**BaseAdapter with Android ListView**

ListView is a ViewGroup that displays a list of vertically scrollable items. The list items are automatically inserted into the list using an adapter that is connected to a source, such as an array or a database query, and each item is converted into a row in the ListView.

**BaseAdapter**

BaseAdapter, as it's name implies, is the base class for so many concrete adapter implementations on Android. It is abstract and therefore, cannot be directly instantiated.

If your data source is an ArrayList or array, we can also use the [ArrayAdapter construct as an alternative](https://github.com/codepath/android_guides/wiki/Using-an-ArrayAdapter-with-ListView). Note that ArrayAdapter itself extends from BaseAdapter.

**Using the BaseAdapter**

To use the BaseAdapter with a ListView, a concrete implementation the BaseAdepter class that implements the following methods must be created:

* int getCount()
* Object getItem(int position)
* long getItemId(int position)
* View getView(int position, View convertView, ViewGroup parent)

Before we create our custom BaseAdapter implementation, we need to create the layout for the ListView row and also a model for the items in the ListView.

### Create model class

Each of our ListView rows will conatain an Item name and Item description, so our Model class is as follows:

**Item.java**

public class Item {

private String itemName;

private String itemDescription;

public Item(String name, String description) {

this.itemName = name;

this.itemDescription = description;

}

public String getItemName() {

return this.itemName;

}

public String getItemDescription() {

return itemDescription;

}

}

### ListView row layout

The xml file for the rows of the listview created in the res/layout folder is shown below: **layout\_list\_view\_row\_items.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">

<TextView

android:id="@+id/text\_view\_item\_name"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

<TextView

android:id="@+id/text\_view\_item\_description"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

</LinearLayout>

### Create the custom BaseAdapter implementation

public class CustomListAdapter extends BaseAdapter {

private Context context; //context

private ArrayList<Item> items; //data source of the list adapter

//public constructor

public CustomListAdapter(Context context, ArrayList<Item> items) {

this.context = context;

this.items = items;

}

@Override

public int getCount() {

return items.size(); //returns total of items in the list

}

@Override

public Object getItem(int position) {

return items.get(position); //returns list item at the specified position

}

@Override

public long getItemId(int position) {

return position;

}

@Override

public View getView(int position, View convertView, ViewGroup parent) {

// inflate the layout for each list row

if (convertView == null) {

convertView = LayoutInflater.from(context).

inflate(R.layout.layout\_list\_view\_row\_items, parent, false);

}

// get current item to be displayed

Item currentItem = (Item) getItem(position);

// get the TextView for item name and item description

TextView textViewItemName = (TextView)

convertView.findViewById(R.id.text\_view\_item\_name);

TextView textViewItemDescription = (TextView)

convertView.findViewById(R.id.text\_view\_item\_description);

//sets the text for item name and item description from the current item object

textViewItemName.setText(currentItem.getItemName());

textViewItemDescription.setText(currentItem.getItemDescription());

// returns the view for the current row

return convertView;

}

}

### Using the custom Adapter

The adapter can simply be used by instantiating it with the required paramters and set as the listview's adapter.

// Setup the data source

ArrayList<Item> itemsArrayList = generateItemsList(); // calls function to get items list

// instantiate the custom list adapter

CustomListAdapter adapter = new CustomListAdapter(this, itemsArrayList);

// get the ListView and attach the adapter

ListView itemsListView = (ListView) findViewById(R.id.list\_view\_items);

itemsListView.setAdapter(adapter);

### ListView Optimization

To optimize the listview's performance, a new row layout should be inflated only when convertView == null. This is because the adapter's getView method is called whenever the listview needs to show a new row on the screen. The convertView gets recycled in this process. Hence, the row layout should be inflated just once when convertView == null, and it contents should be updated on subsequent getView calls, instead of inflating a new row on each call which is very expensive.

### Optimization using the ViewHolder pattern

One of the most common operation in Android is finding an inner view inside an inflated layout. This is usually done using the findViewById method. This operation is not trivial especially when the lisview has to frequently call getView when the list is scrolling.

The aim of the ViewHolder pattern, is to reduce the number of findViewById calls in the adapter's getView. A ViewHolder is practically a lightweight inner class that holds direct references to all inner views from a row. It is then stored as a **tag** in the row's view after inflating it. This way you’ll only have to use findViewById() when the layout is first created. Here’s a new implementation of our adapter's getView with View Holder pattern applied:

@Override

public View getView(int position, View convertView, ViewGroup parent) {

ViewHolder viewHolder;

if (convertView == null) {

convertView = LayoutInflater.from(context).inflate(R.layout.layout\_list\_view\_row\_items, parent, false);

viewHolder = new ViewHolder(convertView);

convertView.setTag(viewHolder);

} else {

viewHolder = (ViewHolder) convertView.getTag();

}

Item currentItem = (Item) getItem(position);

viewHolder.itemName.setText(currentItem.getItemName());

viewHolder.itemDescription.setText(currentItem.getItemDescription());

return convertView;

}

private class ViewHolder {

TextView itemName;

TextView itemDescription;

public ViewHolder(View view) {

itemName = (TextView)view.findViewById(R.id.text\_view\_item\_name);

itemDescription = (TextView) view.findViewById(R.id.text\_view\_item\_description);

}

}