UNIT - V

Toast, Menu, Dialog, List and Adapters

5.1 Menu

- Menus are useful for displaying additional options that are not directly visible on the main UI of an application.

- There are two main types of menus in Android:

  - Options menu
  - Context menu

5.1 Menu

1. Options menu

- Options menu displays options (information) related to the current activity. In Android, you activate the options menu by pressing the MENU button.

5.1 Menu

1. Options menu

- To implement and display the options menu for the activity, you need to implement two methods in your activity:
  - **onCreateOptionsMenu()**
  - **onOptionsItemSelected().**

5.1 Menu

1. Options menu (Example)

activity_main.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >


 </RelativeLayout>
```

5.1 Menu

1. Options menu (Example)

MainActivity.java

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

    public boolean onCreateOptionsMenu(Menu menu)
    {    super.onCreateOptionsMenu(menu);

            MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
            MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
            MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
            MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");

    }
```

5.1 Menu

1. Options menu (Example)

MainActivity.java

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  switch (item.getItemId())
  {
      case 0:
              Toast.makeText(this, "You clicked on Item 1", Toast.LENGTH_LONG).show();
               return true;
      case 1:
               Toast.makeText(this, "You clicked on Item 2", Toast.LENGTH_LONG).show();
               return true;
      case 2:
               Toast.makeText(this, "You clicked on Item 3", Toast.LENGTH_LONG).show();
               return true;
      case 3:
               Toast.makeText(this, "You clicked on Item 4", Toast.LENGTH_LONG).show();
              return true;
   }
      return false;
  }
}
```

5.1 Menu

1. Options menu (Example)

- The **onCreateOptionsMenu(Menu menu)** method is called when the MENU button is pressed which implement and display the options menu as shown in below figure :

5.1 Menu

1. Options menu (Example)

- The **onCreateOptionsMenu(Menu menu)** method takes a **Menu** as argument and adds a series of menu items to it.
- To add a menu item to the menu, you create an instance of the **MenuItem** class and use the **add()** method of the **Menu** object:

  MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");

- The four arguments of the add() method are as follows:
  - **groupId**: It is group identifier that the menu item should be part of. Use 0 if an item is not in a group.
  - **itemId** — A unique item ID
  - **order** — The order in which the item should be displayed
  - **title** — The text to display for the menu item

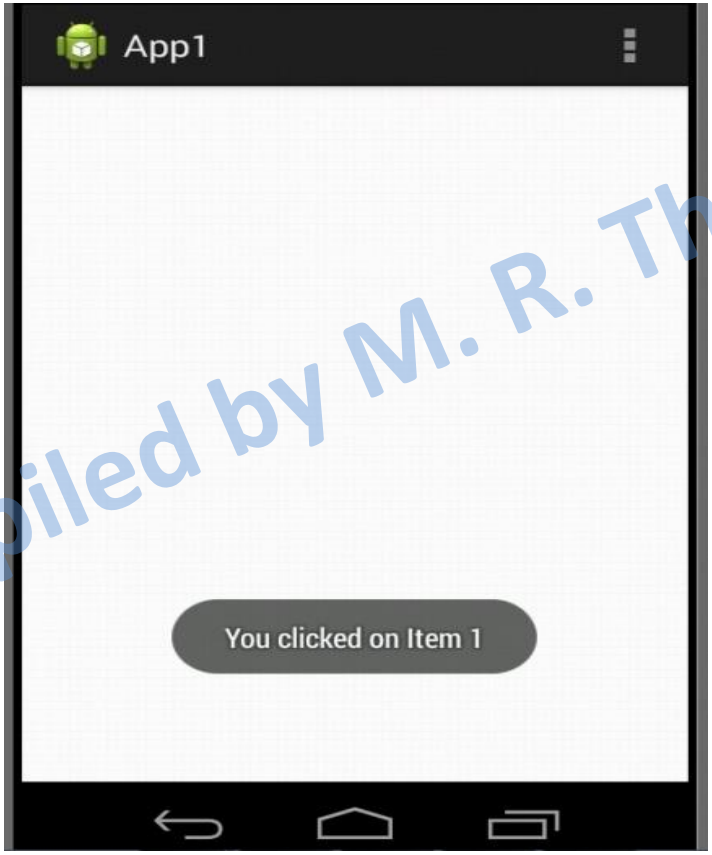
5.1 Menu

1. Options menu (Example)

- When a menu item is selected, the **onOptionsItemSelected(MenuItem item)** method is called which takes **MenuItem** as argument and checks its ID to determine the menu item that is selected. It then displays a Toast message to let the user know which menu item was selected.

5.1 Menu

1. Options menu (Example)



5.1 Menu

2. Context Menu

- Context menu displays options (information) related to a particular view on an activity.
- In Android, to activate a context menu you tap and hold on to it.

5.1 Menu

2. Context Menu

- If you want to associate a context menu with a view on an activity, you need to call the **setOnCreateContextMenuListener()** method of that particular view.

5.1 Menu

2. Context Menu (Example)

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >

   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:text="@string/btn1_text " />

</RelativeLayout>
```

5.1 Menu

2. Context Menu (Example)

strings.xml

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

<resources>
    <string name="app_name">App1</string>
    <string name="action_settings">Settings</string>
    <string name="btn1_text">Button</string>
</resources>
```

5.1 Menu

2. Context Menu (Example)

MainActivity.java

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

      Button btn = (Button) findViewById(R.id.button1);
      btn.setOnCreateContextMenuListener(this);

    }
```

5.1 Menu

2. Context Menu (Example)

MainActivity.java

```
@Override
public void onCreateContextMenu(ContextMenu menu, View view,
                                ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, view, menuInfo);

    MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
    MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
    MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
    MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
}
```

5.1 Menu

2. Context Menu (Example)

MainActivity.java

```
@Override
public boolean onContextItemSelected(MenuItem item)
{    super.onContextItemSelected(item);

     switch (item.getItemId())
     {
        case 0:
                    Toast.makeText(this, "You clicked on Item 1",  Toast.LENGTH_LONG).show();
                    return true;
        case 1:
                    Toast.makeText(this, "You clicked on Item 2", Toast.LENGTH_LONG).show();
                    return true;
        case 2:
                    Toast.makeText(this, "You clicked on Item 3",  Toast.LENGTH_LONG).show();
                    true;
        case 3:
                    Toast.makeText(this, "You clicked on Item 4", Toast.LENGTH_LONG).show();
                    return true;
     }
     return false;
}
}
```

5.1 Menu

2. Context Menu (Example)

- In the preceding example, you call the **setOnCreateContextMenuListener()** method of the Button view to associate it with a context menu.
- When the user taps and holds the Button view, the **onCreateContextMenu()** method is called which implements and display context menu as shown below:

5.1 Menu

2. Context Menu (Example)

- When a menu item is selected, the **onContextItemSelected(MenuItem item)** method is called which takes **MenuItem** as argument and checks its ID to determine the menu item that is selected. It then displays a Toast message to let the user know which menu item was selected.

5.1 Menu

2. Context Menu (Example)

5.2 Dialog

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

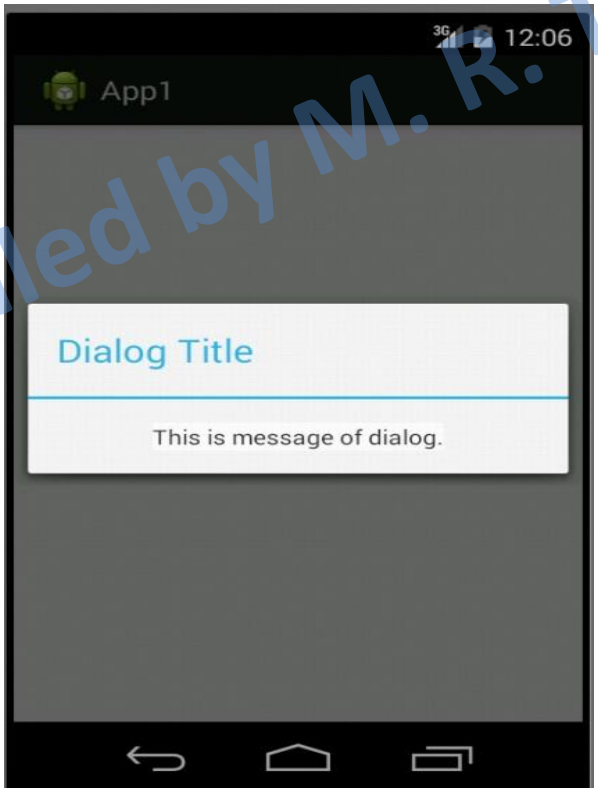
- Dialog can be implemented using:

  - **Dialog class**
  - **AlertDialog class**

5.2 Dialog

- **Dialog Class**

- The **Dialog** class is the base class for dialogs. To use the base Dialog class, it is required to create a new instance and set the title and layout, using the **setTitle()** and **setContentView()** methods. Once it is configured use the **show()** method to display a dialog as shown in below example.



5.2 Dialog

- **Dialog Class (Example)**

MainActivity.java

```
public class MainActivity extends ActionBarActivity
{

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

                Dialog d = new Dialog(this);
                d.setTitle("Dialog Title");
```

5.2 Dialog

- **Dialog Class (Example)**

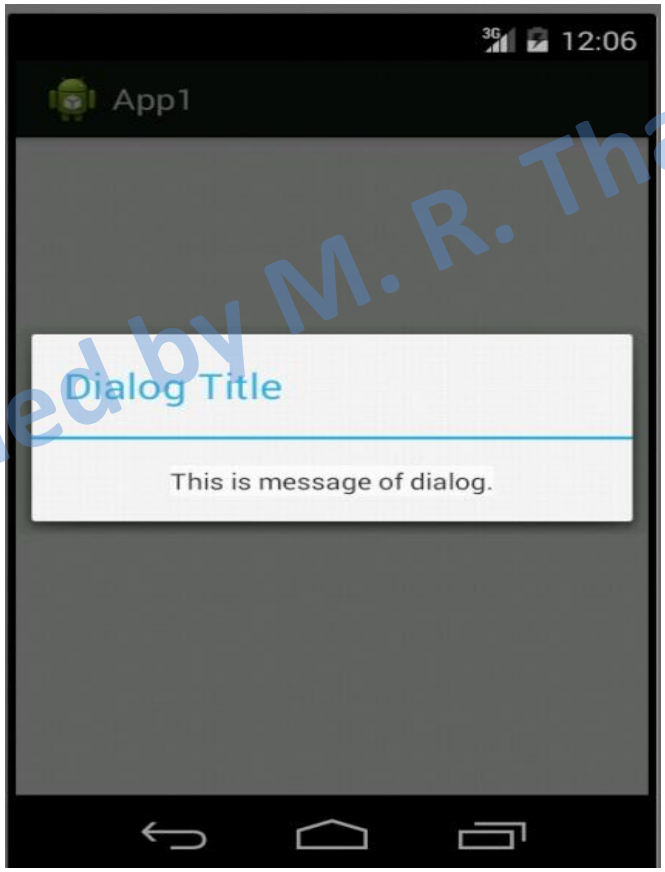
MainActivity.java

```
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);

        TextView  myTextView = new TextView(this);
        myTextView.setText("This is message of dialog.");

        int  Height = LinearLayout.LayoutParams.FILL_PARENT;
        int  Width = LinearLayout.LayoutParams.WRAP_CONTENT;

        ll.addView(myTextView, new LinearLayout.LayoutParams(Height,Width));

        d.setContentView(ll);
        d.show();
    }
}
```

5.2 Dialog

- **Dialog Class**

- When you run this application, the Dialog is displayed as shown in below figure :



5.2 Dialog

- **AlertDialog Class**

- **AlertDialog** class is a subclass of the Dialog class that allows you to build a variety of dialog designs and is the only dialog class which mostly used.

- There are three regions of an alert dialog. An Alert dialog can show a title, up to three buttons, a list of selectable items, or a custom layout.

  - **Title:** Title is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If you need to state a simple message or question such a s the previous dialog, you don't need a title.
  - **Content area:** Content area can display a message, a list, or other custom layout.
  - **Action buttons:** Alert dialog can display up to maximum three action buttons.

5.2 Dialog

- **AlertDialog Class**

- To construct the Alert Dialog user interface, create a new **AlertDialog.Builder** object as follows:

```
AlertDialog.Builder ad = new AlertDialog.Builder(context);
```

- You can then assign values for the **title** and **message** to display, and **optionally assign values to be used for any buttons**, selection items, and text input boxes you wish to display.

5.2 Dialog

- **AlertDialog Class (Example)**

MainActivity.java

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

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

      ad.setTitle("AlertDialog Title");
      ad.setMessage("This is message of AlertDialog.");
      ad.show();

      }
}
```

5.2 Dialog

- **AlertDialog Class**

- When you run this application, the AlertDialog is displayed as shown in below figure :

5.2 Dialog

▪ AlertDialog Class

Adding Buttons

- There are maximum three different action buttons you can add:
  - **Positive:** It should be used to accept and continue with the action like "OK" action.
  - **Negative:** It should be used to cancel the action.
  - **Neutral:** You should use this when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."

5.2 Dialog

- **AlertDialog Class (Example)**

MainActivity.java

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

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

      ad.setTitle("AlertDialog Title");
      ad.setMessage("This is message of AlertDialog.");
```

5.2 Dialog

- **AlertDialog Class (Example)**

MainActivity.java

```
ad.setPositiveButton("OK",  new OnClickListener()  {
                public void onClick(DialogInterface dialog, int arg1)
                {
                        // Code here for OK button
                }
        });
ad.setNegativeButton("Cancel", new OnClickListener(){

                public void onClick(DialogInterface dialog, int arg1)
                {
                        // Code here for Cancel button
                }
        });
```

5.2 Dialog

- **AlertDialog Class (Example)**

MainActivity.java

```
ad.setNeutralButton("Remind me later", new OnClickListener(){

                public void onClick(DialogInterface dialog, int arg1)
                {
                        // Code here for Remind me later button
                }
        });

 ad.show();
 }
 }
```

5.2 Dialog

- **Dialog Class**

- When you run this application, the AlertDialog is displayed as shown in below figure :

5.3 Toast

- A Toast provides **simple notification** about an operation in a **small popup.**

- It only occupies the space required for the message and the current activity remains visible and interactive.

- Toast is a **passive**, **non-blocking** user notification that shows a simple message at the bottom of the user's screen as shown in below figure:

5.3 Toast

5.3 Toast

- The Toast class includes a static **makeText()** method that creates a standard Toast display window.

- It requires to pass the **application Context**, the **text message to display**, and the **length of time to display it (LENGTH_SHORT or LENGTH_LONG)** into the makeText() method to construct a new Toast.

- Once a Toast has been created, display it by calling **show()** method, as shown in below.


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

5.3 Toast

Example

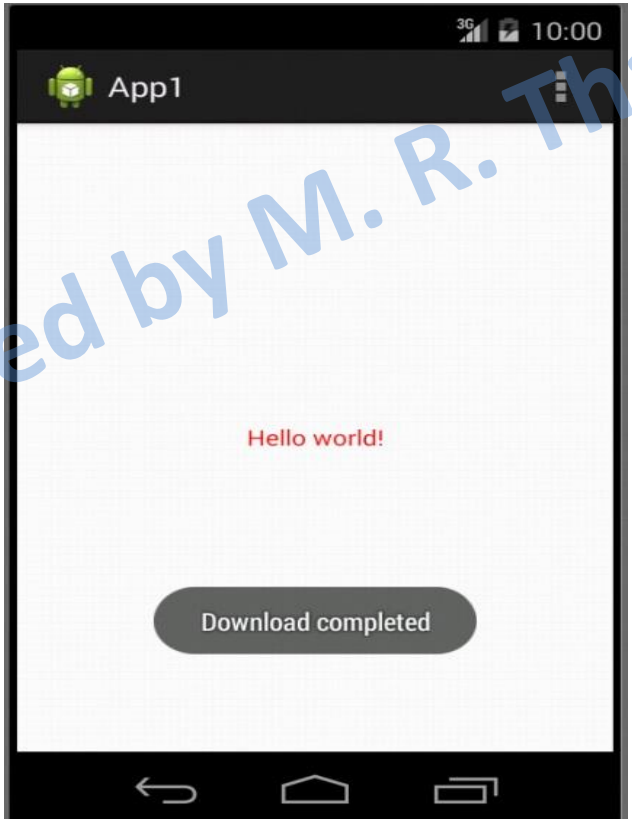
MainActivity.java

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

       Context context = getApplicationContext();
       String msg = "Download completed";
       int duration = Toast.LENGTH_LONG;

       Toast toast = Toast.makeText(context, msg, duration);
        toast.show();

       }
}
```

5.3 Toast

Example

- When you run this application, the Toast is displayed as shown in below figure :



5.3 Toast

Customizing Toasts

- You can modify a Toast by setting its display position and assigning it alternative Views or layouts.

5.3 Toast

Positioning Toast

- A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the **setGravity(int, int, int)** method. This accepts three parameters: a **Gravity constant, an x-position offset, and a y-position offset.**

5.3 Toast

Positioning Toast (Example)

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

        Context context = getApplicationContext();
        String msg = "This is Android Toast message!";
        int duration = Toast.LENGTH_SHORT;

        Toast toast = Toast.makeText(context, msg, duration);

        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

}
}
```

5.3 Toast

Positioning Toast (Example)

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

        Context context = getApplicationContext();
        String msg = "This is Android Toast message!";
        int duration = Toast.LENGTH_SHORT;

        Toast toast = Toast.makeText(context, msg, duration);

        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

}
}
```

5.3 Toast

Positioning Toast (Example)

- When you run this application, the Toast is displayed at the center of the screen as shown in below figure :

5.3 Toast

Assigning alternative view to Toast

- 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 root View object to the **setView(View)** method.

5.3 Toast

Example

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

        Context context = getApplicationContext();
        String msg = "This is Customized Toast!";
        int duration = Toast.LENGTH_LONG;

        Toast toast = Toast.makeText(context, msg, duration);
```

5.3 Toast

Example

```
LinearLayout ll = new LinearLayout(context);
ll.setOrientation(LinearLayout.VERTICAL);

ImageView myImageView = new ImageView(context);
myImageView.setImageResource(R.drawable.ic_launcher);

TextView myTextView = new TextView(context);
myTextView.setText(msg);

int lHeight = LinearLayout.LayoutParams.FILL_PARENT;
int lWidth = LinearLayout.LayoutParams.WRAP_CONTENT;

ll.addView(myImageView, new LinearLayout.LayoutParams(lHeight, lWidth));
ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));

toast.setView(ll);
 toast.show();
}
}
```

5.3 Toast

Example

- When you run this application, the Customized Toast is displayed as shown in below figure :

5.4 List & Adapter

ListView

- Android ListView is a view which groups several items and display them in vertical scrollable list.

- If the list items to be displayed in the list are specified using an array, then it can be inserted to the list using its **android:entries** attribute.

5.4 List & Adapter

Example

- Android ListView is a view which groups several items and display them in vertical scrollable list.

- If the list items to be displayed in the list are specified using an array, then it can be inserted to the list using its **android:entries** attribute.

5.4 List & Adapter

Example

- **activity_main.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/text1_text"
        android:textColor="#0000FF"
        android:textSize="25sp" />

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:entries="@array/car_list" >

    </ListView>

</LinearLayout>
```

5.4 List & Adapter

Example

- **strings.xml**

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

    <string name="app_name">App15</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="text1_text">Car list</string>


    <string-array name="car_list">
        <item>Swift</item>
        <item>Wagonr</item>
        <item>Alto</item>
        <item>SX4</item>
    </string-array>

</resources>
```

5.4 List & Adapter

Example

- **MainActivity.java**

```
public class MainActivity extends ActionBarActivity
 {


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

5.4 List & Adapter

Example

5.4 List & Adapter

Adapter

- An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to spinner, list view, grid view etc.

- The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

5.4 List & Adapter

Array Adapter

- You can use this adapter when your data source is an array.
- By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView.
- Consider you have an array of strings you want to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the **layout for each string** and the **string array**:


```
ArrayAdapter<String> adapter = new ArrayAdapter<String>
                                (this, R.layout.fragment_main, StringArray);
```

5.4 List & Adapter

Array Adapter

- Once you have array adapter created, then simply call **setAdapter()** on your ListView object as follows:

```
ListView listView = (ListView) findViewById(R.id.listView1);
listView.setAdapter(adapter);
```

5.4 List & Adapter

Example

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >

   <ListView
      android:id="@+id/listView1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textAlignment="center" >
   </ListView>

</RelativeLayout>
```

5.4 List & Adapter

Example

fragment_main.xml

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txt1_text" />
```

5.4 List & Adapter

Example

strings.xml

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

<resources>
    <string name="app_name">App1</string>
    <string name="action_settings">Settings</string>
    <string name="txt1_text">Text view</string>
</resources>
```

5.4 List & Adapter

Example

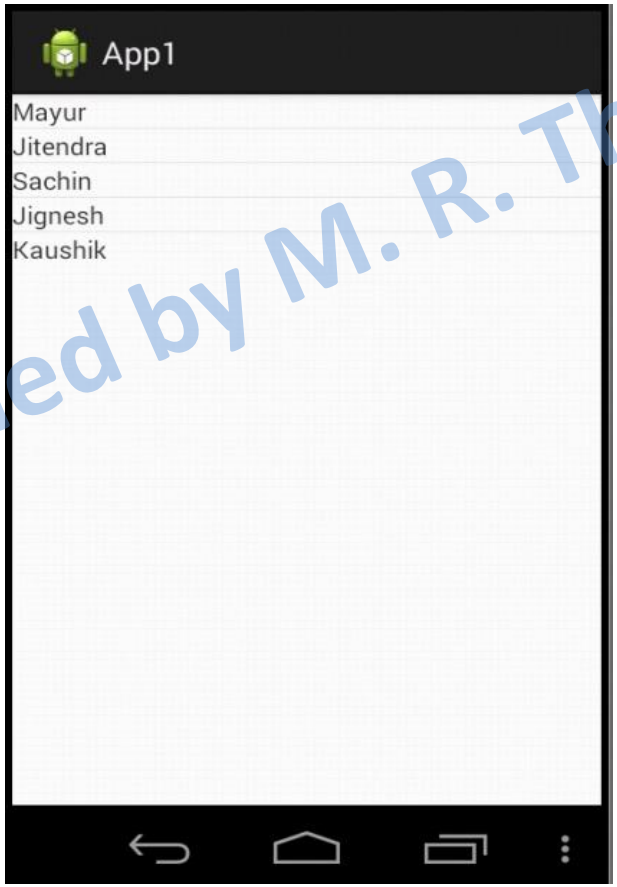
MainActivity.java

```
public class MainActivity extends ActionBarActivity
{
    String[] name={"Mayur","Jitendra","Sachin","Jignesh","Kaushik"};

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

    ArrayAdapter<String> adapter = new
                ArrayAdapter<String>(this,R.layout.fragment_main,name);

    ListView listView = (ListView) findViewById(R.id.listView1);
    listView.setAdapter(adapter);
}
}
```

5.4 List & Adapter

Example

- When you run the application, all the names in the **String array** is displayed in the **ListView** as shown in below figure :



5.6 Basic operations of SQLite Database

- SQLite is a **open source** SQL database that stores data to a **text file** on a device. SQLite is also **embedded** into every Android device.

- The features of SQLite database are:
  - Open-source
  - Standards-compliant
  - Lightweight
  - Single-tier

- SQLite supports the data types **TEXT** similar to String in Java, **INTEGER** similar to long in Java and **REAL** similar to double in Java. All other types must be converted into one of these fields before getting saved in the database.

5.6 Basic operations of SQLite Database

Working with SQLite Database

- A good practice for dealing with databases is to create a helper class to encapsulate all the complexities of accessing the data so that it is transparent to the calling code.

- Hence, for this section, a helper class called **DBAdapter** is created that opens, closes, and perform operations on a SQLite database.

- In DBAdapter class a database called **MyDB** is created which contains one table named **contacts**. This table will have three columns: **id, name, and email.**

5.6 Basic operations of SQLite Database

- **Performing Database Operations**
- With the DBAdapter helper class created, you are now ready to work with the database.
- In the following sections, you will learn how to perform :
  - Insert
  - Update
  - Delete

5.6 Basic operations of SQLite Database

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerInParent="true"
        android:text="@string/txt1_text" />

</RelativeLayout>
```

5.6 Basic operations of SQLite Database

strings.xml

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

<resources>
    <string name="app_name">App1</string>
    <string name="action_settings">Settings</string>
    <string name="txt1_text">SQLite Database Application</string>
</resources>
```

5.6 Basic operations of SQLite Database

- **Insert Contacts**

MainActivity.java

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

             class DBHelper extends SQLiteOpenHelper
             {
                                    public DBHelper()
                                    {
                                                super(getApplicationContext(),"MyDB",null,1);
                                    }
                                    @Override
                                    public void onCreate(SQLiteDatabase db)
                                    {
                                                db.execSQL("create table contacts( id integer primary key
                                                  autoincrement,name  text not null,email text not null)");
                                    }
                                    @Override
                                    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2)
                                    {
                                                // TODO Auto-generated method stub
                                    }
             }
```

5.6 Basic operations of SQLite Database

- **Insert Contacts**

MainActivity.java

```
DBHelper helper = new DBHelper();
SQLiteDatabase db = helper.getWritableDatabase();

db.execSQL("insert into contacts values (1,'Mayur','email1@gmail.com')");
db.execSQL("insert into contacts values (1,'Kaushik','email2@gmail.com')");

Cursor c = db.query(true, "contacts", new String[]{"id", "name", "email"}, null, null, null, null,
    null, null);   //select query

c.moveToFirst();
do
{       //displaying one by one table records in Toast
      Toast.makeText(this, "id: " + c.getString(0) + "\n" +  "Name: " + c.getString(1) + "\n"  +
            "Email:  " + c.getString(2), Toast.LENGTH_LONG).show();
 }while (c.moveToNext());

 helper.close();
}
}
```

5.6 Basic operations of SQLite Database

- **Update a Contact**

MainActivity.java

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

             class DBHelper extends SQLiteOpenHelper
             {
                                public DBHelper()
                                {
                                        super(getApplicationContext(),"MyDB",null,1);
                                }
                                @Override
                                public void onCreate(SQLiteDatabase db)
                                {
                                        db.execSQL("create table contacts( id integer primary key
                                          autoincrement,name  text not null,email text not null)");
                                }
                                @Override
                                public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2)
                                {
                                        // TODO Auto-generated method stub
                                }
             }
```

5.6 Basic operations of SQLite Database

- **Update a Contact**

MainActivity.java

```
DBHelper helper = new DBHelper();
SQLiteDatabase db = helper.getWritableDatabase();

db.execSQL("update contacts set name ='Piyush' where id = 2"); //update name

Cursor c = db.query(true, "contacts", new String[]{"id", "name", "email"}, null, null, null, null,
    null, null);   //select query

c.moveToFirst();
do
{       //displaying one by one table records in Toast
      Toast.makeText(this, "id: " + c.getString(0) + "\n" +  "Name: " + c.getString(1) + "\n"  +
            "Email:  " + c.getString(2), Toast.LENGTH_LONG).show();
 }while (c.moveToNext());

 helper.close();
}
}
```

5.6 Basic operations of SQLite Database

- **Delete a Contact**

MainActivity.java

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

             class DBHelper extends SQLiteOpenHelper
             {
                                public DBHelper()
                                {
                                        super(getApplicationContext(),"MyDB",null,1);
                                }
                                @Override
                                public void onCreate(SQLiteDatabase db)
                                {
                                        db.execSQL("create table contacts( id integer primary key
                                          autoincrement,name  text not null,email text not null)");
                                }
                                @Override
                                public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2)
                                {
                                        // TODO Auto-generated method stub
                                }
             }
```

5.6 Basic operations of SQLite Database

- **Delete a Contact**

MainActivity.java

```
DBHelper helper = new DBHelper();
SQLiteDatabase db = helper.getWritableDatabase();

db.execSQL("delete from contacts where id = 2");   //delete record

Cursor c = db.query(true, "contacts", new String[]{"id", "name", "email"}, null, null, null, null,
    null, null);   //select query

c.moveToFirst();
 do
 {       //displaying one by one table records in Toast
       Toast.makeText(this, "id: " + c.getString(0) + "\n" +  "Name: " + c.getString(1) + "\n"  +
             "Email:  " + c.getString(2), Toast.LENGTH_LONG).show();
  }while (c.moveToNext());

  helper.close();
}
}
```

5.7 Android Application Priorities

- Android devices have limited resources, therefore the Android system is allowed to manage the available resources by terminating running processes or recycling Android components.

- If the Android system needs to terminate processes (application) it follows the following priority system.

5.7 Android Application Priorities

Process Status	Description	Priority
Foreground	An application in which the user is interacting with an activity, or which has an service which is bound to such an activity. Also if a service is executing one of its lifecycle methods or a broadcast receiver which runs its onReceive() method.	1
Visible	User is not interacting with the activity, but the activity is still (partially) visible or the application has a service which is used by a inactive but visible activity.	2
Service	Application with a running service which does not qualify for application priority 1 or 2.	3
Background	Application with only stopped activities and without a service or executing receiver. Android keeps them in a least recent used (LRU) list and if requires terminates the one which was least used.	4
Empty	Application without any active components.	5

****