20:         // Another activity is taking focus (this activity is about to be "paused").
21:     }
22:     @Override
23:     protected void onStop() {
24:         super.onStop();
25:         // The activity is no longer visible (it is now "stopped")
26:     }
27:     @Override
28:     protected void onDestroy() {
29:         super.onDestroy();
30:         // The activity is about to be destroyed.
31:     }
32: }
```

***Note**: Your implementation of these lifecycle methods must always call the superclass implementation before doing any work, as shown in the examples above.*


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Method	Description	Killable after?	Next
onCreate()	Called when the activity is first created. Used for static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured. Always followed by onStart().	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again. Always followed by onStart()	No	onStart()
onStart()	Called just before the activity becomes visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by onPause().	No	onPause()
onPause()	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.	Yes	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it. Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.	Yes	onRestart() or onDestroy()
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called finish() on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.	Yes	nothing

***Note**: **onPause**() is the first of the three methods after which an activity can be killed, so use onPause() to write crucial persistent data (such as user edits) to storage.*

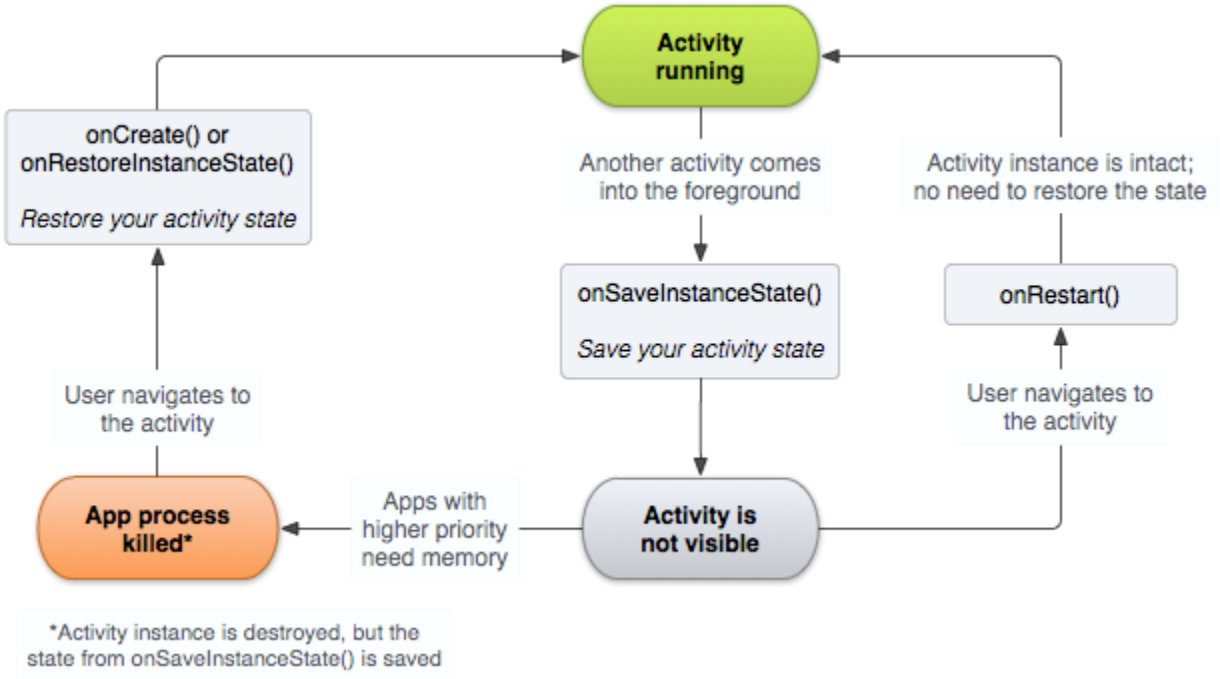
Order or events

The order of lifecycle callbacks is well defined, particularly when the two activities are in the same process and one is starting the other. Here's the order of operations that occur when Activity A starts Acivity B:

1. Activity A's **onPause**() method executes.
2. Activity B's **onCreate**(), **onStart**(), and **onResume**() methods execute in sequence. (Activity B now has user focus.)
3. Then, if Activity A is no longer visible on screen, its **onStop**() method executes.

Saving activity state

- When the system destroys an activity in order to recover memory, the Activity object is destroyed, so the system cannot simply resume it with its state intact.
- You must ensure that important information about the activity state is preserved by implementing an additional callback method that allows you to save information about the state of your activity and then restore it when the system recreates the activity.
- The callback method in which you can save information about the current state of your activity is **onSaveInstanceState**().
- The system calls this method before making the activity vulnerable to being destroyed and passes it a Bundle object.
- If the system kills your application process and the user navigates back to your activity, the system recreates the activity and passes the Bundle to both **onCreate**() and **onRestoreInstanceState**().
- If there is no state information to restore, then the Bundle passed is null.
- However, even if you do not implement **onSaveInstanceState**(), some of the activity state is restored by the Activity class's default implementation of **onSaveInstanceState**().
- Specifically, the default implementation **calls onSaveInstanceState**() for every View in the layout, which allows each view to provide information about itself that should be saved.
- Almost every widget in the Android framework implements this method as appropriate, such that any visible changes to the UI are automatically saved and restored when your activity is recreated.



***Note**: A good way to test your application's ability to restore its state is to simply rotate the device so that the screen orientation changes. When the screen orientation changes, the system destroys and recreates the activity in order to apply alternative resources that might be available for the new screen configuration. For this reason alone, it's very important that your activity completely restores its state when it is recreated, because users regularly rotate the screen while using applications.*

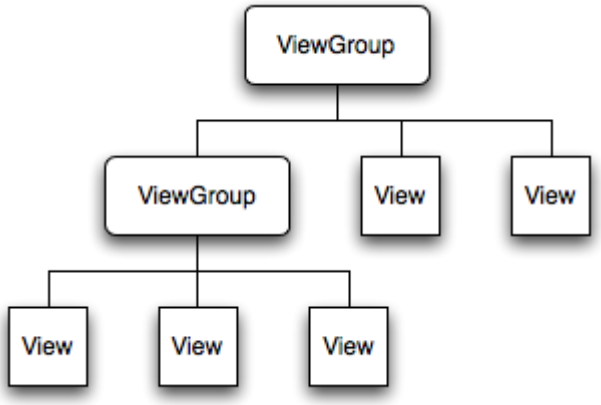
FactorsActivity.java

```
01: package com.st.activities;
02:
03: import android.app.Activity;
04: import android.os.Bundle;
05: import android.view.View;
06: import android.widget.EditText;
07: import android.widget.TextView;
08:
09: public class FactorsActivity extends Activity {
10:   TextView textFactors;
11:   EditText editNumber;
```

```
12:
13:   @Override
14:   protected void onCreate(Bundle data) {
15:         super.onCreate(data);
16:         setContentView(R.layout.activity_factors);
17:         textFactors = (TextView) findViewById(R.id.textFactors);
18:         editNumber = (EditText) findViewById(R.id.editNumber);
19:
20:         if (data != null) {
21:               textFactors.setText(data.getString("output"));
22:         }
23:   }
24:   @Override
25:   protected void onSaveInstanceState(Bundle outState) {
26:         super.onSaveInstanceState(outState);
27:         outState.putString("output",  textFactors.getText().toString());
28:   }
29:
30:   public void getFactors(View v) {
31:         int num = Integer.parseInt(  editNumber.getText().toString());
32:         textFactors.setText("");
33:
34:         for ( int i = 2; i <= num/2; i ++ ) {
35:               if ( num % i == 0 )
36:                  textFactors.append( i + "\n" );
37:
38:         }
39:   }
40: }
```

Views and ViewGroups

- In an Android application, the user interface is built using View and **ViewGroup** objects. There are many types of views and view groups, each of which is a descendant of the **View** class.
- You can build UI using Android's set of predefined widgets and layouts



Widgets

- A widget is a View object that serves as an interface for interaction with the user.
- Android provides a set of fully implemented widgets, like buttons, checkboxes, and text-entry fields, so you can quickly build your UI

Menus

- Menus provide options to users regarding the current activity.
- The most common application menu is revealed by pressing the MENU key on the device.
- Context Menus, which may be shown when the user presses and holds down on an item.
- You define the **onCreateOptionsMenu**() or **onCreateContextMenu**() callback methods for your Activity and declare the items that you want to include in your menu. Android will automatically create the necessary View hierarchy for the menu and draw each of your menu items in it.
- Menus also handle their own events, so there's no need to register event listeners on the items in your menu. When an item in your menu is selected, the **onOptionsItemSelected**() or **onContextItemSelected**() method will be called by the framework.
- You generally declare options for your menu in an XML file.

Styles and Themes

- A style is a set of one or more formatting attributes that you can apply as a unit to individual elements in your layout.
- You could define a style that specifies a certain text size and color, then apply it to only specific View elements.
- A theme is a set of one or more formatting attributes that you can apply as a unit to all activities in an application, or just a single activity. For example, you could define a theme that sets specific colors for the window frame and the panel background, and sets text sizes and colors for menus.
- This theme can then be applied to specific activities or the entire application.
- Styles and themes are resources.
- Android provides some default style and theme resources that you can use

Layouts

- Your layout is the architecture for the user interface in an Activity.
- It defines the layout structure and holds all the elements that appear to the user.
- Each element in XML is either a View or ViewGroup object (or descendant thereof).
- View objects are leaves in the tree, ViewGroup objects are branches in the tree.
- Each layout file must contain exactly one root element, which must be a View or ViewGroup object. Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.
- We must place layout files (XML files) in res/layout folder of application.

You can declare your layout in two ways:

- Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts. This allows you to separate presentation of your application from code. It is also easier to visualize.
- Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.
- You can even club these two ways

The following is the list of different commonly used layout objects.

FrameLayout

- FrameLayout is the simplest type of layout object. It's basically a blank space on your screen that you can later fill with a single object — for example, a picture that you'll swap in and out.
- All child elements of the FrameLayout are pinned to the top left corner of the screen; you cannot specify a different location for a child view.
- Subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/FrameLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="First Button" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Second Button" />
</FrameLayout>
```

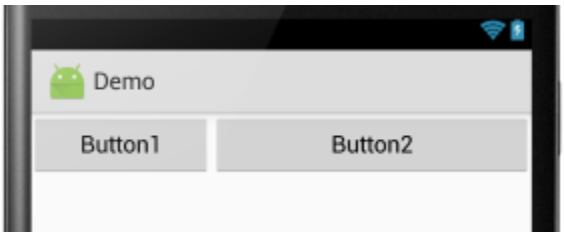
LinearLayout

- Aligns all children in a single direction — vertically or horizontally (default), depending on how you define the orientation attribute.
- It margins between children and the gravity (right, center, or left alignment) of each child.
- It supports assigning a weight to individual children using **layout_weight**. This attribute assigns an "importance" value to a view, and allows it to expand to fill any remaining space in the parent view.
- Child views can specify an integer weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight. Default weight is zero.
- Total of all components weight must be 100.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="25"
        android:text="Button1" />

    <Button
        android:id="@+id/button2"
        android:layout_weight="75"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Button2" />
</LinearLayout>
```



TableLayout

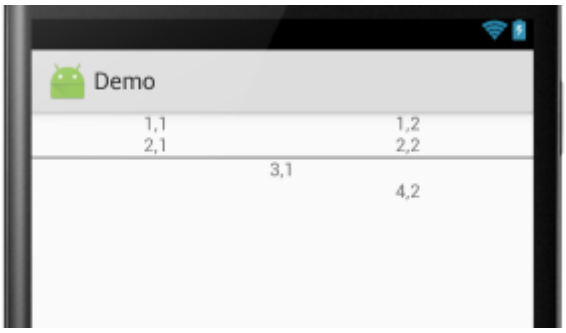
- It positions its children into rows and columns.
- Its containers do not display border lines for their rows, columns, or cells.
- The table will have as many columns as the row with the most cells. A table can leave cells empty.
- **TableRow** objects are the child views of a TableLayout (each TableRow defines a single row in the table). Each row has zero or more cells, each of which is defined by any kind of other View. So, the cells of a row may be composed of a variety of View objects, like ImageView or TextView objects.
- A cell may also be a ViewGroup object (for example, you can nest another TableLayout as a cell).

Attribute Name	Description
android:collapseColumns	The zero-based index of the columns to collapse
android:shrinkColumns	The zero-based index of the columns to shrink
android:stretchColumns	The zero-based index of the columns to stretch

***Note**: The column indices must be separated by a comma: 1, 2, 5. * means all columns.*

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="0,1" >
    <TableRow>
        <TextView android:text="1,1"  android:gravity="center"  />
        <TextView android:text="1,2"  android:gravity="center"  />
    </TableRow>
    <TableRow>
        <TextView android:text="2,1"  android:gravity="center"  />
        <TextView android:text="2,2"  android:gravity="center" />
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="#FF909090" />
    <TableRow>
        <TextView android:text="3,1"  android:gravity="center"
            android:layout_span="2"/>
    </TableRow>
    <TableRow>
        <TextView android:text="4,2"  android:gravity="center"
            android:layout_column="1" />
    </TableRow>

</TableLayout>
```



RelativeLayout

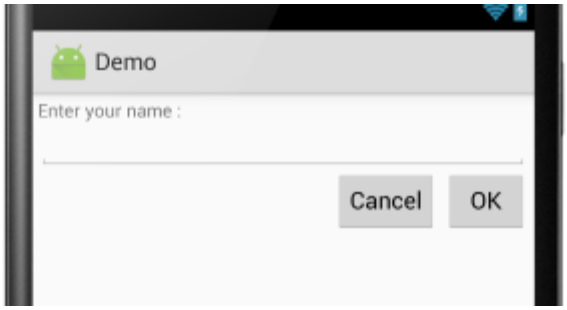
- Lets child views specify their position relative to the parent view or to each other (specified by ID).
- So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.
- Elements are rendered in the order given, so if the first element is centered in the screen, other elements aligning themselves to that element will be aligned relative to screen center.
- Because of this ordering, if using XML to specify this layout, the element that you will reference (in order to position other view objects) must be listed in the XML file before you refer to it from the other views via its reference ID.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="10px" >
    <TextView android:id="@+id/label"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:text="Enter your name :" />

    <EditText android:id="@+id/entry"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_below="@id/label" />

    <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
            android:text="OK" />

    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/ok"
            android:layout_alignTop="@id/ok"
            android:text="Cancel" />
</RelativeLayout>
```



Attributes related to RelativeLayout are:

Attribute Name	Description
android:layout_above	Positions the bottom edge of this view above the given anchor view ID.
android:layout_alignBaseline	Positions the baseline of this view on the baseline of the given anchor view ID.
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID.
android:layout_alignEnd	Makes the end edge of this view match the end edge of the given anchor view ID.
android:layout_alignLeft	Makes the left edge of this view match the left edge of the given anchor view ID.
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent.
android:layout_alignParentEnd	If true, makes the end edge of this view match the end edge of the parent.
android:layout_alignParentLeft	If true, makes the left edge of this view match the left edge of the parent.
android:layout_alignParentRight	If true, makes the right edge of this view match the right edge of the parent.
android:layout_alignParentStart	If true, makes the start edge of this view match the start edge of the parent.
android:layout_alignParentTop	If true, makes the top edge of this view match the top edge of the parent.
android:layout_alignRight	Makes the right edge of this view match the right edge of the given anchor view ID.
android:layout_alignStart	Makes the start edge of this view match the start edge of the given anchor view ID.
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID.
android:layout_alignWithParentIfMissing	If set to true, the parent will be used as the anchor when the anchor cannot be be found for layout_toLeftOf, layout_toRightOf, etc.
android:layout_below	Positions the top edge of this view below the given anchor view ID.
android:layout_centerHorizontal	If true, centers this child horizontally within its parent.
android:layout_centerInParent	If true, centers this child horizontally and vertically within its parent.
android:layout_centerVertical	If true, centers this child vertically within its parent.
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID.
android:layout_toLeftOf	Positions the right edge of this view to the left of the given anchor view ID.
android:layout_toRightOf	Positions the left edge of this view to the right of the given anchor view ID.
android:layout_toStartOf	Positions the end edge of this view to the start of the given anchor view ID.

Summary of Important View Groups

These objects all hold child UI elements. Some provide their own form of a visible UI, while others are invisible structures that only manage the layout of their child views.

Class	Description
FrameLayout	Layout that acts as a view frame to display a single object.
Gallery	A horizontal scrolling display of images, from a bound list.
GridView	Displays a scrolling grid of m columns and n rows.
LinearLayout	A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.
ListView	Displays a scrolling single column list.
RelativeLayout	Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).
ScrollView	A vertically scrolling column of elements.
Spinner	Displays a single item at a time from a bound list, inside a one-row textbox. Rather like a one-row listbox that can scroll either horizontally or vertically.

SurfaceView	Provides direct access to a dedicated drawing surface. It can hold child views layered on top of the surface, but is intended for applications that need to draw pixels, rather than using widgets.
TabHost	Provides a tab selection list that monitors clicks and enables the application to change the screen whenever a tab is clicked.
TableLayout	A tabular layout with an arbitrary number of rows and columns, each cell holding the widget of your choice. The rows resize to fit the largest column. The cell borders are not visible.
ViewFlipper	A list that displays one item at a time, inside a one-row textbox. It can be set to swap items at timed intervals, like a slide show.
ViewSwitcher	Same as ViewFlipper.

Layout Parameters

XML layout attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which it resides.

Attribute	Description
android:layout_height	Specifies the basic height of the view.
android:layout_margin	Specifies extra space on the left, top, right and bottom sides of this view.
android:layout_marginBottom	Specifies extra space on the bottom side of this view.
android:layout_marginLeft	Specifies extra space on the left side of this view.
android:layout_marginRight	Specifies extra space on the right side of this view.
android:layout_marginTop	Specifies extra space on the top side of this view.
android:layout_width	Specifies the basic width of the view.

Layout_height and **Layout_width** could be one of the following:

Constant	Description
fill_parent	The view should be as big as its parent (minus padding). This constant is deprecated starting from API Level 8 and is replaced by **match_parent**.
match_parent	The view should be as big as its parent (minus padding). Introduced in API Level 8.
wrap_content	The view should be only big enough to enclose its content (plus padding).

Layout position

- A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.
- The padding is expressed in pixels for the left, top, right and bottom parts of the view. Padding can be used to offset the content of the view by a specific amount of pixels.

android:paddingBottom
android:paddingLeft
android:paddingRight
android:paddingTop

Unit of measurement

The following units of measure are supported by Android.

Unit	Description
dp	Density-independent Pixels - an abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, so160dp is always one inch regardless of the screen density. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. You should use these units when specifying view dimensions in your layout, so the UI properly scales to render at the same actual size on different screens.

sp	Scale-independent Pixels - this is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.
pt	Points - 1/72 of an inch based on the physical size of the screen.
px	Pixels - corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.
mm	Millimeters - based on the physical size of the screen.
in	Inches - based on the physical size of the screen.

Dimension Resource

- A dimension value defined in XML
- A dimension is specified with a number followed by a unit of measure
- XML file must be in **res/values** folder with any filename.
- The <**dimen**> element's name will be used as the resource ID.

res/values/dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="ball_radius">30dp</dimen>
    <dimen name="font_size">16sp</dimen>
</resources>
```

This application code retrieves a dimension:

```
Resources res = getResources();
float fontSize = res.getDimension(R.dimen.font_size);
```

This layout XML applies dimensions to attributes:

```
<TextView android:layout_height="@dimen/textview_height"    … />
```

Attributes

- Every **View** and **ViewGroup** object supports its own variety of XML attributes.
- Some attributes are specific to a View object but these attributes are also inherited by any View objects that may extend this class.
- Some attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.

ID attribute

- Any View object may have an integer ID associated with it, to uniquely identify the View within the tree.
- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file).
- There are a number of other ID resources that are offered by the Android framework. When referencing an Android resource ID, you do not need the plus-symbol, but must add the android package namespace.
- With the android package namespace in place, we're now referencing an ID from the android.R resources class, rather than the local resources class.

```
android:id="@+id/my_button"
```

Then create an instance of the view object and capture it from the layout (typically in the onCreate() method.

```
<Button android:id="@+id/ok_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ok"/>
```

```
Button okButton = (Button) findViewById(R.id.ok_button);
```

TextView Widget

Displays text to the user and optionally allows them to edit it. For editing, use EditText instead of this.

Method	Meaning
addTextChangedListener (TextWatcher watcher)	Adds a TextWatcher to the list of those whose methods are called whenever this TextView's text changes.
append(CharSequence text)	Append the specified text to the TextView's display buffer, upgrading it to BufferType.EDITABLE if it was not already editable.
getText()	Return the text the TextView is displaying.
length()	Returns the length, in characters, of the text managed by this TextView
setText(CharSequence text)	Sets the string value of the TextView.

Attribute Name	Description
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:capitalize	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.
android:digits	If set, specifies that this TextView has a numeric input method and that these specific characters are the ones that it will accept.
android:editable	If set, specifies that this TextView has an input method.
android:hint	Hint text to display when the text is empty.
android:maxLength	Set an input filter to constrain the text length to the specified number.
android:maxLines	Makes the TextView be at most this many lines tall.
android:numeric	If set, specifies that this TextView has a numeric input method.
android:password	Whether the characters of the field are displayed as password dots instead of themselves.
android:phoneNumber	If set, specifies that this TextView has a phone number input method.
android:selectAllOnFocus	If the text is selectable, select it all when the view takes focus instead of moving the cursor to the start or end.
android:singleLine	Constrains the text to a single horizontally scrolling line instead of letting it wrap onto multiple lines, and advances focus instead of inserting a newline when you press the enter key.
android:text	Text to display.
android:textAllCaps	Present the text in ALL CAPS.
android:textColor	Text color.

Button Widget

Represents a push-button widget.

```
<Button
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="Show Time"
     android:onClick="showTime”/>
```

onClick specifies the method to be used for handling click event. The signature of the method must be as follows:

```
public void  showTime(View v) {
}
```

EditText Widget

- This widget is used to allow user to enter some text such as address.
- It is derived from TextView.

android:inputType

- The type of data being placed in a text field, used to help an input method decide how to let the user enter text. The constants here correspond to those defined by **InputType**.
- Generally you can select a single value, though some can be combined together as indicated. Setting this attribute to anything besides none also implies that the text is editable.
- Must be one or more (separated by '|') of the following constant values.

Constant	Description
none	There is no content type. The text is not editable.
text	Just plain old text.
textCapCharacters	Can be combined with *text* and its variations to request capitalization of all characters.
textCapWords	Can be combined with *text* and its variations to request capitalization of the first character of every word.
textCapSentences	Can be combined with *text* and its variations to request capitalization of the first character of every sentence.
textAutoCorrect	Can be combined with *text* and its variations to request auto-correction of text being input.
textAutoComplete	Can be combined with *text* and its variations to specify that this field will be doing its own auto-completion and talking with the input method appropriately.
textMultiLine	Can be combined with *text* and its variations to allow multiple lines of text in the field. If this flag is not set, the text field will be constrained to a single line.
textUri	Text that will be used as a URI.
textEmailAddress	Text that will be used as an e-mail address..
textEmailSubject	Text that is being supplied as the subject of an e-mail.
textShortMessage	Text that is the content of a short message.
textLongMessage	Text that is the content of a long message.
textPersonName	Text that is the name of a person.
textPostalAddress	Text that is being supplied as a postal mailing address.
textPassword	Text that is a password.
textVisiblePassword	Text that is a password that should be visible.
number	A numeric only field.
numberSigned	Can be combined with *number* and its other options to allow a signed number.
numberDecimal	Can be combined with *number* and its other options to allow a decimal (fractional) number.
phone	For entering a phone number.
datetime	For entering a date and time.
date	For entering a date.
time	For entering a time.

Method	Meaning
void selectAll()	Selects the entire text
void setSelection (int start, int stop)	Selects a part of the text.

CheckBox

- A checkbox is a specific type of two-states button that can be either checked or unchecked.
- This is derived from CompundButton.

Method	Meaning
toggle()	Change the checked state of the view to the inverse of its current state
isChecked()	Returns true if checkbox is checked.
setChecked(boolean checked)	Changes the checked state of this button.
setOnCheckedChangeListener	Register a callback to be invoked when the checked state of this

(CompoundButton.OnCheckedChangeListener listener)	button changes.

RadioButton and RadioGroup

- RadioButton represents a single radio button in a group represented by RadioGroup.
- This is derived from CompoundButton.

```
<RadioGroup
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">
      <RadioButton android:id="@+id/radio_red"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Red" />
      <RadioButton android:id="@+id/radio_blue"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Blue" />
</RadioGroup>
```

ToogleButton

Displays checked/unchecked states as a button with a "light" indicator and by default accompanied with the text "ON" or "OFF".

android:textOff	The text for the button when it is not checked.
android:textOn	The text for the button when it is checked.

Method	**Meaning**
CharSequence getTextOff()	Returns the text for when the button is not in the checked state.
CharSequence getTextOn()	Returns the text for when the button is in the checked state.
void setChecked (boolean checked)	Changes the checked state of this button.
void setTextOff (CharSequence textOff)	Sets the text for when the button is not in the checked state.
void setTextOn (CharSequence textOn)	Sets the text for when the button is in the checked state.

Analog Clock & DigitalClock

These widgets display an analog clock with two hands for hours and minutes and digital clock respectively.

DatePicker Widget

Allows date to be selected by a year, month, and day spinners.

Attribute Name	**Description**
android:calendarViewShown	Whether the calendar view is shown.
android:endYear	The last year (inclusive), for example "2011".
android:maxDate	The minimal date shown by this calendar view in mm/dd/yyyy format.
android:minDate	The minimal date shown by this calendar view in mm/dd/yyyy format.
android:spinnersShown	Whether the spinners are shown.
android:startYear	The first year (inclusive), for example "1940".

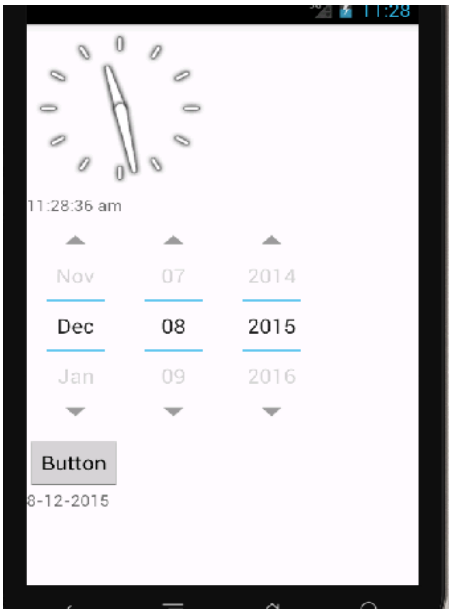
Method	**Meaning**
int getDayOfMonth()	Return day of the month.
long getMaxDate()	Gets the maximal date supported by this DatePicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone.
long getMinDate()	Gets the minimal date supported by this DatePicker in milliseconds since

	January 1, 1970 00:00:00 in getDefault() time zone
int getMonth()	Return month
int getYear()	Returns year.
void setMaxDate (long maxDate)	Sets the maximal date supported by this DatePicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
void setMinDate(long minDate)	Sets the minimal date supported by this NumberPicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
void updateDate(int year, int month, int dayOfMonth)	Updates the current date.

```
<?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" >
    <AnalogClock
        android:id="@+id/analogClock1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <DigitalClock
        android:id="@+id/digitalClock1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="DigitalClock" />
   <DatePicker
        android:id="@+id/datePicker1"
        android:calendarViewShown="false"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"    />
   <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="showDate"
        android:text="Button" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="" />

</LinearLayout>
```



```
public void showDate(View v) {

   DatePicker dp = (DatePicker )  this.findViewById(R.id.datePicker1);
   TextView tv = (TextView) this.findViewById(R.id.textView1);
   tv.setText ( String.format("%d-%d-%d",
          dp.getDayOfMonth(), dp.getMonth() + 1, dp.getYear()));
}
```

Chronometer

Class that implements a simple timer.

Method	Meaning
long getBase()	Return the base time as set through setBase(long).
String getFormat()	Returns the current format string as set through setFormat(String).
Chronometer.OnChronometerTickListener getOnChronometerTickListener()	Returns OnChronometerTickListener.
void setBase(long base)	Set the time that the count-up timer is in reference to.
void setFormat(String format)	Sets the format string used for display.

void setOnChronometerTickListener (Chronometer.OnChronometerTickListener listener)	Sets the listener to be called when the chronometer changes.
void start()	Start counting up.
void stop()	Stop counting up

.

Spinner

A view that displays one child at a time and lets the user pick among them. The items in the Spinner come from the Adapter associated with this view.

android:prompt	The prompt to display when the spinner's dialog is shown.
android:entries	Reference to an array resource that will populate the Spinner.

The list of values may from an array of strings declared in a file in res/values.

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="sportsprompt">Select Sport</string>
    <string-array name="sports">
        <item>Football</item>
        <item>Cricket</item>
        <item>Tennis</item>
        <item>Badminton</item>
    </string-array>

</resources>
```

res/layout/spinner.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" >
    <Spinner
        android:id="@+id/spinner1"
        android:entries="@array/sports"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:prompt="@string/sportsprompt" />
</LinearLayout>
```

SpinnerDemo.java

```
01: public class SpinnerDemo extends Activity {
```

```
02:   Spinner sports;
03:   @Override
04:   public void onCreate(Bundle savedInstanceState) {
05:           super.onCreate(savedInstanceState);
06:     setContentView(R.layout.spinners);
07:     sports = (Spinner) this.findViewById( R.id.spinner1);
08:     sports.setOnItemSelectedListener(
09:      new OnItemSelectedListener() {
10:         @Override
11:         public void onItemSelected(AdapterView<?> parent, View view,
12:               int pos, long id) {
13:           Toast.makeText(parent.getContext(), "You selected  " +
14:                parent.getItemAtPosition(pos).toString(), Toast.LENGTH_LONG).show();
15:         }
16:         @Override
17:         public void onNothingSelected(AdapterView<?> arg0) {
18:         }
19:       }
20:         );
21:       }
22: }
```

You can populate a Spinner with an Adapter, which contains the data to be displayed in the spinner as follows:

```
1: // populate players
2: ArrayAdapter <CharSequence> adapter =
3:  new ArrayAdapter<CharSequence>( this.getApplicationContext(),
4:        android.R.layout.simple_spinner_item);
5: for( int i = 2 ; i <= 11; i ++) {
6:     adapter.add( String.valueOf(i));
7: }
8: players.setAdapter(adapter);
9:
```

Adapter Interface

- An Adapter object acts as a bridge between an **AdapterView** (such as ListView, GridView and Spinner) and the underlying data for that view.
- Adapter is an interface.
- The Adapter provides access to the data items. The Adapter is also responsible for making a `View` for each item in the data set.

Method	**Meaning**
int getCount()	How many items are in the data set represented by this Adapter.
Object getItem(int position)	Get the data item associated with the specified position in the data set.
long getItemId (int position)	Get the row id associated with the specified position in the list.
int getItemViewType (int position)	Get the type of View that will be created by getView(int, View, ViewGroup) for the specified item.
View getView (int position, View convertView, ViewGroup parent)	Get a View that displays the data at the specified position in the data set.
boolean isEmpty()	Returns true if empty

ArrayAdapter

- A concrete BaseAdapter that is backed by an array of arbitrary objects
- By default this class expects that the provided resource id references a single TextView
- However the TextView is referenced, it will be filled with the toString() of each object in the array.
- You can add lists or arrays of custom objects.
- Override the toString() method of your objects to determine what text will be displayed for the item in the list.

```
ArrayAdapter(Context context, int textViewResourceId)
ArrayAdapter(Context context, int textViewResourceId, T[] objects)
ArrayAdapter(Context context, int textViewResourceId, List<T> objects)
```

Method	Meaning
void add(T object)	Adds the specified object at the end of the array.
void addAll(Collection<? extends T> collection)	Adds the specified Collection at the end of the array.
void addAll(T... items)	Adds the specified items at the end of the array.
void clear()	Remove all elements from the list.
static ArrayAdapter<CharSequence> createFromResource(Context context, int textArrayResId, int textViewResId)	Creates a new ArrayAdapter from external resources.
int getCount()	Returns count of items.
View getDropDownView(int position, View convertView, ViewGroup parent)	Get a View that displays in the drop down popup the data at the specified position in the data set.
T getItem(int position)	
int getPosition(T item)	Returns the position of the specified item in the array.
void insert(T object, int index)	Inserts the specified object at the specified index in the array.
void remove(T object)	Removes the specified object from the array.
void setDropDownViewResource(int resource)	Sets the layout resource to create the drop down views.
void sort(Comparator<? super T> comparator)	Sorts the content of this adapter using the specified comparator.

SimpleAdapter

An easy adapter to map static data to views defined in an XML file.

```
SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int
resource, String[] from, int[] to)
```

ListView

A view that shows items in a vertically scrolling list.

```
ListView(Context context)
```

Method	Meaning
void addFooterView(View v)	Add a fixed view to appear at the bottom of the list.
void addHeaderView(View v)	Add a fixed view to appear at the top of the list.
ListAdapter getAdapter()	Returns the adapter currently in use in this ListView.
boolean removeFooterView(View v)	Removes a previously-added footer view.
boolean removeHeaderView(View v)	Removes a previously-added header view.
void setAdapter(ListAdapter adapter)	Sets the data behind this ListView.
void setDivider(Drawable divider)	Sets the drawable that will be drawn between each item in the list.
void setDividerHeight(int height)	Sets the height of the divider that will be drawn between each item in the list.
void setSelection(int position)	Sets the currently selected item.
void setSelectionFromTop(int position, int y)	Sets the selected item and positions the selection y pixels from the top edge of the ListView.
void smoothScrollByOffset(int offset)	Smoothly scroll to the specified adapter position offset.
void smoothScrollToPosition(int position)	Smoothly scroll to the specified adapter position.

```
01: public class ListViewDemoActivity extends ListActivity {
02:     ListView lv;
03:     @Override
04:     public void onCreate(Bundle savedInstanceState) {
05:         super.onCreate(savedInstanceState);
06:         setListAdapter(new ArrayAdapter<String>(this,
07:                         android.R.layout.simple_list_item_1, LANGS));
08:         lv = getListView();
09:         lv.setTextFilterEnabled(true);
10:         lv.setOnItemClickListener(
11:            new OnItemClickListener() {
12:              public void onItemClick(AdapterView<?> parent, View view,
13:                         int position, long id) {
14:              Toast.makeText(getApplicationContext(),
15:                          lv.getItemAtPosition(position).toString(),
16:                          Toast.LENGTH_SHORT).show();
17:            }
18:         });
19:     } // onCreate
20:     static final String[] LANGS =
21:         new String[] {"Java","C++","C","Python","C#",
22:                     "Visual Basic.Net","Ruby","PHP","JavaScript" };
23: }
```

GridView

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a ListAdapter.

activity_grid_view.xml

```
01: <GridView xmlns:android="http://schemas.android.com/apk/res/android"
02:     xmlns:tools="http://schemas.android.com/tools"
03:     android:id="@+id/gridView"
04:     android:layout_width="match_parent"
05:     android:numColumns="auto_fit"
06:     android:gravity="center"
07:     android:padding="5dip"
08:     android:layout_height="match_parent">
09: </GridView>
```

GridViewActivity.java

```
01: package com.st.activities;
02: import java.util.ArrayList;
03: import java.util.Calendar;
04: import android.app.Activity;
05: import android.os.Bundle;
06: import android.widget.ArrayAdapter;
07: import android.widget.GridView;
08:
09: public class GridViewActivity extends Activity {
10:   GridView gv;
11:   @Override
12:   protected void onCreate(Bundle savedInstanceState) {
13:         super.onCreate(savedInstanceState);
14:         setContentView(R.layout.activity_grid_view);
15:         gv = (GridView) findViewById( R.id.gridView);
16:         Calendar c = Calendar.getInstance();
17:         int year = c.get( Calendar.YEAR);
18:
19:         ArrayList<String> years = new ArrayList<String>();
20:         for ( int i = year - 9; i <= year; i ++)
21:               years.add( String.valueOf(i));
22:
23:         ArrayAdapter<String> adapter = new ArrayAdapter<String>( this,
24:                         android.R.layout.simple_list_item_1, years);
25:         gv.setAdapter(adapter);
26:   }
27: }
```

Toast Notification

- A toast notification is a message that pops up on the surface of the window.
- It only fills the amount of space required for the message and the user's current activity remains visible and interactive. The notification automatically fades in and out, and does not accept interaction events.
- Because a toast can be created from a background Service, it appears even if the application isn't visible.
- A toast cannot accept user interaction events; if you'd like the user to respond and take action, consider using a Status Bar Notification instead.
- First, instantiate a **Toast** object with one of the **makeText**() methods. This method takes three parameters: the application **Context**, the text **message**, and the **duration** for the toast. Duration can be either LENGTH_LONG or LENGTH_SHORT.
- **Toast**.**makeText**() returns a properly initialized Toast object.
- You can display the toast notification with show()
- A standard toast notification appears near the bottom of the screen, centered horizontally
- You can change this position with the **setGravity**(int gravity, int xoffset, int yoffset) method.

```
Toast.makeText(context, text, duration).show();
```

For example, if you decide that the toast should appear in the top-left corner, you can set the gravity like this:

```
toast.setGravity(Gravity.TOP, 0, 0);
```

Creating a Custom Toast View

- If a simple text message isn't enough, you can create a customized layout for your toast notification.
- To create a custom layout, define a View layout, in XML or in your application code, and pass the rootView object to the **setView**(View) method.

res/layout/toast_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/toast_layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="10dp"
              android:background="#DAAA">
    <ImageView android:id="@+id/image"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:layout_marginRight="10dp"/>
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="fill_parent"
              android:textColor="#FFF"/>
</LinearLayout>
```

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout,
                    (ViewGroup) findViewById(R.id.toast_layout_root));

ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Hello! This is a custom toast!");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

App Manifest

- Every application must have AndroidManifest.xml file in its root directory
- It contains the following information
  - Package name that serves as a unique identifier for the application.
  - Components of the application — the activities, services, broadcast receivers, and content providers that the application is composed of.
  - Processes that will host application components.
  - Permissions the application must have in order to access protected parts of the API and interact with other applications.
  - The permissions that others are required to have in order to interact with the application's components.
  - The minimum level of the Android API that the application requires.
  - It lists the libraries that the application must be linked against.
- Elements at the same level need not be ordered. For example, <activity>, <provider>, and<service> elements can be intermixed in any sequence. (An <activity-alias> element is the exception to this rule: It must follow the <activity> it is an alias for.)
- Except for some attributes of the root <manifest> element, all attribute names begin with an android:prefix
- The name of the class must include the full package designation. However, if first character of the string is a period, the string is appended to application's package name.
- Names that refer to resources are expressed as @[package:] type:name, where package can be omitted if resource is in the same package as application, type is type of resource – string or drawable and name identifies the specific resources.
- Components advertise their capabilities — the kinds of intents they can respond to — through intent filters. Since the Android system must learn which intents a component can handle before it launches the component, intent filters are specified in the manifest as <intent-filter> elements.
- Each permission is identified by a unique label. Often the label indicates the action that's restricted. For example, here are some permissions defined by Android:
  - android.permission.READ_OWNER_DATA
  - android.permission.SET_WALLPAPER
- If your application uses code from any of these packages, it must explicitly asked to be linked against them. The manifest must contain a separate <uses-library>element to name each of the libraries.

Fragments

- A Fragment represents a behavior or a portion of user interface in an Activity.
- A fragment is a subclass of **Fragment** class.
- A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.
- When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout
- Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets.

Life Cycle

Like an activity, a fragment can exist in three states:

Resumed	The fragment is visible in the running activity.
Paused	Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).
Stopped	The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive (all state and member information is retained by the system). However, it is no longer visible to the user and will be killed if the activity is killed.

***Note**: Also like an activity, you can retain the state of a fragment using a **Bundle**, in case the activity's process is killed and you need to restore the fragment state when the activity is recreated. You can save the state during the fragment's **onSaveInstanceState**() callback and restore it during either **onCreate**(), **onCreateView**(), or **onActivityCreated**()*

The following are important methods in Fragment's lifecycle.

onCreate()	The system calls this when creating the fragment. Within your implementation, 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 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 from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
onPause()	The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

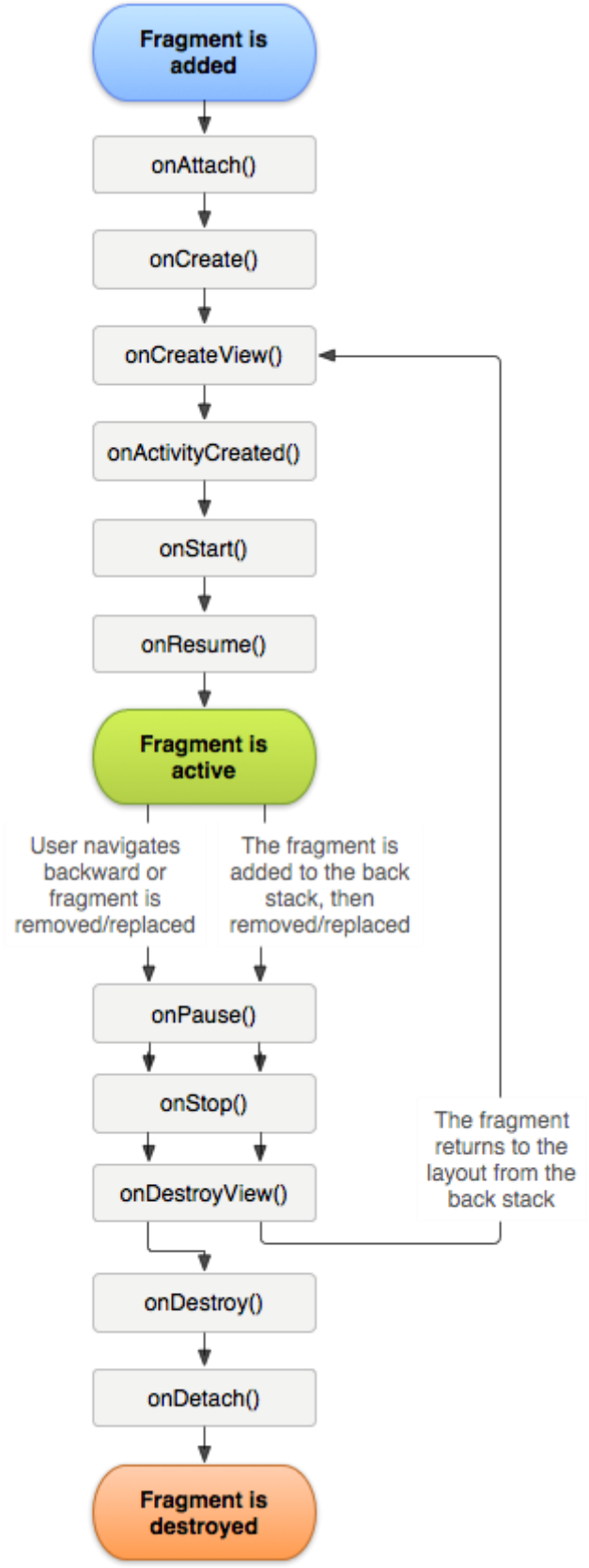
Subclasses of Fragment

There are also a few subclasses that you might want to extend, instead of the base Fragment class:

DialogFragment
Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the Activity class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.

ListFragment
Displays a list of items that are managed by an adapter (such as a SimpleCursorAdapter), similar toListActivity. It provides several methods for managing a list view, such as the onListItemClick()callback to handle click events.

PreferenceFragment
Displays a hierarchy of Preference objects as a list, similar to PreferenceActivity. This is useful when creating a "settings" activity for your application.


Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed

Example

The following example shows how we use two different Fragments in a single layout and send data from one to another.

This example also shows how we can support different modes like Portrait and Landscape so that fragments are arranged differently in different modes.

CourseFragment.java

```
01: package com.st.demo;
02: import android.app.Fragment;
03: import android.os.Bundle;
04: import android.util.Log;
05: import android.view.LayoutInflater;
06: import android.view.View;
07: import android.view.ViewGroup;
08: import android.widget.RadioGroup;
09:
10: public class CoursesFragment extends Fragment
11:                       implements RadioGroup.OnCheckedChangeListener {
12:     public View onCreateView(LayoutInflater inflater, ViewGroup container,
13:                              Bundle savedInstanceState) {
14:         View v = inflater.inflate(R.layout.fragment_courses, container, false);
15:         RadioGroup rgCourses = (RadioGroup) v.findViewById(R.id.rgCourses);
16:         rgCourses.setOnCheckedChangeListener(this);  // register event handler
17:         return v;
18:     }
19:     public void onCheckedChanged(RadioGroup group, int checkedId) {
20:         String desc = "";
21:         switch (checkedId) {
22:             case R.id.rbAndroid:
23:                 desc = "Android Programming ..";
24:                 break;
25:             case R.id.rbJava:
26:                 desc = "Java SE and EE ..";
27:                 break;
28:             case R.id.rbOracle:
29:                 desc = "Oracle Database ..";
30:                 break;
31:         }
32:         FragmentsDemo fd = (FragmentsDemo) getActivity();
33:         fd.setCourseDescription(desc);
34:     }
35: }
```

CourseDescriptionFragment.java

```
01: package com.st.demo;
02: import android.app.Fragment;
03: import android.os.Bundle;
04: import android.util.Log;
05: import android.view.LayoutInflater;
06: import android.view.View;
07: import android.view.ViewGroup;
08: import android.widget.TextView;
09:
10: public class CourseDescriptionFragment extends Fragment {
11:     View _view;
12:     TextView _tv;
13:     @Override
14:     public View onCreateView(LayoutInflater inflater,
15:                                  ViewGroup container, Bundle savedInstanceState) {
16:         _view =  inflater.inflate(R.layout.fragment_course_description,
17:                                     container,false);
18:         return _view;
19:     }
20:     public void setCourseDescription(String desc) {
```

```
21:         _tv = (TextView) getView().findViewById(R.id.tvDescription);
22:         _tv.setText(desc);
23:     }
24: }
25:
```

FragmentsActivity.java

```
01: package com.st.demo;
02: import android.app.Activity;
03: import android.os.Bundle;
04: import android.util.Log;
05: import android.widget.TextView;
06:
07: public class FragmentsActivity extends Activity {
08:     CourseDescriptionFragment f;
09:     @Override
10:     protected void onCreate(Bundle savedInstanceState) {
11:         super.onCreate(savedInstanceState);
12:         setContentView(R.layout.activity_fragments);
13:     }
14:     public void setCourseDescription(String desc) {
15:         f = (CourseDescriptionFragment) getFragmentManager().
16:                                             findFragmentById(R.id.courseDescFragment);
17:         f.setCourseDescription(desc);
18:     }
19: }
```

res/layout/activity_fragments.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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"
tools:context="com.st.demo.FragmentsActivity">
<fragment
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:name="com.st.demo.CoursesFragment"
    android:layout_marginBottom="20dp"
    android:id="@+id/coursesFragment" />

<fragment
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:name="com.st.demo.CourseDescriptionFragment"
    android:id="@+id/courseDescFragment"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"/>
</LinearLayout>
```

res/layout-land/activity_fragments.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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="horizontal"
tools:context="com.st.demo.FragmentsActivity">
<fragment
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:name="com.st.demo.CoursesFragment"
    android:layout_marginRight="20dp"
    android:id="@+id/coursesFragment" /> " />
```

```
<fragment
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:name="com.st.demo.CourseDescriptionFragment"
    android:id="@+id/courseDescFragment"
    android:layout_gravity="center_horizontal" />
</LinearLayout>
```

fragement_course_description.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"
    tools:context="com.st.demo.CoursesDescriptionFragment">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Java Course Details...."
        android:id="@+id/tvDescription"
        android:layout_gravity="left" />
</FrameLayout>
```

fragement_courses.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"
    tools:context="com.st.demo.CoursesFragment">
    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center_horizontal|top"
        android:id="@+id/rgCourses">
        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Java"
            android:checked="true"
            android:id="@+id/rbJava" />
        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Andriod"
            android:id="@+id/rbAndroid" />
        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Oracle Database"
            android:id="@+id/rbOracle" />
    </RadioGroup>
</FrameLayout>
```