

ITIS/CS 4180/5180

Mobile App. Development

ListView and Adapters
Mohamed Shehab

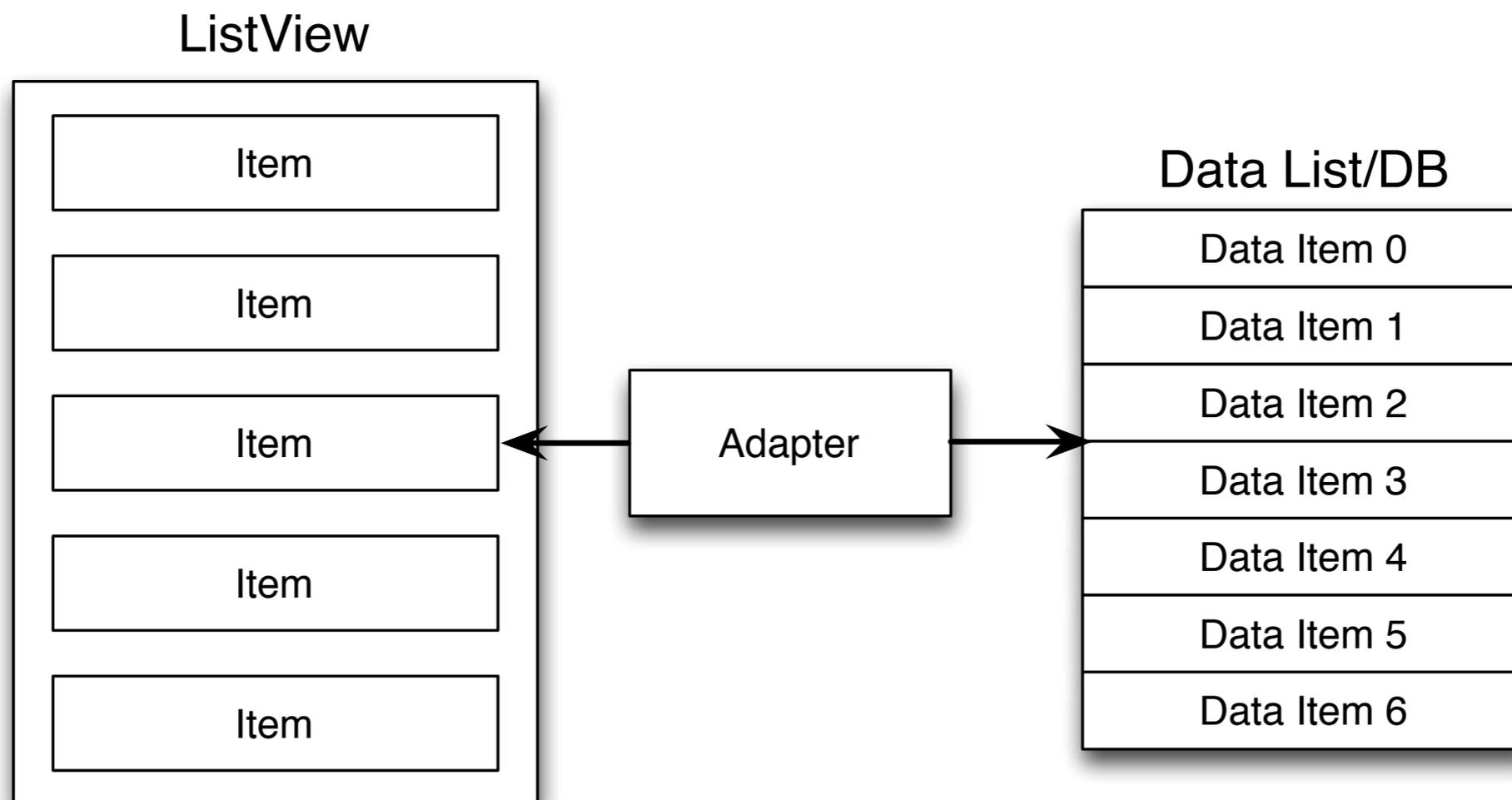


Simple ListView

- The Android ListView is used to display elements in a list.
- The user can scroll through the list of displayed items, and usually upon clicking an item this triggers something.
- Given the list of data items, to display
 - You need to prepare an Adapter.
 - Pass the created Adapter to the ListView.
 - Also you need to set the layout for each item displayed in the ListView.

Simple ListView

- Adapter: will retrieve the data from the data list or database result provided. It will also render the items in the ListView.
- The Adapter classes are bridging classes that bind data to user-interface Views.



Simple ListView

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

    <ListView
        android:id="@+id/mylist"
        android:layout_width="match_parent"
        android:layout_height="336dp"
        android:layout_weight="0.95" >
    </ListView>

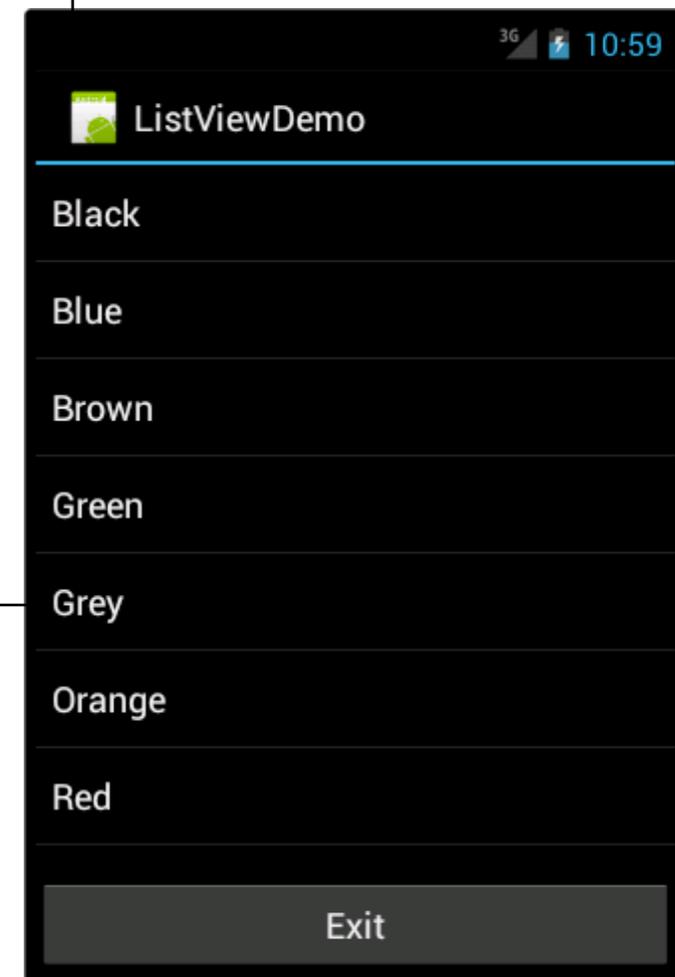
    <Button
        android:id="@+id/exitButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Exit" />

</LinearLayout>
```

Simple ListView

```
public class SimpleListViewActivity extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.layout1);  
  
        ListView myListView = (ListView) findViewById(R.id.mylist);  
        String[] values = new String[] {"Black", "Blue", "Brown", "Green",  
            "Grey", "Orange", "Red", "White", "Yellow"};  
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
            android.R.layout.simple_list_item_1, android.R.id.text1, values);  
        myListView.setAdapter(adapter);  
  
        Button b = (Button) findViewById(R.id.exitButton);  
        b.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                finish();  
            }  
        });  
    }  
}
```

In this example we use the ArrayAdapter



Note the use of the Android layout and setting the text1 to the passed values

Simple ListView

- To capture the selection provided by the user, the ListView provides an interface for setting an on-click listener:

```
public class SimpleListViewActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout1);

        ListView myListView = (ListView) findViewById(R.id.mylist);
        String[] values = new String[] {"Black", "Blue", "Brown", "Green",
            "Grey", "Orange", "Red", "White", "Yellow"};
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, android.R.id.text1, values);
        myListView.setAdapter(adapter);

        myListView.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                Log.d("demo1", (String) ((TextView) view).getText() );
            }
        });

        Button b = (Button) findViewById(R.id.exitButton);
        b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```

parent	The AdapterView where the click happened
view	The view within the AdapterView that was clicked
position	The position of the view in the adapter
id	The row id of the item that was clicked

Simple ListView

- The ArrayAdapter is not limited to receiving a String array
 - You can also pass any Java object as input
 - The default behavior of the Adapter is to call toString() method on each object for the data to be displayed.

Simple ListView

```
public class SimpleListViewObjectActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout1);
        ListView myListView = (ListView) findViewById(R.id.mylist);
        Color[] values = new Color[]{new Color("Black"),
            new Color("Blue"), new Color("Brown"), new Color("Green"),
            new Color("Grey"), new Color("Orange"), new Color("Red"),
            new Color("White"), new Color("Yellow")};
        ArrayAdapter<Color> adapter = new ArrayAdapter<Color>(this,
            android.R.layout.simple_list_item_1, android.R.id.text1, values);
        myListView.setAdapter(adapter);

        myListView.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                Log.d("demo1", (String) ((TextView) view).getText() );
            }
        });
        Button b = (Button) findViewById(R.id.exitButton);
        b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }

    public static class Color{
        String c;
        public Color(String c){
            this.c = c;
        }
        public String toString() {
            return c;
        }
    }
}
```

Simple ListView

- The ArrayAdapter allows for dynamic modifications to the underlying data.
 - `add()` method will append a new value on the end of the array data.
 - `insert()` method will add a new value at a specified position within the array data. `remove()` method to remove an object out of the array.
 - To resync the ListView, either call the “`notifyDataSetChanged()`” method of the adapter or set “`setNotifyOnChange(true)`”.
- The ArrayAdapter’s constructor doesn’t copy references to each of the elements in the array or list. Instead, it directly uses the list that’s passed in. So if you are going to use `add()`, `insert()`, you should provide an `ArrayList` not an `Array`.

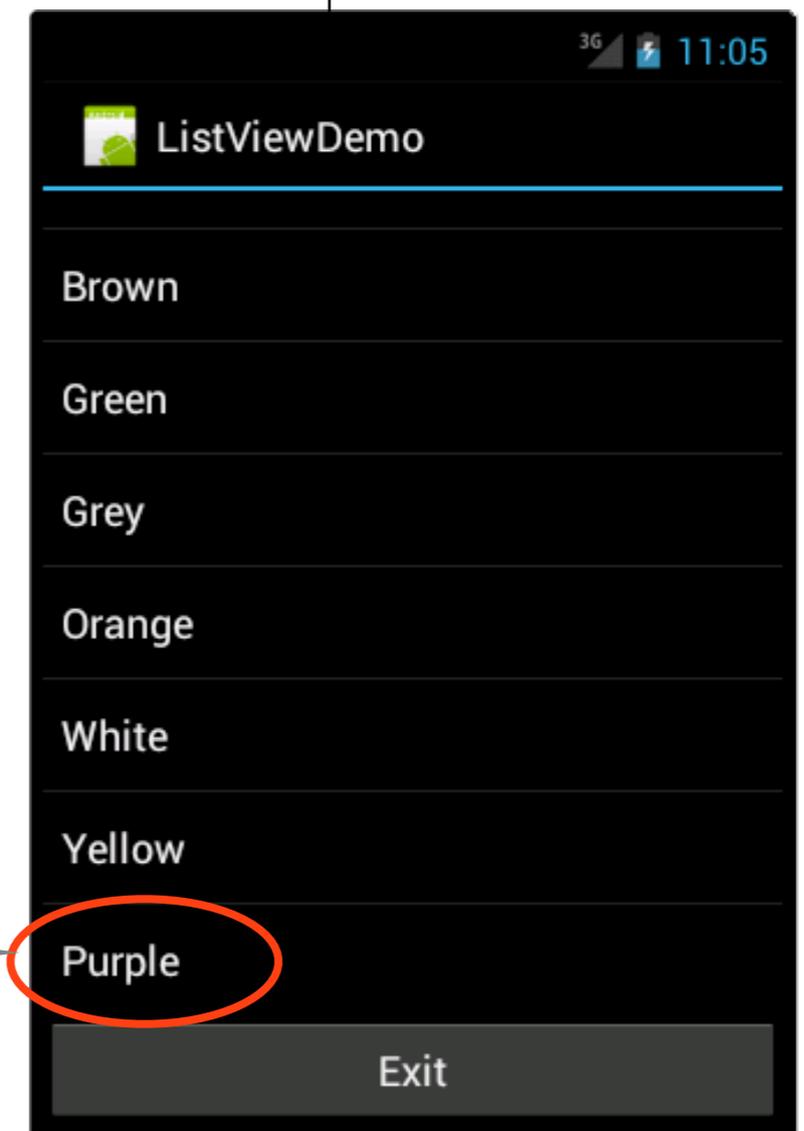
Simple ListView

```
public class ListViewObjectBinding extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout1);
        ListView myListView = (ListView) findViewById(R.id.mylist);
        ArrayList<Color> colors = new ArrayList<Color>();
        colors.add(new Color("Black")); colors.add(new Color("Blue")); colors.add(new Color("Brown"));
        colors.add(new Color("Green")); colors.add(new Color("Grey")); colors.add(new Color("Orange"));
        colors.add(new Color("White")); colors.add(new Color("Yellow"));

        ArrayAdapter<Color> adapter = new ArrayAdapter<Color>(this,
            android.R.layout.simple_list_item_1, android.R.id.text1, colors);
        myListView.setAdapter(adapter);
        adapter.setNotifyOnChange(true);
        adapter.add(new Color("Purple"));
        myListView.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                Log.d("demo1", (String) ((TextView) view).getText() );
            }
        });

        Button b = (Button) findViewById(R.id.exitButton);
        b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
}

public static class Color{
    String c;
    public Color(String c){
        this.c = c;
    }
    public String toString() {
        return c;
    }
}
```



Purple
added

Extending the ArrayAdapter



List Item

Each list item has a layout which describes how the list item should be displayed.

Extending the ArrayAdapter

- For added functionality, you can create your own adapter by extending an existing adapter implementation ex. ArrayAdapter.
 - The most important method is the “getView()” method. This method is called for every line in the ListView. It also creates (inflates) the View to display.
 - The “getView()” method is where the ListView will ask us for a View object to represent each list item it needs to display.

Extending the ArrayAdapter

- To extend the Adapter:
 - Create a custom layout to be used to represent each row of the ListView.
 - Extend the ArrayAdapter Class.
 - Create a Constructor
 - Override the getView() method.
 - You will need to inflate the custom layout using the LayoutInflater Class.

Extending the ArrayAdapter

- Within the `getView()` method you would inflate an XML layout to be used for each of the list items and then set the values of the individualViews in the layout.
 - For inflating an XML layout you can use the system service `LayoutInflater`.
 - This service can get accessed via the Activity or via the `context.getSystemService(Context.LAYOUT_INFLATER_SERVICE)` method call.
- After inflating the XML layout into a view, the individual elements in the layout can be found via the `findViewById()` method call.

Extending the ArrayAdapter

- Below is the custom row layout “item_row_layout.xml”:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="35dp"
    android:orientation="horizontal" >

    <TextView
        android:id="@+id/colorHex"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="Hex"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/colorName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Color Name"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</LinearLayout>
```

Extending the ArrayAdapter

- Below is the extended ArrayAdapter:

```
public static class ColorAdapter extends ArrayAdapter<Color>{
    Context context;
    Color[] objects;

    public ColorAdapter(Context context, Color[] objects) {
        super(context, R.layout.item_row_layout, objects);
        this.context = context;
        this.objects = objects;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View itemRowView = inflater.inflate(R.layout.item_row_layout, parent, false);

        TextView colorName = (TextView) itemRowView.findViewById(R.id.colorName);
        TextView colorHex = (TextView) itemRowView.findViewById(R.id.colorHex);
        colorName.setText(objects[position].colorText);
        colorHex.setText(objects[position].colorHex);
        return itemRowView;
    }
}
```

```
public static class Color{
    public String colorText, colorHex;
    public Color(String colorText, String colorHex){
        this.colorText = colorText;
        this.colorHex = colorHex;
    }
    public String toString() {
        return colorText;
    }
}
```

convertView

- A ListView typically contain more data then the number of displayed rows.
- When the user scrolls some views outside the visible area, the Adapter offers the getView() method the Java objects which represent the rows so that they can be reused for newly visible rows.
 - These views are passed as the convertView parameter of the getView() method.
- A performance optimized adapter assigns the new data to the convertView. This avoids inflating an XML file and creating new Java objects.
- In case no View is available for reuse, Android will pass null to the convertView parameter. Therefore the adapter implementation need to check for this.

Extending the ArrayAdapter

```
public class ListViewCustomAdapter extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout1);

        ListView myListView = (ListView) findViewById(R.id.mylist);
        Color[] values = new Color[]{new Color("Black", "#000000"),
            new Color("Blue", "#0000FF"), new Color("Brown", "#654321"), new Color("Green", "#006600"),
            new Color("Grey", "#BBBBBB"), new Color("Orange", "#FF6600"), new Color("Red", "#FF0000"),
            new Color("White", "#FFFFFF"), new Color("Yellow", "#FFFF00")};

        ColorAdapter adapter = new ColorAdapter(this, values);
        myListView.setAdapter(adapter);

        myListView.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                TextView tv = (TextView) view.findViewById(R.id.colorName);
                Log.d("demo1", tv.getText() + "");
            }
        });

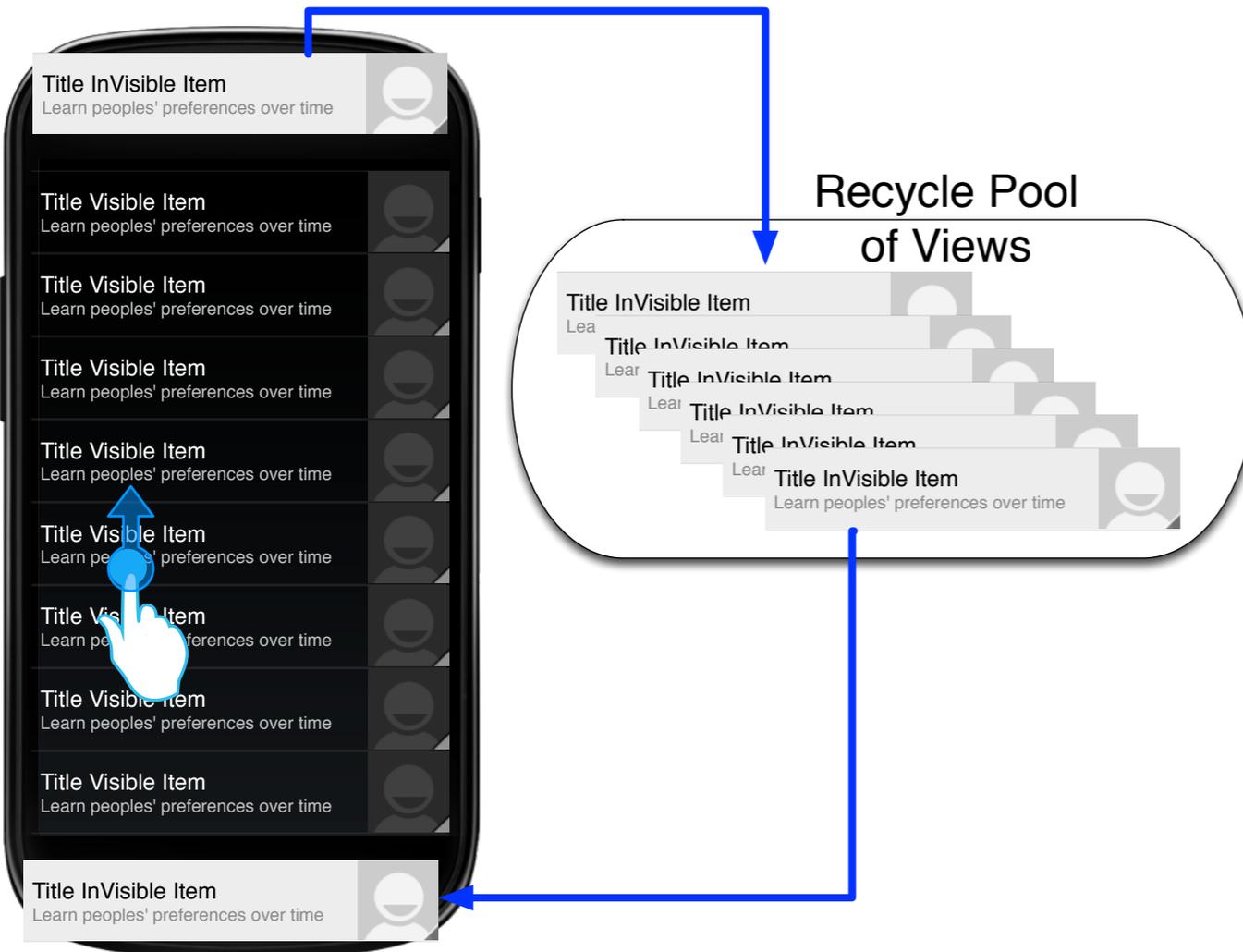
        Button b = (Button) findViewById(R.id.exitButton);
        b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```



Performance Tips for ListViews

- ListViews are designed for scalability and performance which means:
 - ListViews will only render and display a limited number of items visible on the screen.
 - ListViews provide mechanisms to reuse views and reduce the number of new view inflations.
- ListViews provide enable the recycling of non-visible views (recycle pool) as you swipe up or down.
- This approach enables the developers to simply reuse views from the recycle pool instead of inflating a new layout for every row.

Performance Tips for ListViews



Every time ListView needs to show a new row on screen, it will call the `getView()` method from its adapter.

The `convertView` argument is essentially a Recycled View. It will have a non-null value when ListView is asking you recycle the row layout.

```
public static class ColorAdapter extends ArrayAdapter<Color>{  
    .....  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        .....  
    }  
}
```

Performance Tips for ListViews

- When convertView is not null, you should simply update its contents instead of inflating a new row layout.

```
public static class ColorAdapter extends ArrayAdapter<Color>{
    Context context;
    Color[] objects;

    public ColorAdapter(Context context, Color[] objects) {
        super(context, R.layout.item_row_layout, objects);
        this.context = context;
        this.objects = objects;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            convertView = inflater.inflate(R.layout.item_row_layout, parent, false);
        }

        TextView colorName = (TextView) convertView.findViewById(R.id.colorName);
        TextView colorHex = (TextView) convertView.findViewById(R.id.colorHex);
        colorName.setText(objects[position].colorText);
        colorHex.setText(objects[position].colorHex);

        return convertView;
    }
}
```