



# List View and Dialogs

# List View

- A list view allows you to display a list of data that you can then use to navigate through the app.
- **List View** is a **View Group** that is used to display the list of scrollable of items in multiple rows and the list items are automatically inserted to the list using an **adapter**. Each view is positioned immediately below the previous view in the list.
- An adapter acts as a bridge between a view and a data source. It reads data from various data sources, converts it into a View and provide it to the linked Adapter view to create UI components.



- A list view is an adapter view that does not know the details, such as type and contents, of the views it contains. Instead, list view requests views on demand from a List Adapter as needed, such as to display new views as the user scrolls up or down.

## Define List View in XML

```
<ListView  
    android:id="@+id/"  
    android:layout_width=""  
    android:layout_height=""  
    android:divider="@color/"  
    android:dividerHeight="" />
```

- ❖ android:divider: we can specify a divider between List items. A drawable or any color can be specified as a value for this attribute.
- ❖ android:dividerHeight: used to specify height of the divider.  
when a second activity is started, it is placed on the top of the stack and the previous activity will be pushed down.

## Organize our Activity

❖ It is better to partition our ideas in to three different types of activity.

- ✓ Top-level Activities

- ✓ Category Activities

- ✓ Detail/edit Activities

❖ **Top-level Activities**: contains the things that are most important to the user, and gives them an easy way of navigating to those things.

In most apps, the first activity the user sees will be a top-level activity.

❖ **Category Activities**: show the data that belongs to a particular category, often in a list. These activities are often used to help the user navigate to detail/edit activities.

❖ **Detail/edit Activity**: displays details for a particular record, let the user edit the record or allow the user to enter new records.

# Types of Adapters

❖ The most commonly used types of adapters in android are:

- ✓ Base Adapter
- ✓ Array Adapter
- ✓ Custom Array Adapter
- ✓ Simple Cursor Adapter

❖ **Base Adapter**: is a base class of a general implementation of an Adapter that can be used in ListView, GridView, Spinner etc.

we use when we need a customized list in a ListView or customized grids in a GridView. Base Adapter can be extended to create a Custom Adapter for displaying a custom list item.

```
public class CustomAdapter extends BaseAdapter {  
    @Override public int getCount() { return 0; }  
    @Override public Object getItem(int i) { return null; }  
    @Override public View getView(int i, View view,  
        ViewGroup viewGroup) { return null; }  
}
```

## Cont'd

- ❖ **Array Adapter**: is a type of adapter that specializes in working with arrays.

- ❖ Array Adapter is used when

  - ✓ Our data source is array

  - E.g., List of phone contacts, countries or name.

- ❖ By default, Array Adapter expects a layout with a single TextView.

  - Array Adapter creates a view for each item by calling toString() on each item and place the content in a TextView.

- ❖ Array Adapter constructor takes the following parameters:

```
ArrayAdapter<String> adapter
```

```
=new ArrayAdapter<String> (this, android. layout. simple_list_item1, myStringArray);
```

## Cont'd (Array Adapter)

- ❖ Context: the reference of current class
- ❖ Resource: the layout that contains a TextView for each string in the array.
- ❖ Objects: the String array.

Then we can simply call SetAdapter() on our ListView:

```
ListView listView = binding.your listview id;  
listView.setAdapter(adapter);
```

Try to create list view that displays list of contacts name  
`String [] contacts_name={};`

- ❖ N.B: For complex layouts or if there is need to customize our list view or grid view Array Adapter is not suitable rather use custom adapters.



- ❖ **Custom Array Adapters:** ArrayAdapter is an implementation of BaseAdapter, so if we want more customization then we can create a custom adapter and extend ArrayAdapter on that class.

For e.g.

if we want to create a view something other than a TextView like displaying an image with text for each item, we extend the ArrayAdapter class and override getView() to return the type of view you want for each item.

- ❖ **Simple Cursor Adapter:** An easy adapter to map columns from a cursor to TextViews or ImageViews defined in an XML file. You can specify which columns you want, which views you want to display the columns, and the XML file that defines the appearance of these views.

✓ Used when our data comes from database.

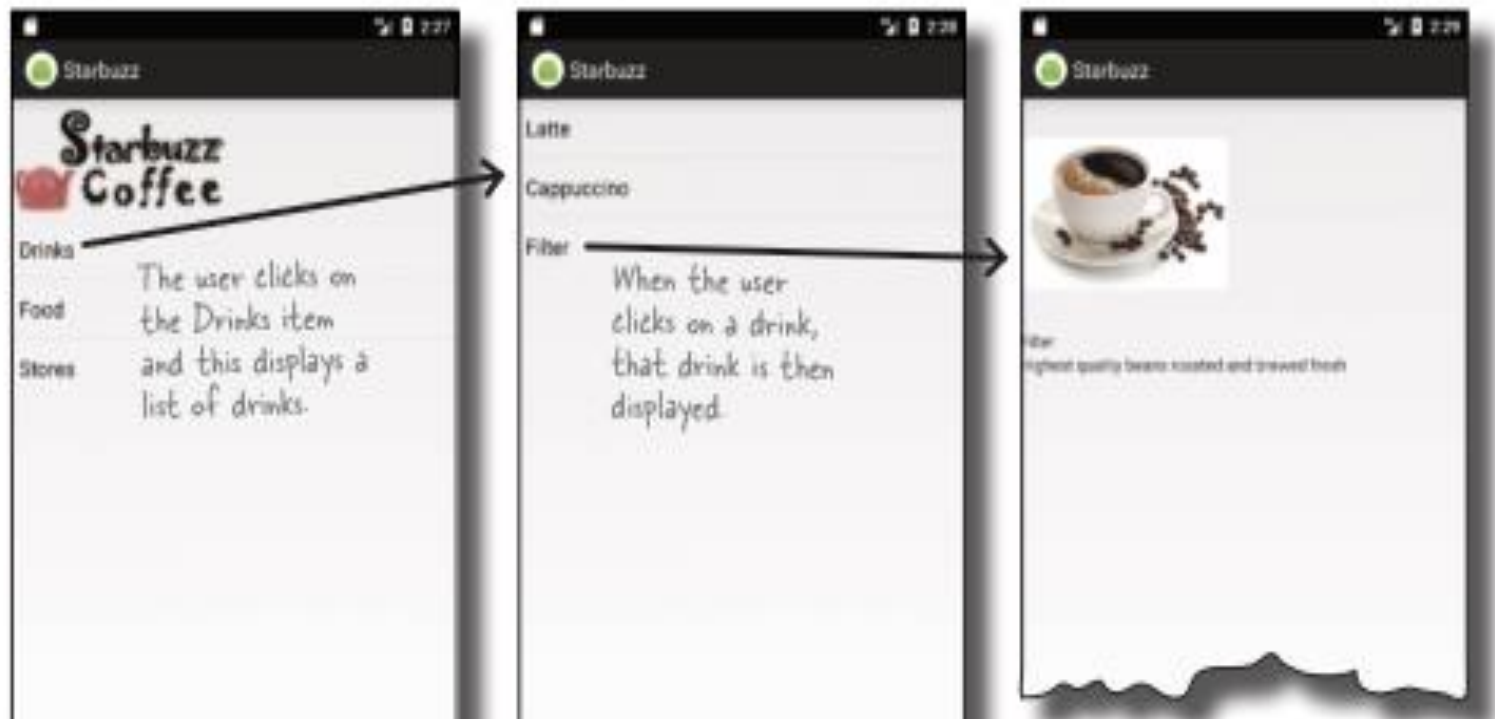
- ❖ SimpleCursorAdapter Constructor takes the following parameters

SimpleCursor Adapteradapter

```
=newSimpleCursorAdapter(this,R.layout.simple_list_item_1,cursor,fromc  
olumns,toViews,0);
```

fromcolumns and toviews arrays tells which columns in the cursor to match to which views.

# Try this



# Top level Activity

## ❖ XML

```
<LinearLayout
    android:orientation="vertical">
    <ImageView
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:src="@drawable/starbuzz_logo"
        android:contentDescription="@string/starbuzz_logo" />
    <ListView
        android:id="@+id/list_options"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/options" />
```

### Strings.xml

```
<string-array name="options">
    <item>Drinks</item>
    <item>Food</item>
    <item>Stores</item>
</string-array>
```

## ❖ Java

```
        //onCreatemethod
        AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener(){
                public void onItemClick(AdapterView<?> listView,
                    View itemView,
                    int position,
                    long id) {
                    if (position == 0) {
                        Intent intent = new Intent(TopLevelActivity.this,
                            DrinkCategoryActivity.class);
                        startActivity(intent);
                    }
                }
            };

        //Add the listener to the list view
        ListView listView = (ListView) findViewById(R.id.list_options);
        listView.setOnItemClickListener(itemClickListener);
    }
}
```

## Drink Category Activity

### ❖ XML

```
<LinearLayout
    android:orientation="vertical">
    <ListView
        android:id="@+id/list_drinks"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

To hold our drink data, we will create pure java class

Refer the code Page no. 256 (The Drink Class).

# Cont'd

❖ Java:

```
//oncreate method
ArrayAdapter<Drink> listAdapter = new ArrayAdapter<> (
    this,
    android.R.layout.simple_list_item_1,
    Drink.drinks);
ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
listDrinks.setAdapter(listAdapter);
ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
listDrinks.setAdapter(listAdapter);
AdapterView.OnItemClickListener itemClickListener =
    new AdapterView.OnItemClickListener(){
    public void onItemClick(AdapterView<?> listDrinks,
        View itemView,
        int position,
        long id) {
        //Pass the drink the user clicks on to DrinkActivity
        Intent intent = new Intent(DrinkCategoryActivity.this,
            DrinkActivity.class);
        intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
        startActivity(intent);
    }
};
//Assign the listener to the list view
listDrinks.setOnItemClickListener(itemClickListener);
```

N.B. In List View **position** and **id** refers to same value in most cases. One of the scenarios they differ is when list view contains header. In that case the position starts counting from header whereas id start counting from list view items. In the above example we use id for passing the list item information.

## Drink Detail Activity

### ❖XML

```
<LinearLayout
    android:orientation="vertical">
    <ImageView
        android:id="@+id/photo"
        android:layout_width="190dp"
        android:layout_height="190dp" />
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/description"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

# Cont'd

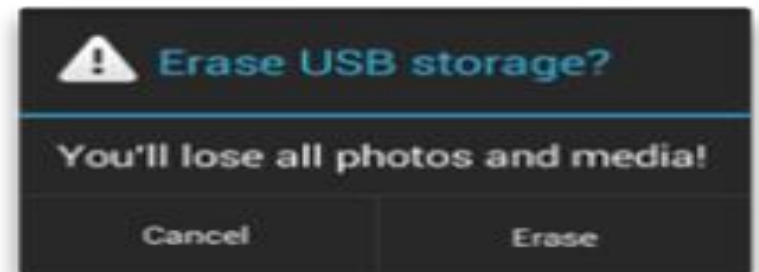
## ❖ Java

```
public static final String EXTRA_DRINKID = "drinkId";  
int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);  
    Drink drink = Drink.drinks[drinkId];  
        //Populate the drink name  
    TextView name = (TextView) findViewById(R.id.name);  
        name.setText(drink.getName());  
        //Populate the drink description  
    TextView description = (TextView) findViewById(R.id.description);  
        description.setText(drink.getDescription());  
        //Populate the drink image  
    ImageView photo = (ImageView) findViewById(R.id.photo);  
        photo.setImageResource(drink.getImageResourceId());  
        photo.setContentDescription(drink.getName());
```



# Dialogs

- ❖ is small window that prompts the user to make a decision or enter additional information's.
- ❖ Dialog boxes in Android are partially transparent, floating Activities or Fragments that partially obscure the user interface.
- ❖ A dialog doesn't fill the screen and is normally used for modal events that require users to take an action before they can proceed.



## Why Dialogs?

- ❖ Dialogs can be used in different ways
  - ✓ Can be used to help users answers some questions
  - ✓ To make Selection
  - ✓ Display warning or error messages
  - ✓ Confirm Actions

## Types of Dialogs

- ❖ The Dialog class is the base class for dialogs but we have to instantiate the subclasses of Dialog rather than using directly.
- ❖ In android the most commonly used types of dialogs are:
  - ✓ Alert Dialog
  - ✓ Progress Dialog (Progress Bar)
  - ✓ Date and Time Picker
  - ✓ Custom Dialog

**Alert Dialog:** shows alert messages and gives the answer in the form of yes or no. According to our response the next step is processed.

- ❖ Android Alert Dialog is built with three fields:
  - ✓ Title
  - ✓ Message (Content Area)
  - ✓ Action Button



**Title:** This is optional and should be used only when the content area is occupied by a detailed message, a list or custom layout.

We use setTitle () method for displaying Alert Dialog box Title, setIcon () is used to set the icon before the title.

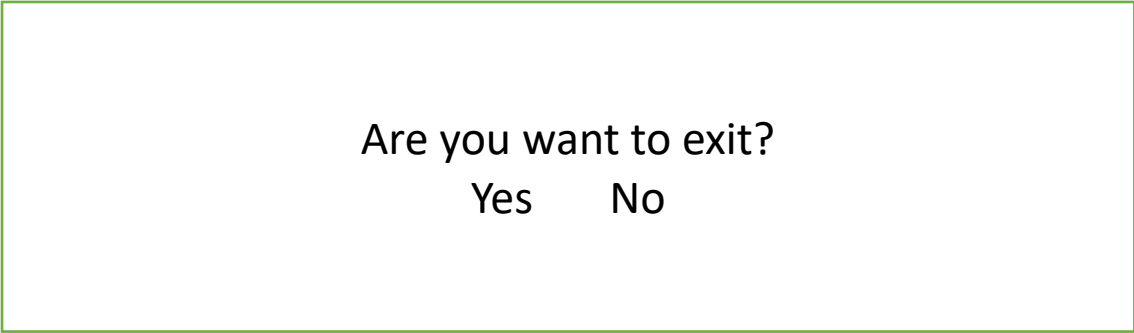
**Message (Content) Area:** this can display a message, a list, or other custom layout.

We use setMessage () method for displaying the message.

**Action Button:** There should be no more than three action buttons in a dialog.

- ✓ Positive: used to accept and continue with the action (the “Ok” or “Yes” action).
- ✓ Negative: used to cancel the action (“No” action)

- ✓ Neutral: used when the user may not want to proceed with the action, but doesn't necessarily want to cancel (“Remind Me Later” action).
- ❖ We can add a list in Alert Dialog
  - ✓ A traditional single-choice list
  - ✓ A single-choice list (Radio Buttons)
  - ✓ A multiple-choice list (Check Boxes)
- ❖ `setItems()` method can be used to add our list items.
- ❖ `setSingleChoiceItems()` used to create a list of Radio Buttons.
- ❖ `setMultipleChoiceItems()` used to create a list of Check Boxes.



Are you want to exit?  
Yes    No

## ❖ Java

```
AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
    builder.setTitle("Login Alert")
        .setMessage("Are you sure, you want to exit ?")
        .setCancelable(false)
        .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                MainActivity.this.finish();
            }
        })
        .setNegativeButton("No", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
    //Creating dialog box
    AlertDialog dialog = builder.create();
    dialog.show();
});
```

**Progress Dialog:** is used to display the status of work being done like analyzing status of work or downloading a file.

- ❖ By default, the Progress Bar will be displayed as a Spinning wheel, if we want to display it in a horizontal bar, we have to change the style property to horizontal like:

style="?android:attr/progressBarStyleHorizontal".

- ❖ Progress bar supports two types of modes to show the progress.

- ✓ Determinate
- ✓ Indeterminate

- ❖ **Determinate:** progress mode is used in progress bar when we want to show the quantity of progress has occurred. For example, the percentage of file downloaded or percent remaining of an audio file that is playing.

To indicate Determinate progress, we set the style of the progress bar to **progressBarStyleHorizontal** and set the amount of progress using **android:progress** attribute.

❖ Indeterminate: progress mode is used in progress bar when we don't know how long an operation will take or how much work has done.

In indeterminate mode the actual progress will not be shown, only the cyclic animation will be shown to indicate that some progress is happening.

❖ Progress Bar is different from Progress Dialog

- ✓ Progress Bar is a View, which can be used in the layout to show some progress while the user may still interact with other parts.

- ✓ Progress Dialog is a Dialog with built in Progress Bar (deprecated in API 26). It is used when we want to prevent the user from interacting with the application. The Dialog freezes the user from doing anything until it is dismissed.



## Indeterminate Mode Example:

### Using Progress Dialog

ProgressDialog loading;

```
loading= ProgressDialog(context,title,message,indeterminate(boolean), cancelable);
```

### Using Progress Bar

```
<ProgressBar  
    android:id="@+id/simpleProgressBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:visibility="invisible"  
    android:layout_centerHorizontal="true"/>
```

In our Activity

```
final ProgressBar simpleProgressBar = (ProgressBar) findViewById(R.id.simpleProgressBar);  
Button startButton = (Button) findViewById(R.id.startButton);  
    // perform click event on button  
startButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // visible the progress bar  
        simpleProgressBar.setVisibility(View.VISIBLE);  
    }  
});
```

## Determinate Mode Example:

```
progressDialog = new ProgressDialog(MainActivity.this);
progressDialog.setMax(100); // Progress Dialog Max Value
progressDialog.setMessage("Loading..."); // Setting Message
progressDialog.setTitle("ProgressDialog"); // Setting Title

progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL); //
    Progress Dialog Style Horizontal
progressDialog.show(); // Display Progress Dialog
progressDialog.setCancelable(false);
    new Thread (new Runnable() {
        @Override
        public void run() {
            try {
                while (progressDialog.getProgress() <=
                    progressDialog.getMax()) {
                    Thread.sleep(200);
                    progressDialog.incrementProgressBy(2);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }).start();
}
```

**Date and Time Picker Dialog:** A dialog that is used to select date and time.

**Custom Dialog:** To create a Custom Dialog we need to create a Custom Layout for the Dialog window with Layout and Widget elements.

We set the Custom Layout as the dialog content view:

```
Context mContext = getApplicationContext();
AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
    dialog.setView(inflated layout);
    dialog.setTitle("Custom Dialog");
    TextView text = (TextView) dialog.findViewById(R.id.text);
    text.setText("Hello, this is a custom dialog!");
    ImageView image = (ImageView) dialog.findViewById(R.id.image);
    image.setImageResource(R.drawable.android);
```