List-Based Widgets:

Lists, Grids, and Scroll Views

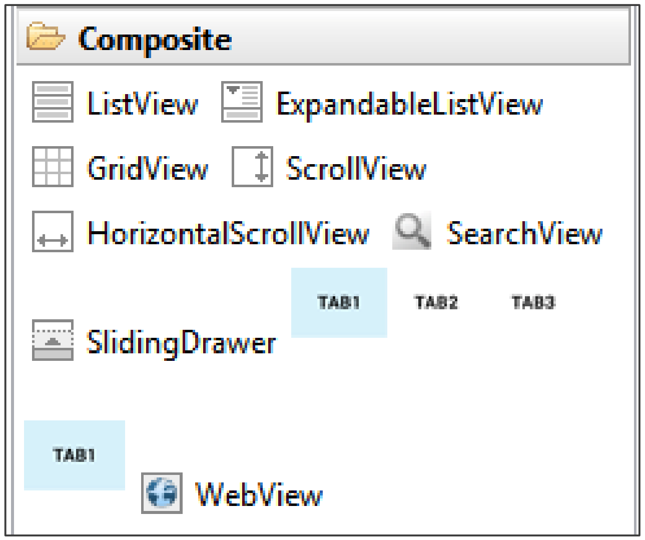
List-Based Widgets

GUI Design for Selection Making

- RadioButtons and CheckButtons are widgets suitable for selecting options offered by a *small* set of choices. They are intuitive and uncomplicated; however they occupy a permanent space on the GUI (which is not a problem when only a few of them are shown)

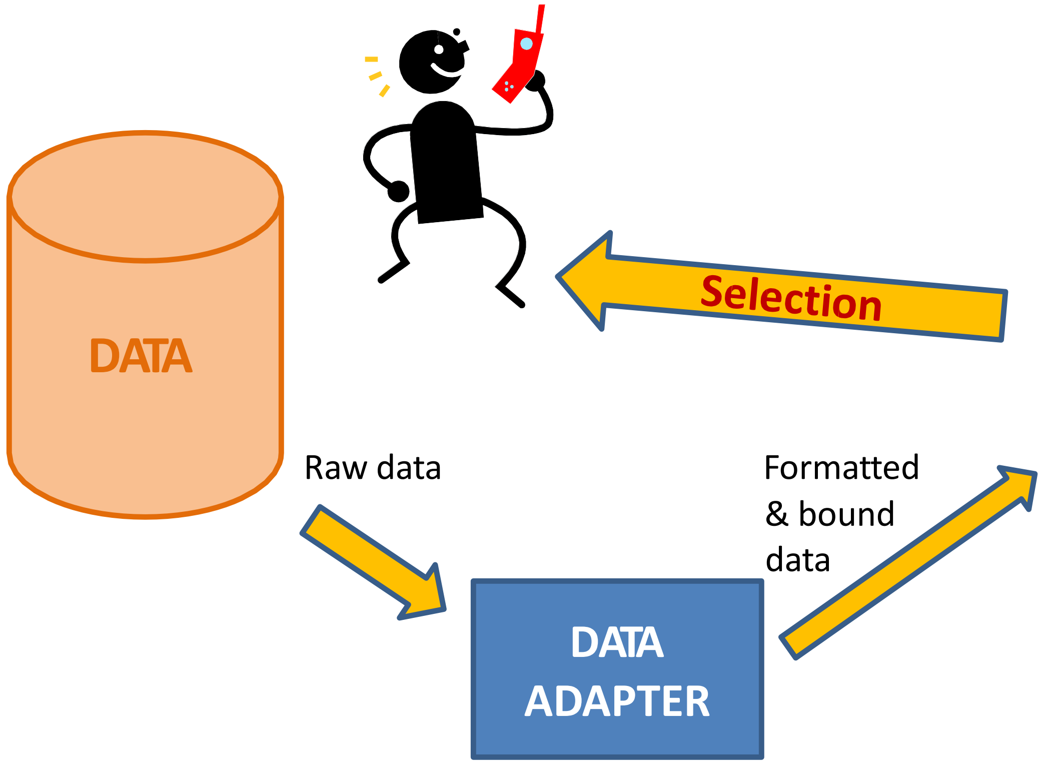
- When the set of values to choose from is large, other Android **List-Based Widgets** are more appropriate.

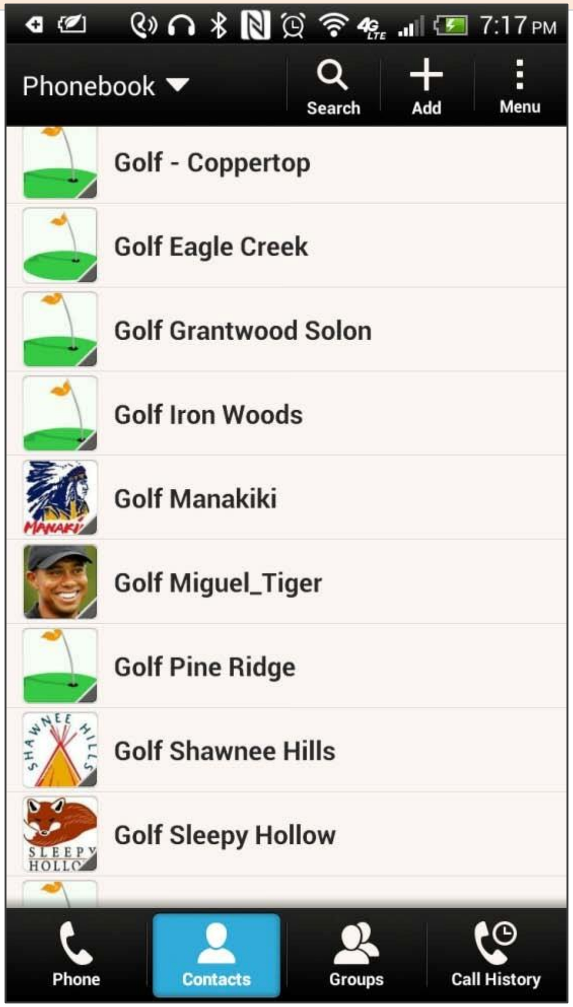
- Example of **List-Based Widgets** include:
  - ***ListViews,***
  - ***Spinner,***
  - ***GridView***
  - ***Image Gallery***
  - ***ScrollViews***, etc.



List-Based Widgets

Showing a large set of choices on the GUI





- The Android ***DataAdapter*** class is used to feed a collection of data items to a *List-Based Widget.*
- The *Adapter 's* raw data may come from a variety of sources, such as small arrays as well as large databases.

Destination layout
Holding a **ListView**

List-Based App = ListView + Data + DataAdapter

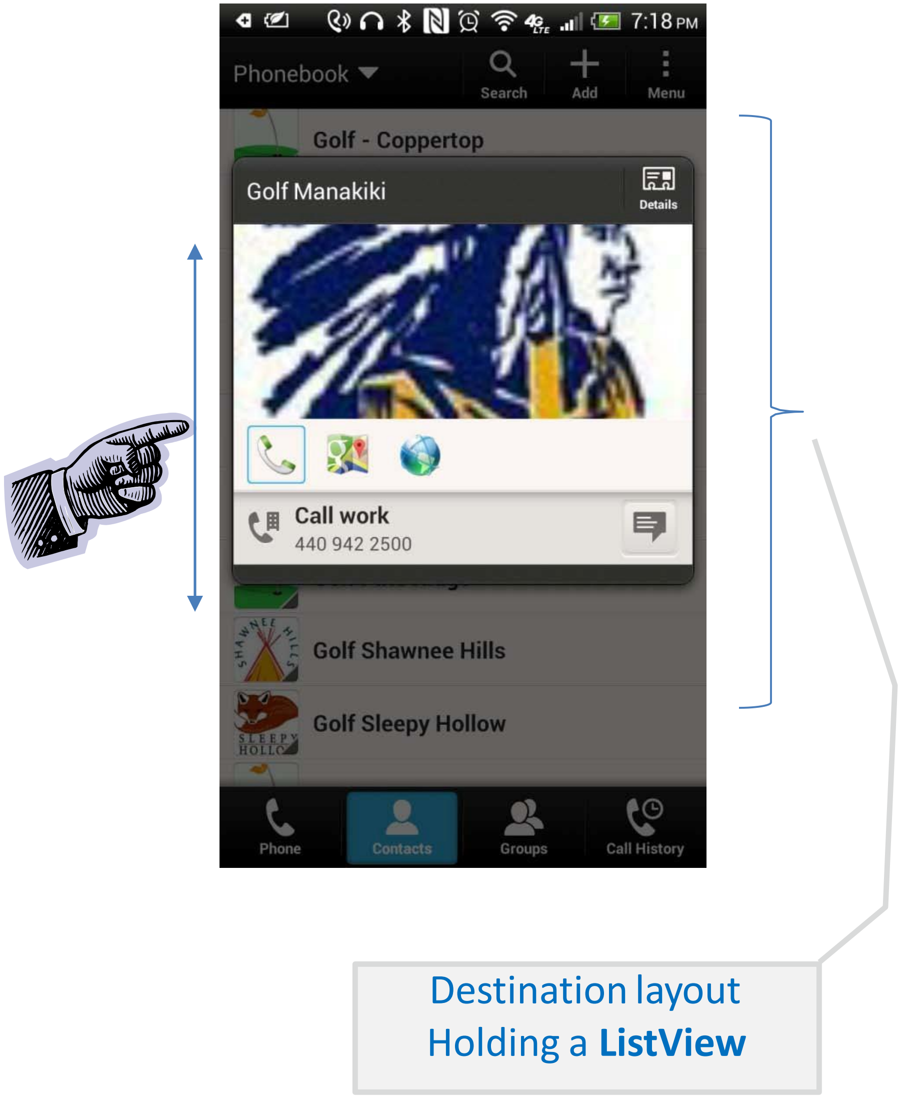
ListViews

The Android **ListView** widget is the most common element used to display data supplied by a **data adapter**.

ListViews are scrollable, each item from the base data set can be shown in an individual row.

Users can tap on a row to make a selection.

A row could display one or more lines of text as well as images.



Destination layout
Holding a **ListView**

List-Based App = ListView + Data + DataAdapter

ArrayAdapter (A Data Beautifier**)**

- An `ArrayAdapter<T>` accepts for input an **array** (or **ArrayList**) of objects of some arbitrary type T.
- The adapter works on each object by (a) applying its `toString()` method, and (b) moving its formatted output string to a **TextView**.
- The formatting operation is guided by a user supplied XML layout specification which defines the appearance of the receiving TextView.

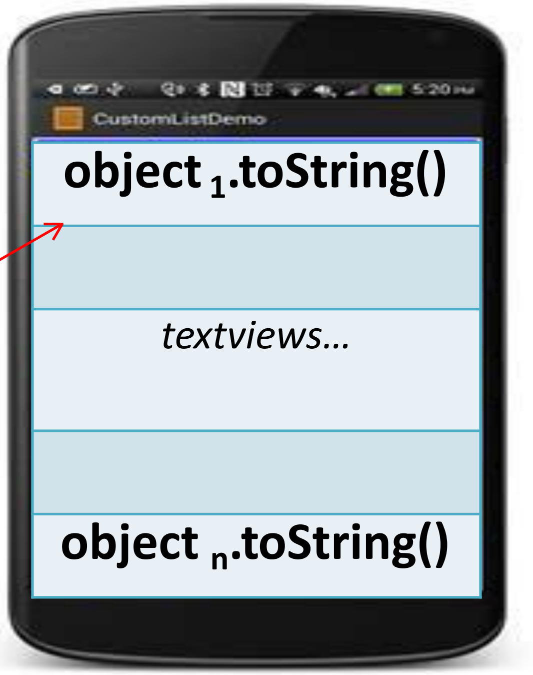
List-Based App = ListView + Data + DataAdapter

Output: 'Pretty' GUI

Input Data - array or java.util.List

{ object$_1$, object$_2$, …, object$_n$ }

Array Adapter



Input XML Specification

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

List-Based App = ListView + Data + DataAdapter

Using the ArrayAdapter<String> Class

```
String[] items = { "Data-0", "Data-1", "Data-2", "Data-3",
                   "Data-4", "Data-5", "Data-6", "Data-7"  };

ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                      this,
                      android.R.layout.simple_list_item_1,
                      items );
```

Parameters:

1. The current activity's **context** `(this)`
2. The **TextView** layout indicating how an individual row should be written (`android.R.Layout.simple_list_item_1`).
3. The actual **data source** (**Array** or **Java.List** containing `items` to be shown).

Using Activity + ArrayAdapter

Example1B: Using Activity & ArrayAdapter

- You may use a common **Activity** class instead of a **ListActivity**.
- Which is purely to implement Lists

  The Layout below uses a ListView identified as @+id/my_list

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:text="Using ListViews..."
        android:textSize="16sp" />

    <ListView                                          try:
        android:id="@+id/my_list"                      wrap_content
        android:layout_width="match_parent"            to see limitations
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```

Using Activity + ArrayAdapter

Example1B: You must 'wired-up' the ListView to a Java activity class, and later bind it to an Adapter.

Example 1B – MainActivity 1 of 2

```
public class ListViewDemo2 extends Activity {
      String[] items = { "Data-0", "Data-1", "Data-2", "Data-3",
                         "Data-4", "Data-5", "Data-6", "Data-7" };
      ListView myListView;
      TextView txtMsg;

      @Override
      public void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         myListView = (ListView) findViewById(R.id.my_list);

         ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
                                     android.R.layout.simple_list_item_1
                                     // R.layout.my_text,   //try this later...
                                     items);
         myListView.setAdapter(aa);

         txtMsg = (TextView) findViewById(R.id.txtMsg);
    }//onCreate
}
```

Using Activity + ArrayAdapter

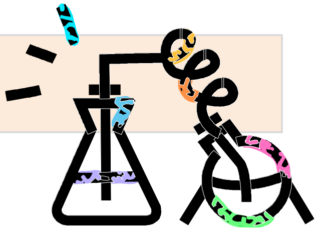
Example 1B – MainActivity 2 of 2

```
myListView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> av, View v,
                                         int position, long id) {

           String text = "Position: " + position
                      + "\nData: " + items[position];

           txtMsg.setText(text);
        }
        });
```

To provide a listener to the ListView control add the fragment above to the **onCreate** method.

Using Activity + ArrayAdapter

Example1C: Custom ListView

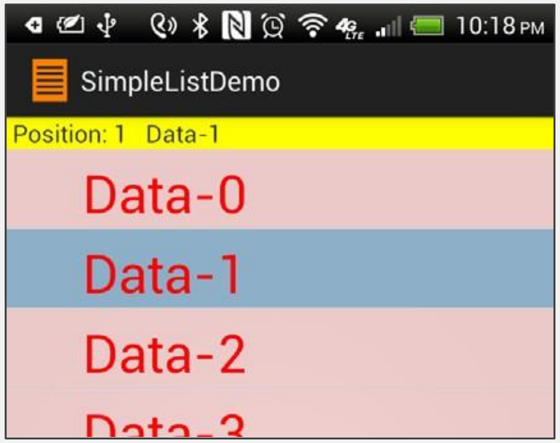


You may want to modify the ListView control to use your **own** GUI design. For instance, you may replace

`android.R.layout.simple_list_item_1` with
`R.layout.my_custom_text`.

Where *my_custom_text* is the Layout specification listed below (held in the **res/layout** folder). It defines how each row is to be shown.

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="2dp"
    android:paddingTop="5dp"
    android:padding="5dp"
    android:textColor="#ffff0000"
    android:background="#22ff0000"
    android:textSize="35sp" />
```



10:18 PM

SimpleListDemo

Position: 1 Data-1

Data-0

Data-1

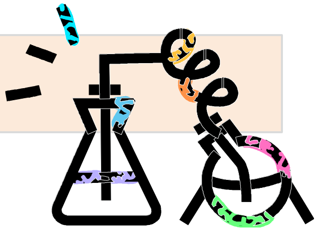
Data-2

Data-3

Note: As of SDK4.0 a TextView could also include an image (For example `.setDrawableLeft(some_image)`)

Using Activity + ArrayAdapter

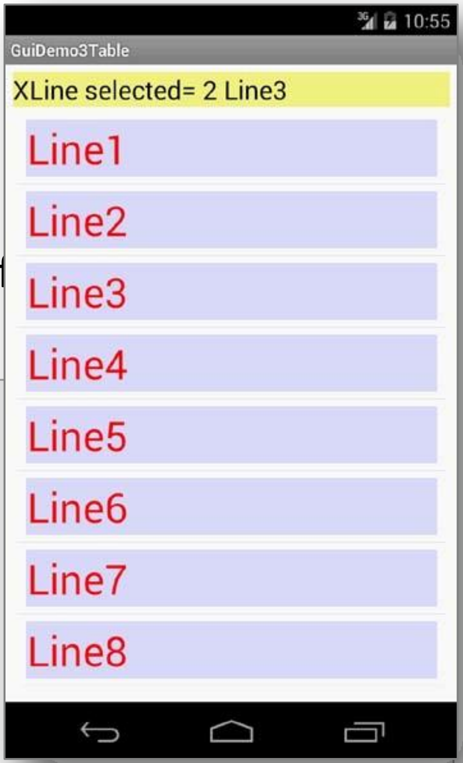
Example1C: Custom ListView



You may also create the ArrayAdapter with more parameters. For instance, the following statement:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                                  getApplication(),
                                  R.layout.my_custom_line3,
                                  R.id.my_custom_textview3,
                                  data );
```

Defines a custom *list* and *textview* layout to show the contents of the `data` array.



XLine selected= 2 Line3

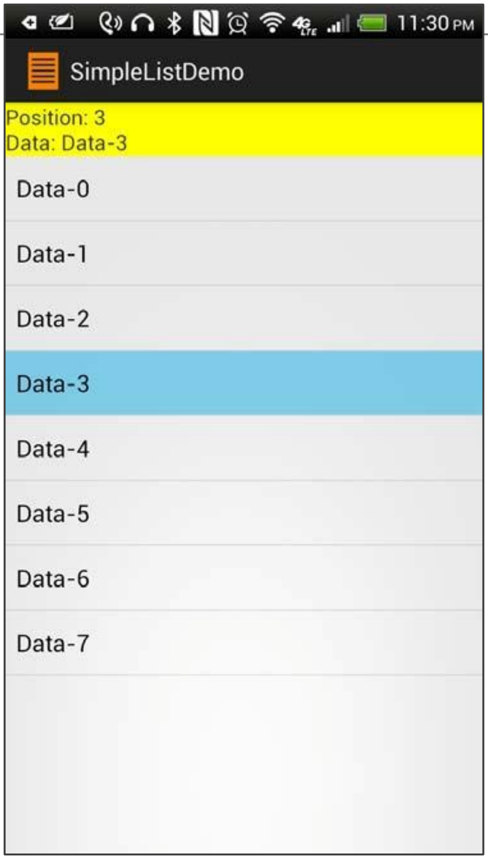
Line1
Line2
Line3
Line4
Line5
Line6
Line7
Line8

```
<!-- my_custom_line3 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >
    <TextView
        android:id="@+id/my_custom_textview3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#220000ff"
        android:padding="1dp"
        android:textColor="#ffff0000"
        android:textSize="35sp" />
</LinearLayout>
```

Using ListActivity + ArrayAdapter

Example1A: ListView showing a simple list (plain text)

Assume a large collection of input data items is held in a **String[]** array.
Each row of the ListView must show a line of text taken from the array.
In our example, when the user makes a selection, you must display on a TextView the selected item and its position in the list.



```
Position: 3
Data: Data-3
```

Data-0
Data-1
Data-2
Data-3
Data-4
Data-5
Data-6
Data-7

Using ListActivity + ArrayAdapter

Example1A: Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:text="Using ListViews..."
        android:textSize="16sp" />

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

    <TextView
        android:id="@android:id/empty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0000"
        android:text="empty list"  />

</LinearLayout>
```

Pay attention to the use of predefined Android components:
`@android:id/list`
`@android:id/empty`

See Appendix A for a description of
`@android:id/list`

Android's built-in list layout

Used for empty lists

Using ListActivity + ArrayAdapter

Example1A: MainActivity (using a ListActivity !)

```
package csu.matos;

import ...

public class ListViewDemo extends ListActivity {

   TextView txtMsg;

   String[] items = {  "Data-0", "Data-1", "Data-2", "Data-3",
                       "Data-4", "Data-5", "Data-6", "Data-7"  };

    // next time try an empty list such as:
    // String[] items = {};
```

CAUTION:
A **ListActivity** is not a "plain" Activity. It is bound to a built-in ListView called @android:id/list

Data Source

```
...
    <ListView android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"   >
    </ListView>
...
```

Fragment already defined in Layout: activity_main.xml

Using ListActivity + ArrayAdapter

Example1A: MainActivity (using a ListActivity !)

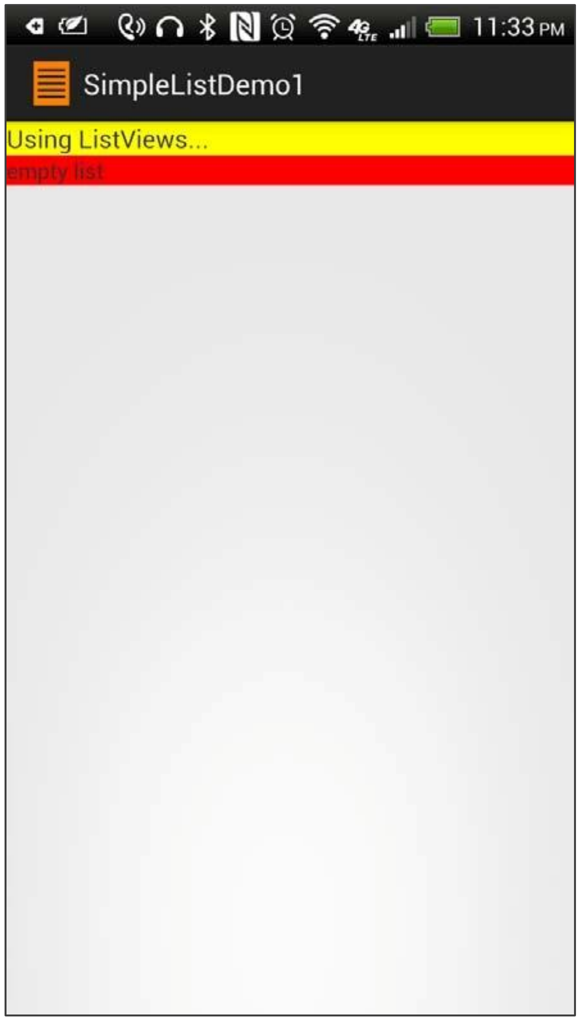
```
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      setListAdapter(new ArrayAdapter<String>(this,
                                        android.R.layout.simple_list_item_1,
                                        items));

      //getListView().setBackgroundColor(Color.GRAY);  //try this idea later

      txtMsg = (TextView) findViewById(R.id.txtMsg);
   }


   @Override
   protected void onListItemClick(ListView l, View v, int position, long id) {
      super.onListItemClick(l, v, position, id);
      String text = " Position: " + position + "    " + items[position];
      txtMsg.setText(text);
   }

}
```

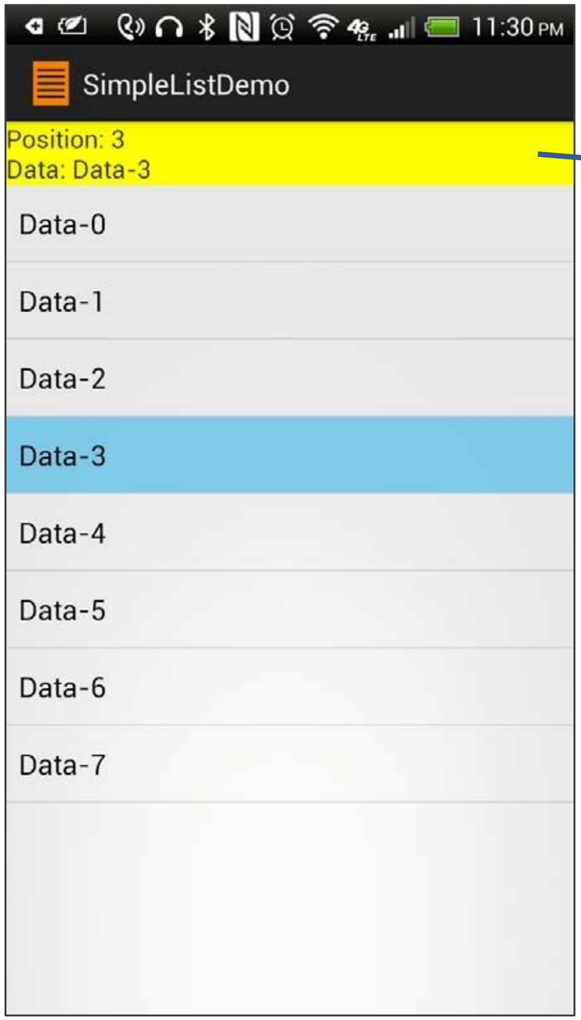
List adapter

List Click Listener

Using ListActivity + ArrayAdapter

Example1A: MainActivity (using a ListActivity !)





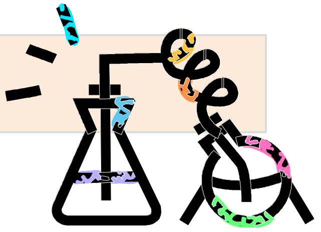
Selection seen by the listener

Background flashes blue to acknowledge the user's selection

ListView displayed on a device running SDK4.3

Using ListActivity + ArrayAdapter

An experiment based on Example1A

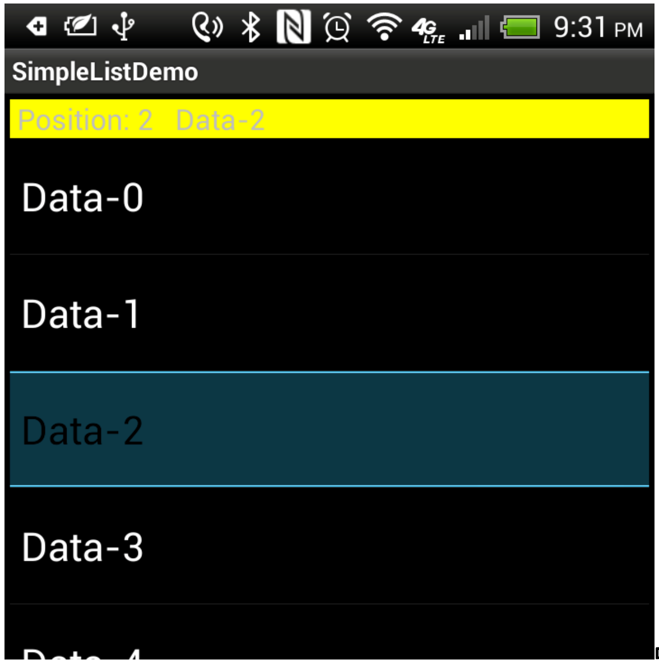


1. Open the AndroidManifest.xml file. Under the <Application> tag look for the clause
   `android:theme="@style/AppTheme"`

2. Change the previous line to the following value
   `android:theme="@android:style/Theme.Black"`

3. Try some of the other styles, such as:

```
Theme.DeviceDefault
Theme.Dialog
Theme.Holo
Theme.Light
Theme.Panel
Theme.Translucent
Theme.Wallpaper
```
etc.



9:31 PM

SimpleListDemo

Position: 2 Data-2

Data-0

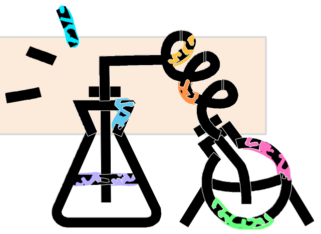
Data-1

Data-2

Data-3

Using ListActivity + ArrayAdapter

Another code experiment based on Example1A



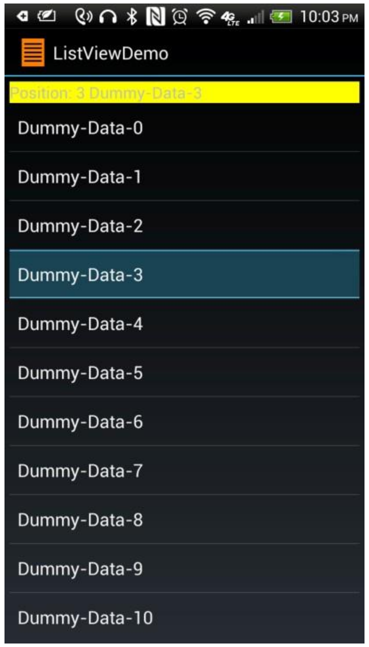
1. Open the AndroidManifest.xml file. Under the <Application> tag look for the clause `android:theme="@style/AppTheme"`

2. Now open the **`res/values/styles`** folder. Look for the entry
   ```
   <style name="AppTheme" parent="android:Theme.Light" />
   ```
   which indicates to use the "Light" theme (white background instead of black).

3. Remove from the manifest the entry *android:theme*.

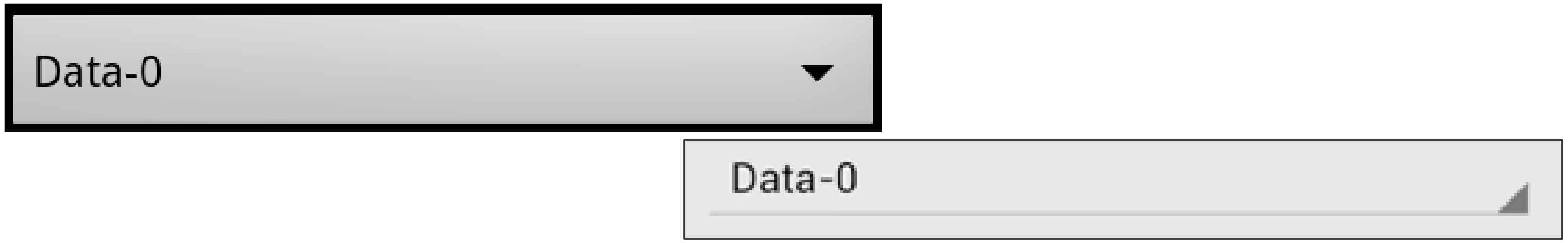
4. Remove from the onCreate method the statement:
   ```
   getListView().setBackgroundColor(Color.GRAY);
   ```

3. Run the application again. Observe its new look.

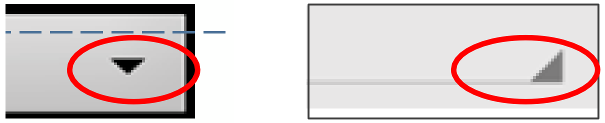


```
10:03 PM
ListViewDemo
Position: 3 Dummy-Data-3
Dummy-Data-0
Dummy-Data-1
Dummy-Data-2
Dummy-Data-3
Dummy-Data-4
Dummy-Data-5
Dummy-Data-6
Dummy-Data-7
Dummy-Data-8
Dummy-Data-9
Dummy-Data-10
```

The Spinner Widget

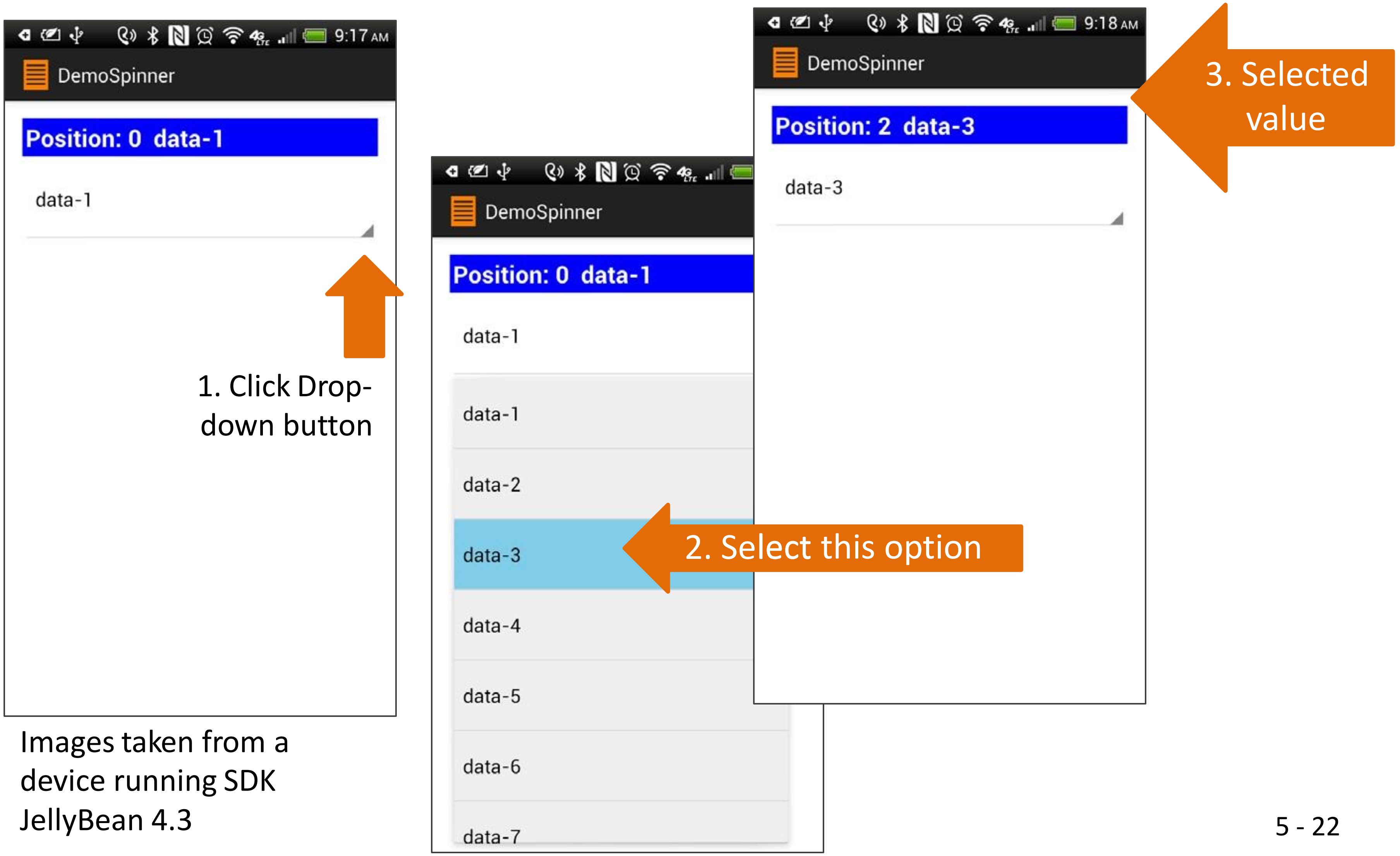


- Android's **Spinner** is equivalent to a *drop-down* selector.
- Spinners have the same functionality of a ListView but take less screen space.
- An Adapter is used to supply its data using ***setAdapter(…)***
- A listener captures selections made from the list with ***setOnItemSelectedListener(…)***.
- The ***setDropDownViewResource(…)*** method shows the drop-down multi-line window



Example2: Using the Spinner Widget

Example 2. A list of options named ‘Data-0’, ‘Data-1’, ‘Data-2’ and so on, should be displayed when the user taps on the ‘down-arrow’ portion of the spinner.



```
9:17 AM
DemoSpinner
Position: 0  data-1
data-1
```

```
DemoSpinner
Position: 0  data-1
data-1
data-1
data-2
data-3
data-4
data-5
data-6
data-7
```

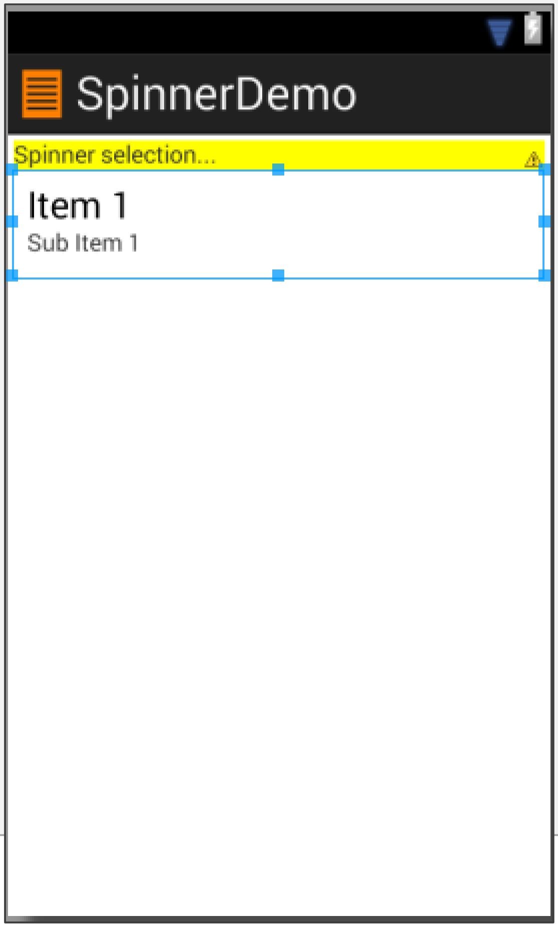
```
9:18 AM
DemoSpinner
Position: 2  data-3
data-3
```

Images taken from a device running SDK JellyBean 4.3

Using the Spinner Widget

Example2: Spinner Demo - Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="3dp"
    tools:context=".MainActivity" >

    <TextView android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffffff00"
        android:text="@string/hello_world" />

    <Spinner android:id="@+id/spinner1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />


</LinearLayout>
```



Using the Spinner Widget

Example2: Spinner Demo - MainActivity 1 of 2

```
public class MainActivity extends Activity

                             implements AdapterView.OnItemSelectedListener{

   // GUI objects
   TextView txtMsg;
   Spinner spinner;

   // options to be offered by the spinner
   String[] items = { "Data-0", "Data-1", "Data-2", "Data-3", "Data-4",
                      "Data-5", "Data-6", "Data-7" };

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtMsg = (TextView) findViewById(R.id.txtMsg);

      spinner = (Spinner) findViewById(R.id.spinner1);


      // use adapter to bind items array to GUI layout
      ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            this,
            android.R.layout.simple_spinner_dropdown_item,
            items);
```

Using the Spinner Widget

Example2: Spinner Demo - MainActivity   2 of 2

```
        // bind everything together
        spinner.setAdapter(adapter);

        // add spinner a listener so user can meake selections by tapping an item
        spinner.setOnItemSelectedListener(this);

    }
    // next two methods implement the spinner's listener
    @Override
    public void onItemSelected(AdapterView<?> parent, View v, int position,
            long id) {
        // echo on the textbox the user's selection
        txtMsg.setText(items[position]);
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO do nothing – needed by the interface

    }

}
```

Assignment 2

1. For ListViews showing **complex** arrangement of visual elements –such as text plus images- you need to provide a **custom made adapter** in which the **`getView(…)`** method explains how to manage the placement of each data fragment in the complex layout.

Customize the ListView with a TextView and another Widget ImageView.

2. Make Spinner as selection tool for ListView.