Unit – 6
ListView, GridView and RecyclerView [6 Hrs]

ListView

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.

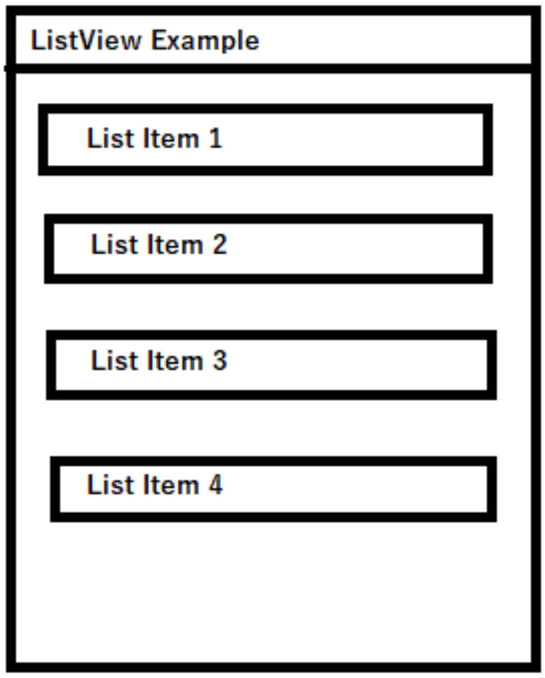


Figure 6-1. A ListView

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can take the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

To display a list, you can include a list view in your layout XML file:

```
<ListView
      android:id="@+id/list_view"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
```

Following are the attributes associated with ListView:

android:divider	Drawable or color to draw between list items.
android:dividerHeight	Height of the divider.
android:entries	Reference to an array resource that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider after each header view.

We can display items in ListView using ArrayAdapter as follows:

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

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file.
- Third argument is a **TextView** used for each string in the array.
- Final argument is an array of strings which will be populated in the TextView.

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

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

Features of ListView:

1. It displays a vertically-scrollable collection of views, where each view is positioned immediately below the previous view in the list.
2. ListView uses Adapter classes which add the content from data source (such as string array, array, database etc.)
3. ListView is a default scrollable which does not use other scroll view.
4. ListView is implemented by importing *android.widget.ListView* class.

Implementing ListView in an Application

Following example will demonstrate ListView. Here we are creating two layout file. One for the ListView and another for list items to be displayed in a ListView. After creating layout files, we use ArrayAdapter class to display String array in a ListView.

So, let's begin by creating two layout files,

listview_example.xml

```
<?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="match_parent">

   <ListView
       android:layout_width="match_parent"
       android:id="@+id/mylist"
       android:layout_height="match_parent" />

</RelativeLayout>
```

listview_items.xml

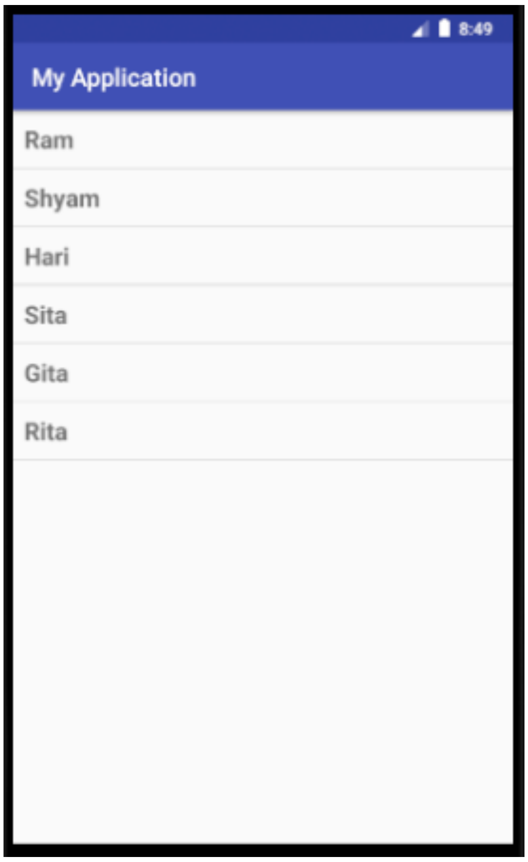
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:id="@+id/text"
        android:textStyle="bold"
        android:layout_margin="10dp" />

</RelativeLayout>
```

Now we are creating a java file to for displaying String array in a ListView.

ListViewExample.java

```
public class ListViewExample extends AppCompatActivity {
    ListView listView;
    @Override
    protected void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.listview_example);

        listView=findViewById(R.id.mylist);
        //creating string array
        String names[]=
              {"Ram","Shyam","Hari","Sita","Gita","Rita"};

        //displaying list using ArrayAdapter
        ArrayAdapter<String> adapter=new ArrayAdapter<String>
              (this,R.layout.listview_items,R.id.text,names);
        listView.setAdapter(adapter);

    }
}
```

Above code produces following output:



Ram
Shyam
Hari
Sita
Gita
Rita

Figure 6-2. Output Demonstrating ListView

Following code can be used for handling clicks in a ListView:

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
                       position, long l) {
        //retrieving value
        String value=adapter.getItem(position);
        //displaying in Toast
       Toast.makeText(getApplicationContext(),value,Toast.LENGTH_SHORT).
                                                              show();

    }
});
```

Creating Custom ListView

After creating simple ListView, android also provides facilities to customize our ListView. Like simple ListView, custom ListView also uses Adapter classes which added the content from data source (such as string array, array, database etc). Adapter bridges data between an AdapterViews and other Views.
To display a more custom view for each item in your dataset, extend **ArrayAdapter** and create and configure the view for each data item in **getView(...).** Example is shown below:

custom_list.xml

```
<?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="match_parent">

    <ListView
        android:layout_width="match_parent"
        android:id="@+id/mylist"
        android:layout_height="match_parent" />

</RelativeLayout>
```

customlist_items.xml

```
<?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="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"
        android:id="@+id/image" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:id="@+id/title"
        android:textStyle="bold"
        android:layout_marginTop="10dp"
        android:layout_toRightOf="@+id/image"
        android:text="Title" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:id="@+id/description"
        android:layout_below="@+id/title"
        android:text="This is description"
        android:layout_toRightOf="@+id/image” />
</RelativeLayout>
```

MyListAdapter.java

```
public class MyListAdapter extends ArrayAdapter<String> {
    Activity context;
    String[] title;
    String[] description;
    int[] image;

    public MyListAdapter(Activity context, String[] title, String[]
               description, int[] image) {
        //ArrayAdapter needs String so we are supplying title
        super(context, R.layout.customlist_items,title);
        this.context=context;
        this.title=title;
        this.description=description;
        this.image=image;
    }

    public View getView(int position, View view, ViewGroup parent) {
        LayoutInflater inflater=context.getLayoutInflater();
        View rowView=inflater.inflate(R.layout.customlist_items,
                                      null,true);

        //wiring widgets
        TextView txtTitle = (TextView) rowView.findViewById(R.id.title);
        ImageView imageView = (ImageView)  rowView.findViewById
                              (R.id.image);
        TextView txtDescription = (TextView) rowView.findViewById
                              (R.id.description);

        txtTitle.setText(title[position]);
        imageView.setImageResource(image[position]);
        txtDescription.setText(description[position]);

        return rowView;

    };
}
```

CustomListExample.java

```
public class CustomListExample extends AppCompatActivity {
    ListView listView;
    @Override
    protected void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.custom_list);
        listView=findViewById(R.id.mylist);

        // creating arrays
        String[] title={
                "Title 1", "Title 2",
                "Title 3", "Title 4"};
        String[] description={
                "This is description 1",
                "This is description 2",
                "This is description 3",
                "This is description 4"
        };
        int[] image={
                //Replace with different images
                R.drawable.ic_launcher,
                R.drawable.ic_launcher,
                R.drawable.ic_launcher,
                R.drawable.ic_launcher
        };

        //passing arrays to constructor of MyListAdapter
        MyListAdapter adapter=new MyListAdapter
                               (this,title,description,image);
        listView.setAdapter(adapter);

    }
}
```

Above code produces following output:

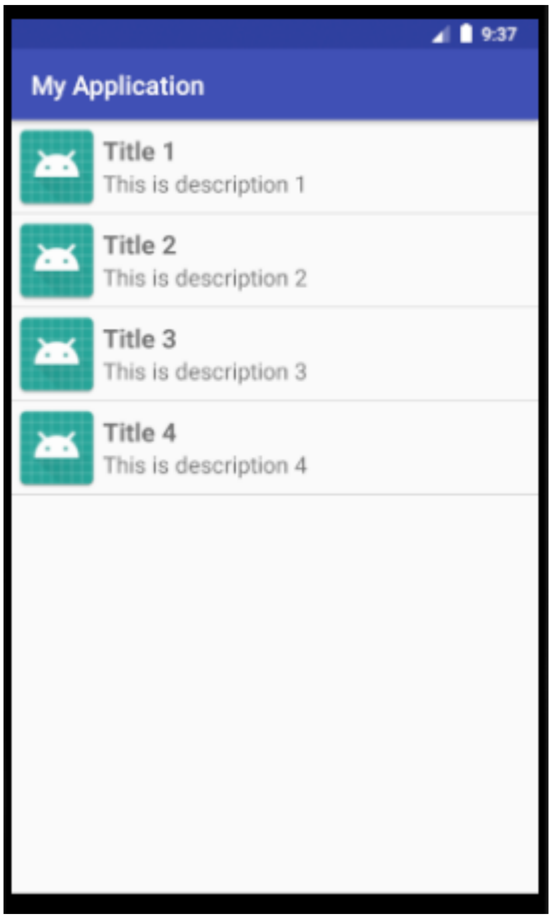


Figure 6-3. Output Demonstrating Custom ListView

Handling clicks in Custom ListView

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
                position, long l) {
        //retrieving title and description
        String title=adapter.title[position];
        String descrption=adapter.description[position];

        //handle click according to position
        if(position==0){
            //indicates first list item
        }
        if(position==1){
            //indicated second list item
        }

    }
});
```

GridView

Android **GridView** shows items in two-dimensional scrolling grid (rows & columns). The items in the grid come from the ListAdapter associated with this view.

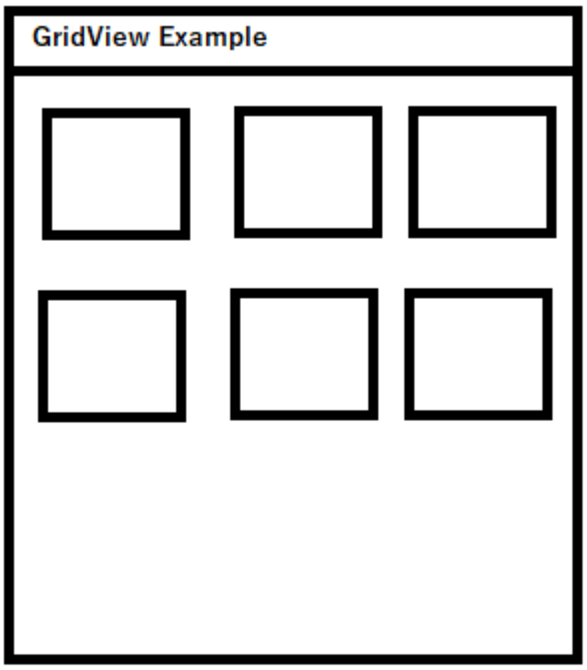


Figure 6-4. GridView having 3 columns

Following are the attributes associated with GridView:

android:columnWidth	Specifies the fixed width for each column.
android:gravity	Specifies the gravity within each cell.
android:horizontalSpacing	Defines the default horizontal spacing between columns.
android:numColumns	Defines how many columns to show.
android:stretchMode	Defines how columns should stretch to fill the available empty space, if any.
android:verticalSpacing	Defines the default vertical spacing between rows.

Features of GridView

1. GridView displays items in two-dimensional scrolling grid.
2. GridView uses Adapter classes which add the content from data source (such as string array, array, database etc.)
3. GridView is a default scrollable which does not use other scroll view.
4. GridView is implemented by importing *android.widget.GridView* class.

Implementing GridView in an Application

Following example will demonstrate GridView. Here we are creating two layout file. One for the GridView and another for grid items to be displayed in a GridView. After creating layout files, we use ArrayAdapter class to display String array in a GridView.

So, let's begin by creating two layout files,

gridview_example.xml

```
<?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="match_parent">

   <GridView
       android:layout_width="match_parent"
       android:id="@+id/mygrid"
       android:numColumns="3"
       android:layout_height="match_parent" />

</RelativeLayout>
```

gridview_items.xml

```
<?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="match_parent">
```

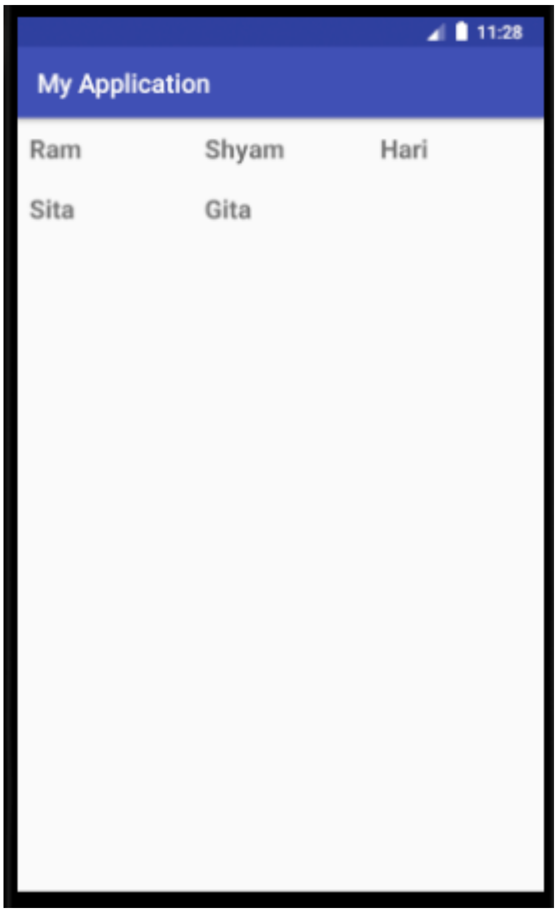
```
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:id="@+id/text"
        android:textStyle="bold"
        android:layout_margin="10dp" />

</RelativeLayout>
```

Now we are creating a java file to for displaying String array in a GridView.

GridViewExample.java

```
public class GridViewExample extends AppCompatActivity {
    GridView gridView;
    @Override
    protected void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.gridview_example);

        gridView =findViewById(R.id.mygrid);
        //creating string array
        String names[]=
              {"Ram","Shyam","Hari","Sita","Gita"};

        //displaying list using ArrayAdapter
        ArrayAdapter<String> adapter=new ArrayAdapter<String>
              (this,R.layout.gridview_items,R.id.text,names);
        gridView.setAdapter(adapter);

    }
}
```

Above code produces following output:



Ram Shyam Hari
Sita Gita

Figure 6-5. Output Demonstrating GridView

Following code can be used for handling clicks in a GridView:

```
gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
                        position, long l) {
        //retrieving value
        String value=adapter.getItem(position);
        //displaying in Toast
       Toast.makeText(getApplicationContext(),value,Toast.LENGTH_SHORT).
                                                              show();

    }
});
```

Creating Custom GridView

After creating simple GridView, android also provides facilities to customize our GridView. Like simple GridView, custom GridView also uses Adapter classes which added the content from data source (such as string array, array, database etc). Adapter bridges data between an AdapterViews and other Views.
To display a more custom view for each item in your dataset, extend **ArrayAdapter** and create and configure the view for each data item in **getView(...).** Example is shown below:

custom_grid.xml

```xml
<?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="match_parent">

   <GridView
       android:layout_width="match_parent"
       android:id="@+id/mygrid"
       android:numColumns="2"
       android:layout_height="match_parent" />

</RelativeLayout>
```

customgrid_items.xml

```xml
<?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="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"
        android:layout_centerHorizontal="true"
        android:id="@+id/image" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:id="@+id/title"
        android:textStyle="bold"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/image"
        android:text="Title" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:id="@+id/description"
        android:text="This is description"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/title" />

</RelativeLayout>
```

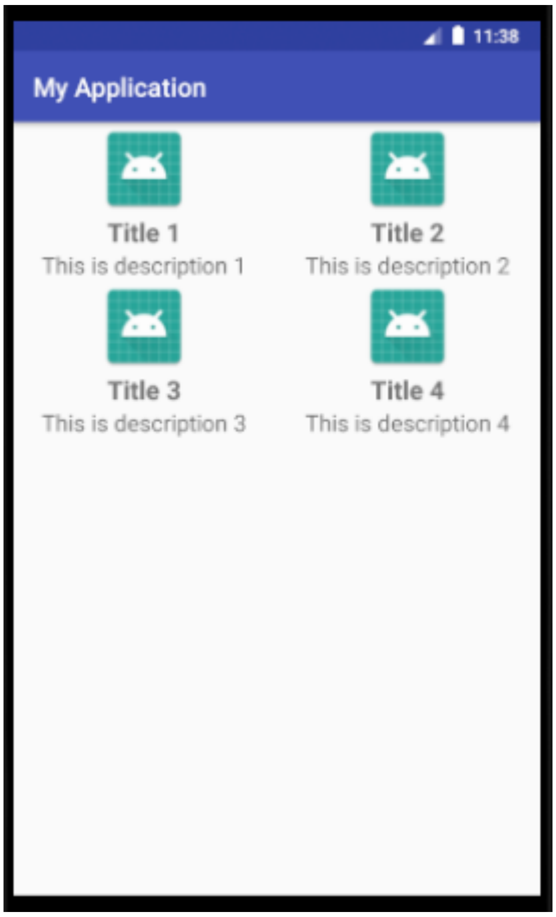
MyGridAdapter.java

```
public class MyGridAdapter extends ArrayAdapter<String> {
    Activity context;
    String[] title;
    String[] description;
    int[] image;

    public MyGridAdapter(Activity context, String[] title, String[]
               description, int[] image) {
        //ArrayAdapter needs String so we are supplying title
        super(context, R.layout.customgrid_items,title);
        this.context=context;
        this.title=title;
        this.description=description;
        this.image=image;
    }

    public View getView(int position, View view, ViewGroup parent) {
        LayoutInflater inflater=context.getLayoutInflater();
        View rowView=inflater.inflate(R.layout.customgrid_items,
                                      null,true);

        //wiring widgets
        TextView txtTitle = (TextView) rowView.findViewById(R.id.title);
        ImageView imageView = (ImageView)  rowView.findViewById
                                 (R.id.image);
        TextView txtDescription = (TextView) rowView.findViewById
                                 (R.id.description);

        txtTitle.setText(title[position]);
        imageView.setImageResource(image[position]);
        txtDescription.setText(description[position]);

        return rowView;

    };
}
```

CustomGridExample.java

```
public class CustomGridExample extends AppCompatActivity {
    GridView gridView;
    @Override
    protected void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.custom_grid);
        gridView =findViewById(R.id.mygrid);

        // creating arrays
        String[] title={
                "Title 1", "Title 2",
                "Title 3", "Title 4"};
        String[] description={
                "This is description 1",
                "This is description 2",
                "This is description 3",
                "This is description 4"
        };
        int[] image={
                //Replace with different images
                R.drawable.ic_launcher,
                R.drawable.ic_launcher,
                R.drawable.ic_launcher,
                R.drawable.ic_launcher
        };

        //passing arrays to constructor of MyListAdapter
        MyGridAdapter adapter=new MyGridAdapter
                                    (this,title,description,image);
        gridView.setAdapter(adapter);

    }
}
```

Above code produces following output:



Title 1
This is description 1
Title 2
This is description 2
Title 3
This is description 3
Title 4
This is description 4

Figure 6-6. Output Demonstrating Custom GridView

Handling clicks in Custom GridView

```
gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
               position, long l) {
        //retrieving title and description
        String title=adapter.title[position];
        String descrption=adapter.description[position];

        //handle click according to position
        if(position==0){
            //indicates first list item
        }
        if(position==1){
            //indicated second list item
        }

    }
});
```

RecyclerView

The RecyclerView widget is a more advanced and flexible version of ListView. If your app needs to display a scrolling list of elements based on large data sets (or data that frequently changes), you should use RecyclerView. RecyclerView is mostly used to design the user interface with the fine-grain control over the lists and grids of android application. It was introduced in *Marshmallow*.

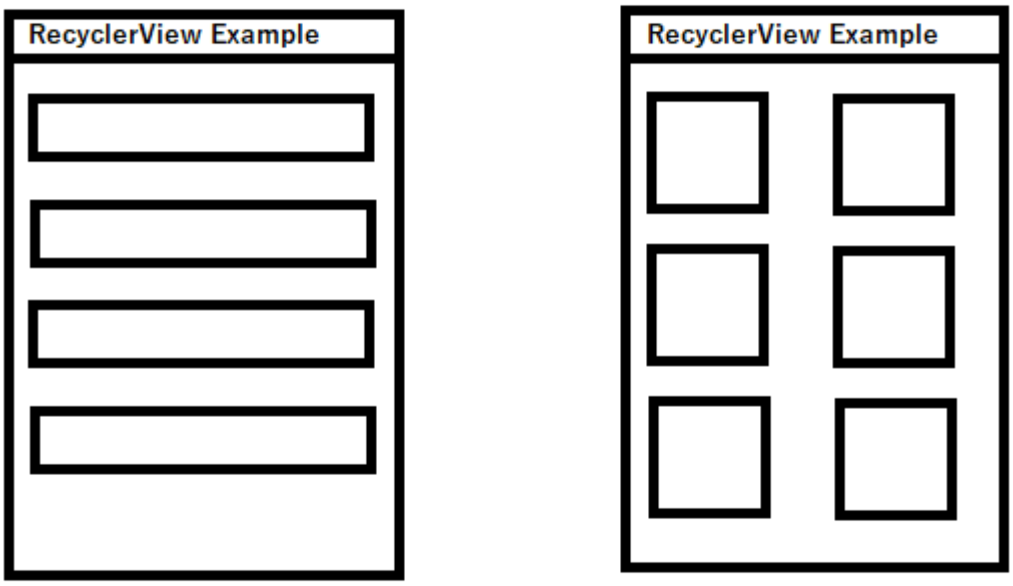


Figure 6-7. RecyclerView Displaying elements in List and Grid

In the RecyclerView model, several different components work together to display your data. The overall container for your user interface is a RecyclerView object that you add to your layout. The RecyclerView fills itself with views provided by a layout manager that you provide. **You can use one of our standard layout managers (such as LinearLayoutManager or GridLayoutManager), or implement your own**.

The views in the list are represented by view holder objects. These objects are instances of a class you define by extending **RecyclerView.ViewHolder**. Each view holder is in charge of displaying a single item with a view. For example, if your list shows music collection, each view holder might represent a single album. The RecyclerView creates only as many view holders as are needed to display the on-screen portion of the dynamic content, plus a few extra. As the user scrolls through the list, the RecyclerView takes the off-screen views and rebinds them to the data which is scrolling onto the screen.

The view holder objects are managed by an adapter, which you create by extending **RecyclerView.Adapter**. The adapter creates view holders as needed. The adapter also binds the view holders to their data. It does this by assigning the view holder to a position, and calling the adapter's **onBindViewHolder()** method. That method uses the view holder's position to determine what the contents should be, based on its list position.

This RecyclerView model does a lot of optimization work so you don't have to:

- When the list is first populated, it creates and binds some view holders on either side of the list. For example, if the view is displaying list positions 0 through 9, the RecyclerView creates and binds those view holders, and might also create and bind the view holder for position 10. That way, if the user scrolls the list, the next element is ready to display.
- As the user scrolls the list, the RecyclerView creates new view holders as necessary. It also saves the view holders which have scrolled off-screen, so they can be reused. If the user switches the direction they were scrolling, the view holders which were scrolled off the screen can be brought right back. On the other hand, if the user keeps scrolling in the same direction, the view holders which have been off-screen the longest can be re-bound to new data. The view holder does not need to be created or have its view inflated; instead, the app just updates the view's contents to match the new item it was bound to.
- When the displayed items change, you can notify the adapter by calling an appropriate **RecyclerView.Adapter.notify…()** method. The adapter's built-in code then rebinds just the affected items.

Following is an attribute associated with RecyclerView:

- **RecyclerView_layoutManager,** used to defining the layout of RecyclerView i.e., **LinearLayout** or **GridLayout**.

Features of RecyclerView

1. RecyclerView widget is a more advanced and flexible version of ListView. So, we can use RecyclerView to display large dataset.
2. RecyclerView contains integrated animations for adding, updating and removing items.
3. RecyclerView enforces the recycling of views by using the ViewHolder pattern.
4. RecyclerView supports both grids and lists.
5. RecyclerView supports vertical and horizontal scrolling.

Implementing RecyclerView in Application

To access the RecyclerView widget, you need to add following dependency to your project as follows:

1. Open the build.gradle file for your app module.
2. Add the support library to the dependencies section

```
dependencies {
    implementation 'com.android.support:recyclerview-v7:28.0.0'
}
```

Now you can add the RecyclerView to your layout file as follows:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Once you have added a RecyclerView widget to your layout, obtain a handle to the object, connect it to a layout manager, and attach an adapter for the data to be displayed:

```
public class MyActivity extends Activity {
    private RecyclerView recyclerView;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager layoutManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_activity);
        recyclerView = (RecyclerView) findViewById
                    (R.id.my_recycler_view);

        // use a linear layout manager
        layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);

        // specify an adapter (see also next example)
        mAdapter = new MyAdapter(myDataset);
        recyclerView.setAdapter(mAdapter);
    }
    // ...
}
```

To feed all your data to the list, you must extend the **RecyclerView.Adapter** class as follows:

```
public class MyAdapter extends
RecyclerView.Adapter<MyAdapter.MyViewHolder> {
     //add stuffs for displaying dataset in recyclerview
}
```

Following example demonstrates RecyclerView. Here, we are creating three arrays for name, address and image. We are going to display these arrays in RecyclerView.

We are starting by creating layout file for placing RecyclerView.

recyclerview_example.xml

```
<?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="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:scrollbars="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</RelativeLayout>
```

Now we are creating layout file for placing RecyclerView items.

recyclerview_items.xml

```
<?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">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"
        android:id="@+id/image"  />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:id="@+id/txtName"
        android:textStyle="bold"
        android:layout_toRightOf="@+id/image"
        android:layout_marginTop="10dp"
        android:text="Name"  />

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:id="@+id/txtAddress"
        android:text="Address"
        android:layout_toRightOf="@+id/image"
        android:layout_below="@+id/txtName" />

</RelativeLayout>
```

Now it’s time to create java files. Here we need to java classes one will be the Activity and another is needed for creating Adapter for RecycerView.

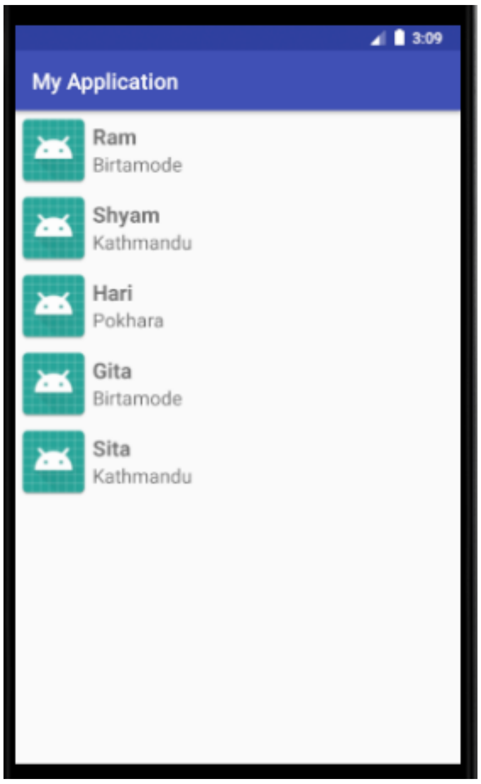
RecyclerViewExample.java

```
public class RecyclerViewExample extends AppCompatActivity {
    RecyclerView recyclerView;
    RecyclerView.Adapter adapter;
    RecyclerView.LayoutManager layoutManager;
    @Override
    protected void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.recyclerview_example);
        recyclerView=findViewById(R.id.recyclerview);

        //creating array
        String[] name={
                "Ram","Shyam","Hari",
                "Gita","Sita"
        };
        String[] address={
                "Birtamode","Kathmandu","Pokhara",
                "Birtamode","Kathmandu"
        };
        int[] image={
                R.drawable.ic_launcher,R.drawable.ic_launcher,
                R.drawable.ic_launcher,R.drawable.ic_launcher,
                R.drawable.ic_launcher
        };

        //setting layout manager
        layoutManager=new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);

        //passing array to Adapter class
        adapter=new RecyclerViewAdapter(this,name,address,image);
        recyclerView.setAdapter(adapter);

    }
}
```

RecyclerViewAdapter.java

```
public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {
    Activity context;
    int[] image;
    String[] name;
    String[] address;

    public RecyclerViewAdapter(Activity context,String[] name,
               String[] address, int[] image){
        this.name=name;
        this.address=address;
        this.image=image;
        this.context=context;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int
               viewType) {
        LayoutInflater layoutInflater = LayoutInflater.from
                                         (context);
        View listItem= layoutInflater.inflate
               (R.layout.recyclerview_items, parent, false);
        ViewHolder viewHolder = new ViewHolder(listItem);
        return viewHolder;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position){
        holder.txtName.setText(name[position]);
        holder.txtAddress.setText(address[position]);
        holder.imageView.setImageResource(image[position]);
    }


    @Override
    public int getItemCount() {
        return name.length;
    }

    public static class ViewHolder extends RecyclerView.ViewHolder{
        TextView txtName,txtAddress;
        ImageView imageView;
        public ViewHolder(View itemView) {
            super(itemView);
            txtName = itemView.findViewById(R.id.txtName);
            txtAddress = itemView.findViewById(R.id.txtAddress);
            imageView = itemView.findViewById(R.id.image);
        }
    }
}
```

Above code produces following output:



Ram
Birtamode
Shyam
Kathmandu
Hari
Pokhara
Gita
Birtamode
Sita
Kathmandu

Figure 6-8. Output Demonstrating RecyclerView

Following code snipped can be used if you want to display items in grid layout.

```
//setting layout manager
layoutManager=new GridLayoutManager(this,3);
//3 represents no. of columns
recyclerView.setLayoutManager(layoutManager);
```

Displaying Data Using MVC Pattern in RecyclerView

For this purpose, we need a model where data is to be stored. So, now I am going to create a model named as **MyData.java** which contains all required variables i.e., name, address and image.

MyData.java

```
public class MyData {

    private String name;
    private String address;
    private int image;


public MyData(String name,String address,int image){
        this.name=name;
        this.address=address;
        this.image=image;
    }
    public String getName(){
        return name;
    }
    public String getAddress(){
        return address;
    }
    public int getImage(){
        return image;
    }
}
```

RecyclerViewExample.java

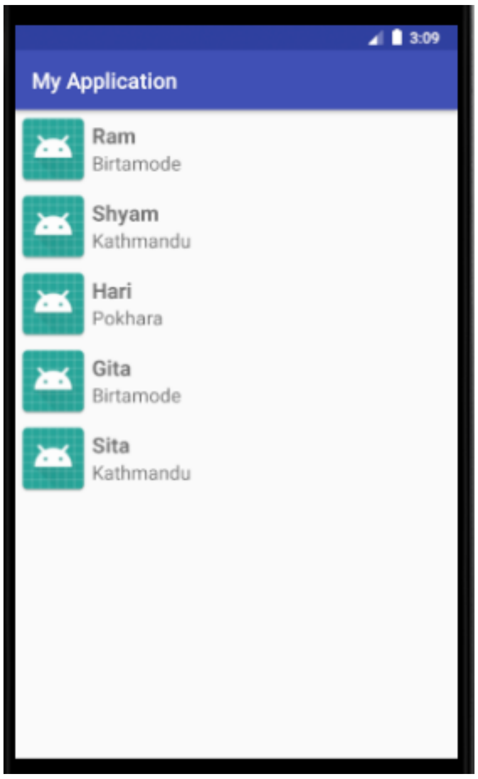
```
public class RecyclerViewExample extends AppCompatActivity {
    RecyclerView recyclerView;
    RecyclerView.Adapter adapter;
    RecyclerView.LayoutManager layoutManager;
    @Override
    protected void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.recyclerview_example);
        recyclerView=findViewById(R.id.recyclerview);

        //filling data in model
        ArrayList<MyData> data=new ArrayList<>();
        data.add(new MyData("Ram","Birtamode",R.drawable.ic_launcher));
        data.add(new MyData("Shyam","Kathmandu",R.drawable.ic_launcher));
        data.add(new MyData("Hari","Pokhara",R.drawable.ic_launcher));
        data.add(new MyData("Gita","Birtamode",R.drawable.ic_launcher));
        data.add(new MyData("Sita","Kathmandu",R.drawable.ic_launcher));

        //setting layout manager
        layoutManager=new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);

        //passing array to Adapter class
        adapter=new RecyclerViewAdapter(this,data);
        recyclerView.setAdapter(adapter);

    }
}
```

RecyclerViewAdapter.java

```
public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {
    Activity context;
    ArrayList<MyData> data;

    public RecyclerViewAdapter(Activity context, ArrayList<MyData> data){
        this.context=context;
        this.data=data;
    }

    @Override
   public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater layoutInflater = LayoutInflater.from(context);
        View listItem= layoutInflater.inflate
```

```
                      (R.layout.recyclerview_items, parent, false);
        ViewHolder viewHolder = new ViewHolder(listItem);
        return viewHolder;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        MyData current=data.get(position);
        holder.txtName.setText(current.getName());
        holder.txtAddress.setText(current.getAddress());
        holder.imageView.setImageResource(current.getImage());
    }

    @Override
    public int getItemCount() {
        return data.size();
    }
    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView txtName,txtAddress;
        ImageView imageView;
        public ViewHolder(View itemView) {
            super(itemView);
            txtName = itemView.findViewById(R.id.txtName);
            txtAddress = itemView.findViewById(R.id.txtAddress);
            imageView = itemView.findViewById(R.id.image);
        }
    }
}
```

Above code will produce following output:



Ram
Birtamode
Shyam
Kathmandu
Hari
Pokhara
Gita
Birtamode
Sita
Kathmandu

Figure 6-9. Output Demonstrating RecyclerView using MVC Pattern

Handling clicks in RecyclerView

```
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    final MyData current=data.get(position);
    holder.txtName.setText(current.getName());
    holder.txtAddress.setText(current.getAddress());
    holder.imageView.setImageResource(current.getImage());

    //handling clicks
    holder.imageView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
             //retieving data
            String name=current.getName();
            String address=current.getAddress();

            Toast.makeText(context,name,Toast.LENGTH_SHORT).show();
        }
    });

}
```

Exercise

1. What do you mean by ListView? Explain its features.
2. What do you mean by GridView? Explain its features.
3. What do you mean by RecyclerView? Explain its features.
4. Differentiate ListView and GridView with example.
5. Differentiate ListView and RecyclerView with example.
6. Why RecyclerView is recommended over ListView? Explain with example.
7. Develop an android application to display id, name and address of 5 students using ListView.
8. Develop an android application to display image, name and address of 5 students using GridView.
9. Write a code snippet for retrieving data from ListView and GridView.
10. Develop an image gallery using GridView. Your gallery should display at least 5 drawable images. If any of the image is clicked it should be displayed in another activity in large size.
11. Develop an android application for the same information given in question no. 7 using RecyclerView.
12. Develop an android application for the same information given in question no. 8 using RecyclerView (use MVC pattern).
13. Develop an android application for the same information given in question no. 10 using RecyclerView.
14. Write a code snippet for retrieving data from RecyclerView.