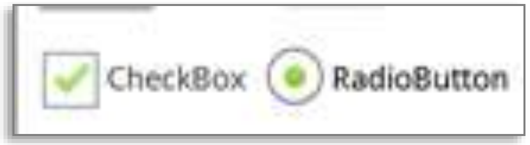
List-Based Widgets:

Lists, Grids, and Scroll Views

ONE LOVE. ONE FUTURE.

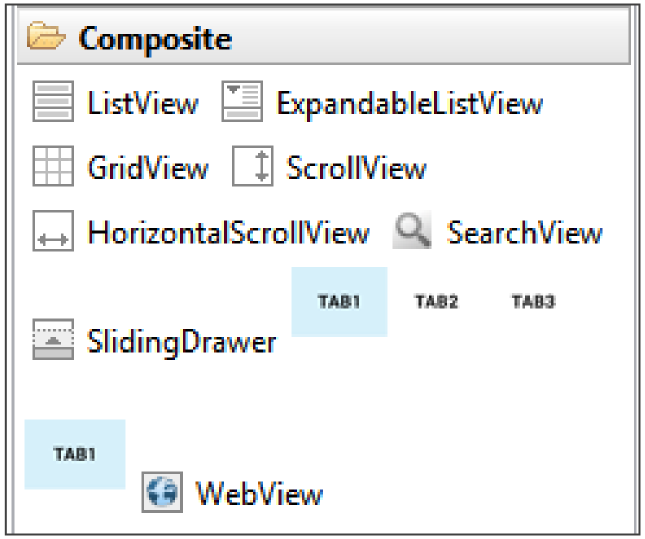
GUI Design for Selection Making

- 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)

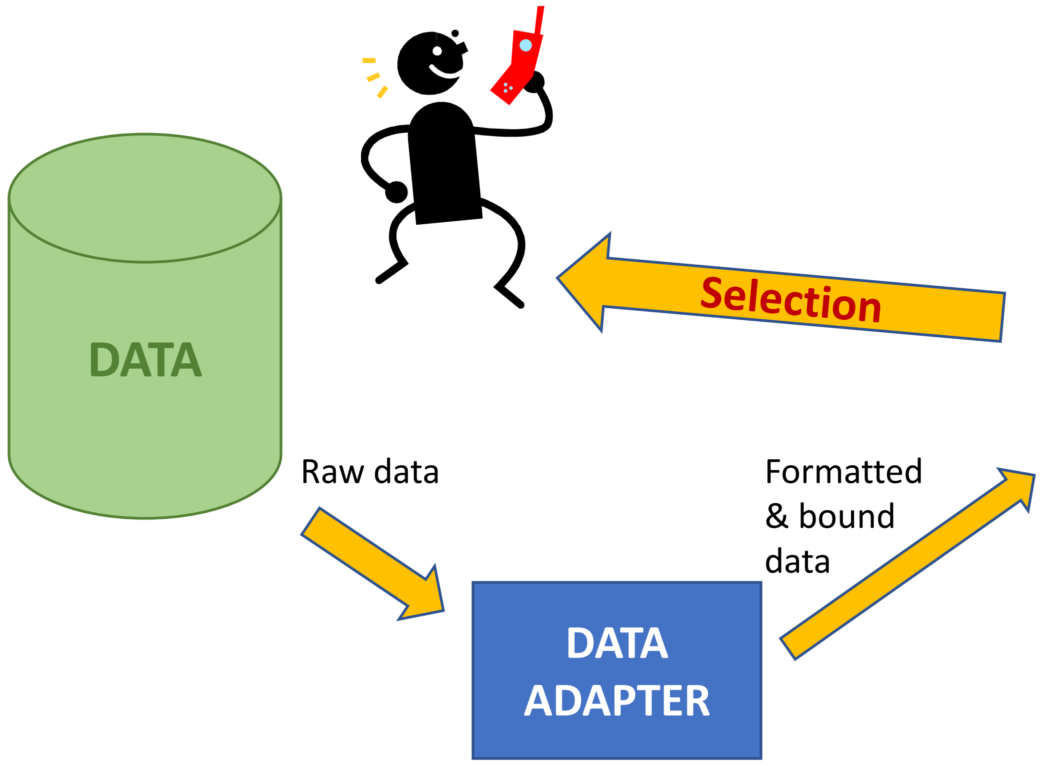


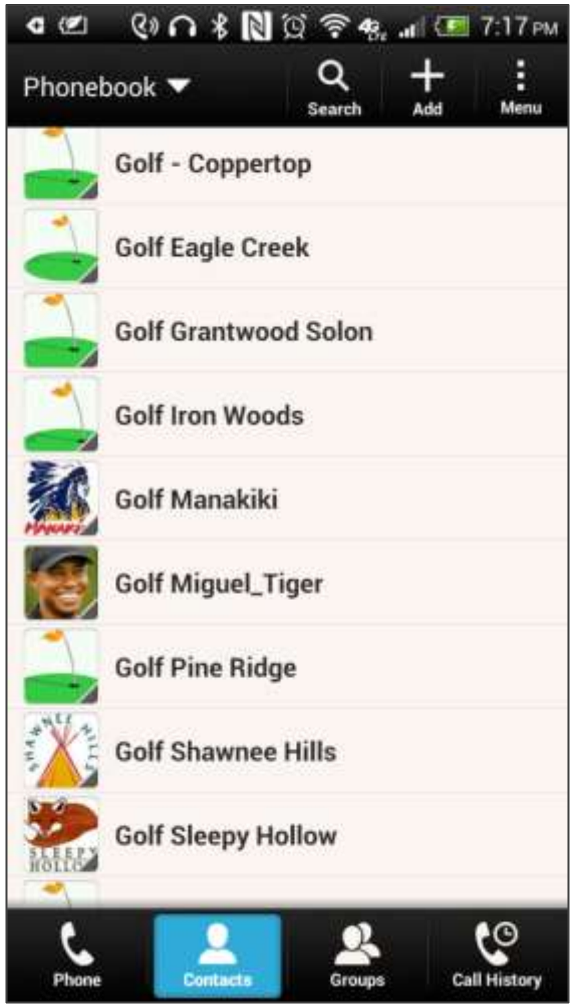
- When the set of values to choose from is large, other Android **List-Based Widgets** are more appropriate.

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



Showing a large set of choices on the GUI





Destination layout
Holding a **ListView**

- 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.

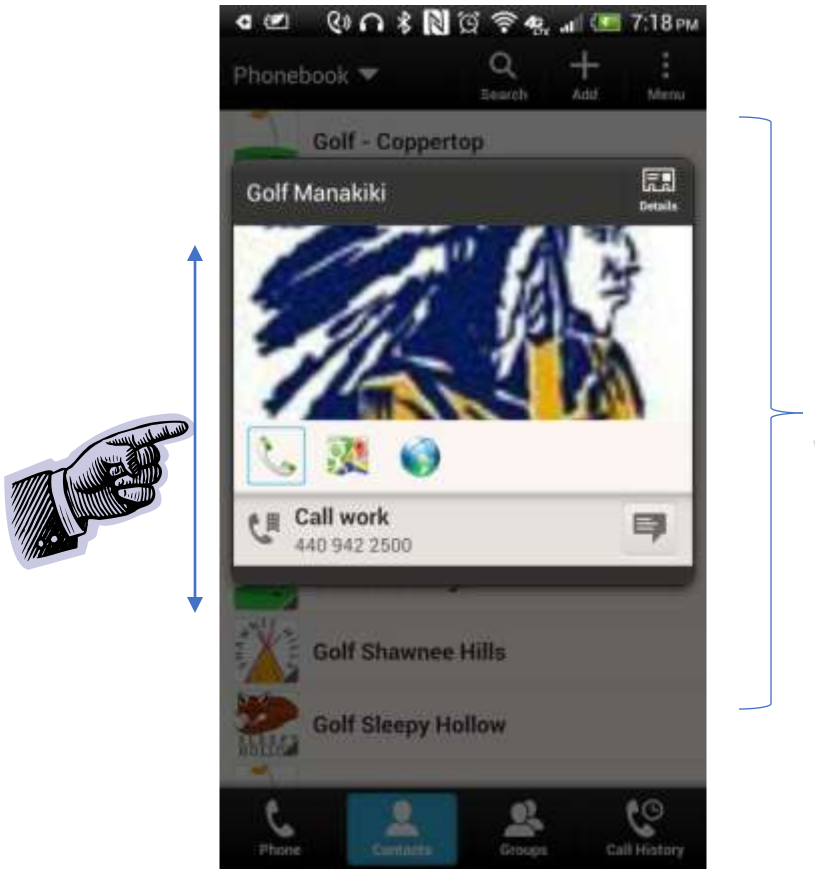
ListViews

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**

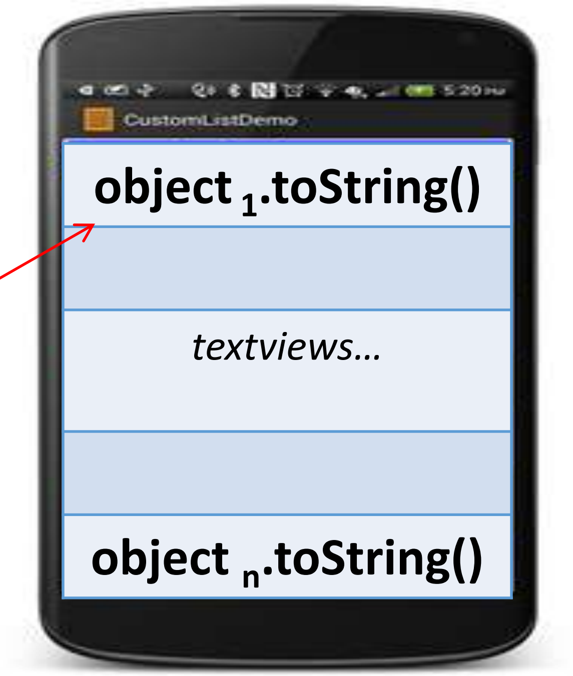
ArrayAdapter (A Data Beautifier)

- 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.
- 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.

ArrayAdapter (A Data Beautifier)

Input Data - array or 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"
    ...
/>
```

Using the ArrayAdapter<String> Class

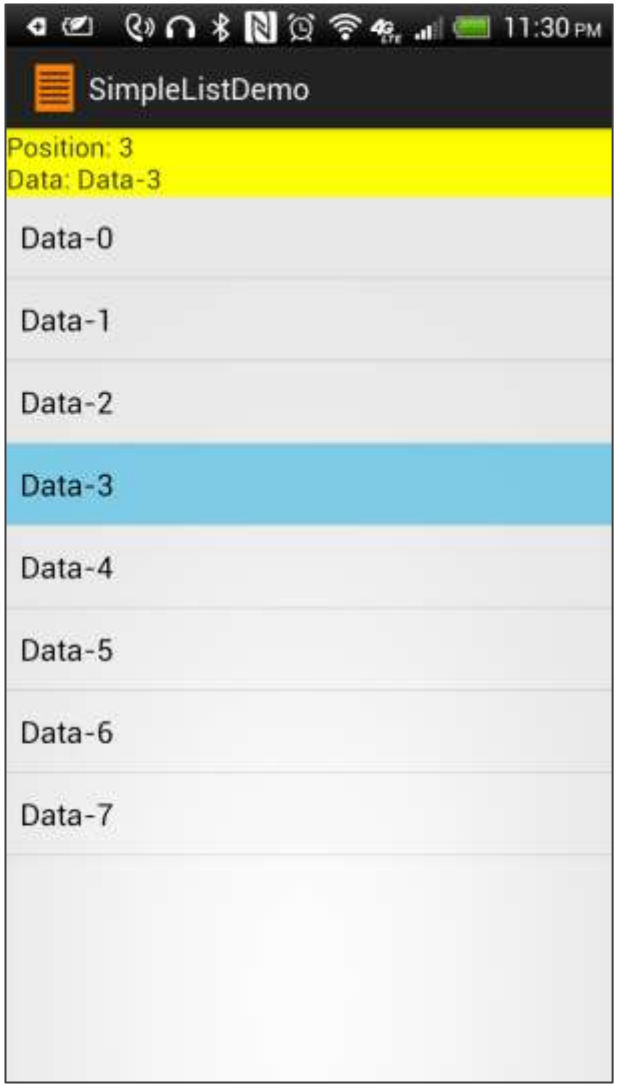
```
val items: Array<String> = arrayOf(
    "Data-0", "Data-1", "Data-2", "Data-3",
    "Data-4", "Data-5", "Data-6", "Data-7")
val adapter: ArrayAdapter<String> = ArrayAdapter(
    this,
    android.R.layout.simple_list_item_1,
    items)
```

Parameters:

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

Example 1: 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

Example 1: Layout

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

</LinearLayout>
```

Example 1: MainActivity

```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val txtMsg: TextView = findViewById(R.id.txtMsg)

        val items: Array<String> = arrayOf("Data-0", "Data-1", "Data-2", "Data-3",
            "Data-4", "Data-5", "Data-6", "Data-7")
        val adapter: ArrayAdapter<String> = ArrayAdapter(
            this,
            android.R.layout.simple_list_item_1,
            items)
        val listView: ListView = findViewById(R.id.my_list)
        listView.adapter = adapter

        listView.setOnItemClickListener { adapterView, view, i, l -> txtMsg.text =
"Position: $i\nData: ${items[i]}"}
    }
}
```

When dataset is changed

Using **notifyDatasetChanged()** method of adapter to refresh the list, when:

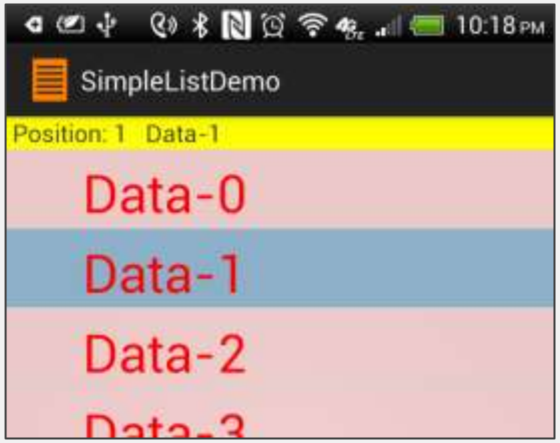
- Data is updated
- Items are removed
- Items are added

Example1: 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 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" />
```

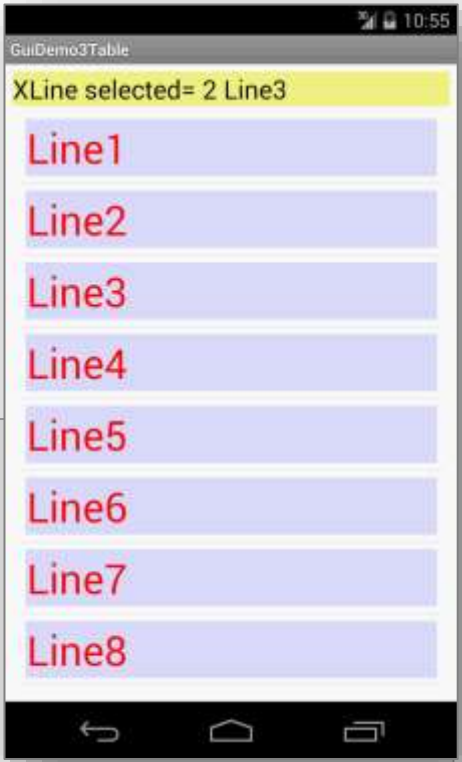


Example1: Custom ListView

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

```
val adapter: ArrayAdapter<String> = ArrayAdapter(
    this,
    R.layout.item_view,
    R.id.text_item,
    items)
```

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


GuiDemo3Table

XLine selected= 2 Line3

Line1

Line2

Line3

Line4

Line5

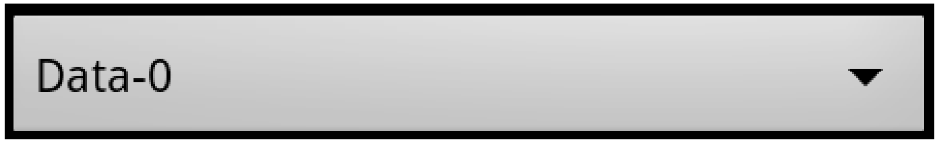
Line6

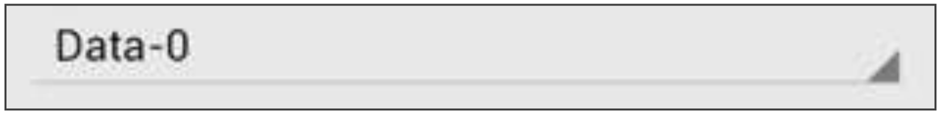
Line7

Line8

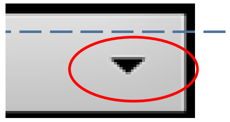
```
<!- item_view -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >
    <TextView
        android:id="@+id/text_item"
        android:layout_width="match_parent"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

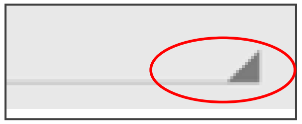
The Spinner Widget





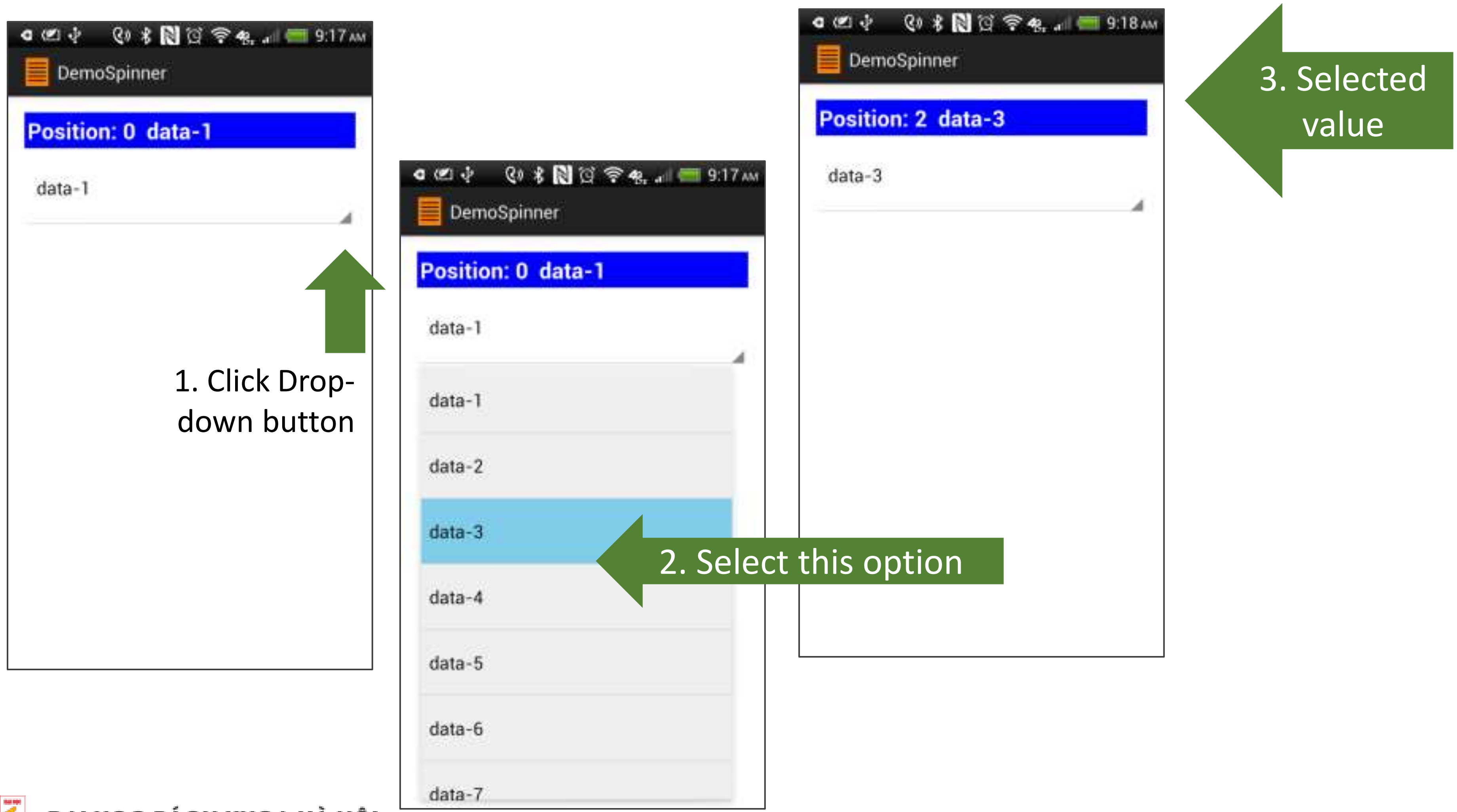
- Android's **Spinner** is equivalent to a *drop-down* selector.
- Spinners have the same functionality of a ListView but take less screen space.
- An Adapter is used to supply its data using ***adapter***
- A listener captures selections made from the list with ***setOnItemSelectedListener***.





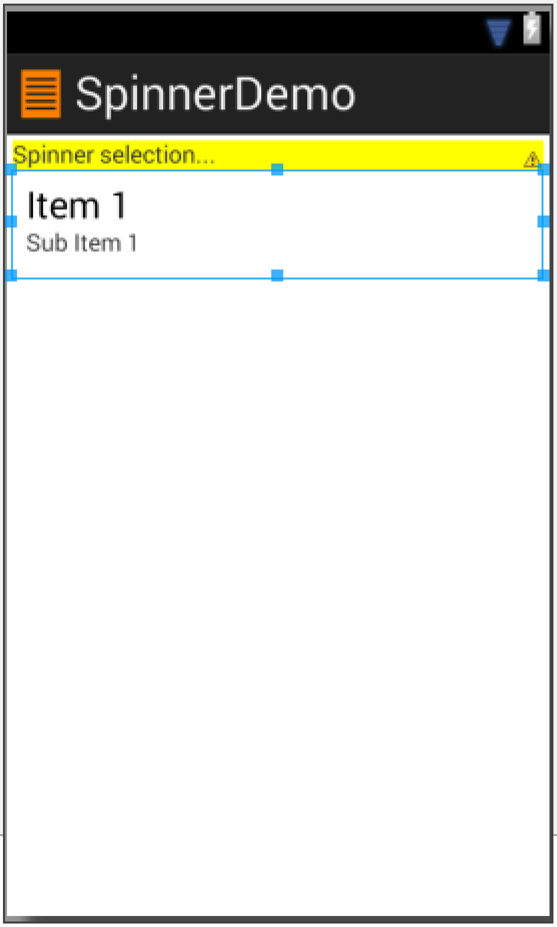
Example2: Using the Spinner Widget

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.



Example 2: 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="Spinner selection" />

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

</LinearLayout>
```



Example 2: MainActivity

```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val txtMsg: TextView = findViewById(R.id.txtMsg)

        val items: Array<String> = arrayOf("Data-0", "Data-1", "Data-2", "Data-3",
            "Data-4", "Data-5", "Data-6", "Data-7")
        val arrayAdapter: ArrayAdapter<String> = ArrayAdapter(
            this,
            android.R.layout.simple_dropdown_item_1line,
            items)

        findViewById<Spinner>(R.id.spinner1).run {
            adapter = arrayAdapter
            onItemSelectedListener = object : OnItemSelectedListener {
                override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
                    txtMsg.text = items[p2]
                }
                override fun onNothingSelected(p0: AdapterView<*>?) {
                    TODO("Not yet implemented")
                }
            }
        }
    }
}
```

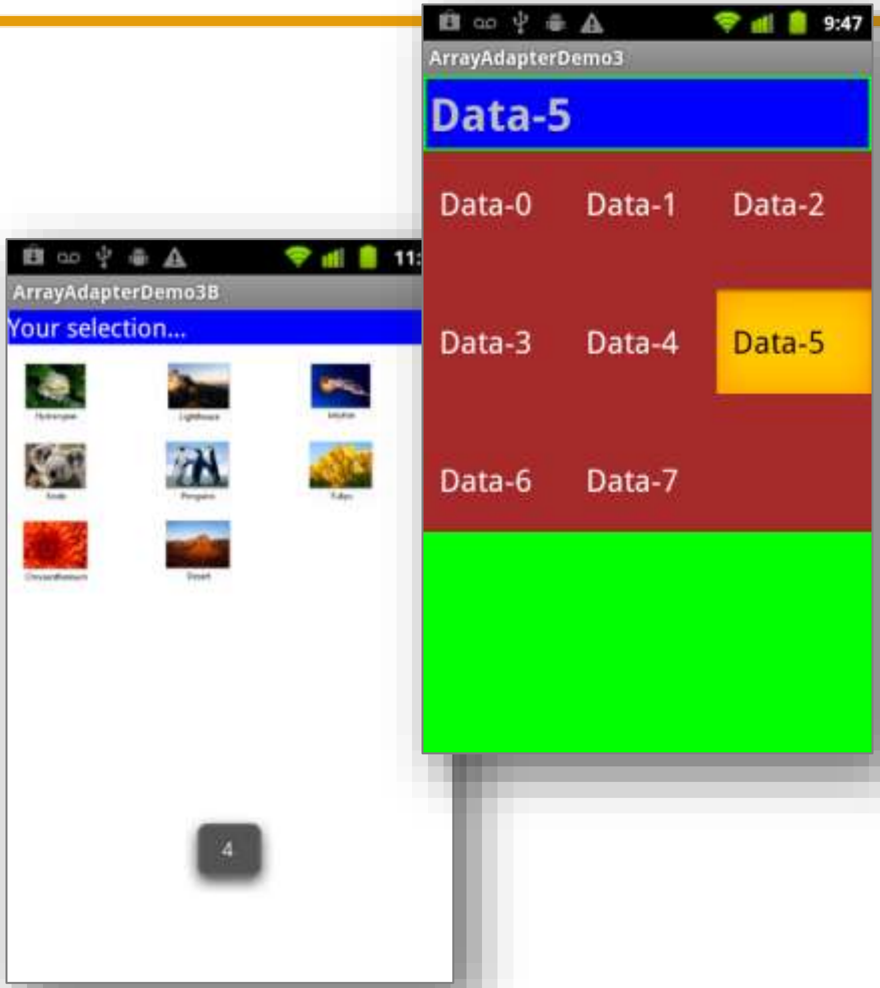
The GridView widget

GridView

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Data items shown by the grid are supplied by a data adapter.

Grid cells can show text and/or images



GridView: Useful Properties

Some properties used to determine the number of columns and their sizes:

- **android:numColumns**
  indicates how many columns to show. When used with option "auto_fit", Android determines the number of columns based on available space and the properties listed below.

- **android:verticalSpacing** and **android:horizontalSpacing**
  indicate how much free space should be set between items in the grid.

- **android:columnWidth**
  column width in **dip**s.

- **android:stretchMode**
  indicates how to modify image size when there is available space not taken up by columns or spacing .

GridView: Fitting the View to the Screen

Suppose the screen is **320** (dip) pixels wide, and we have

> **android:columnWidth** set to **100dip** and
> **android:horizontalSpacing** set to **5dip**.

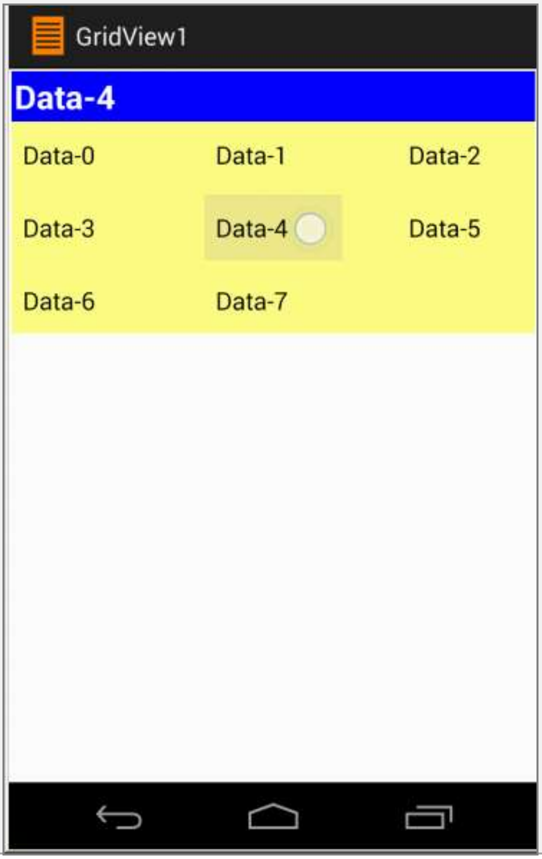
The user would see three columns taking **310** pixels (three columns of 100 pixels and two separators of 5 pixels).

With **android:stretchMode** set to *columnWidth*, the three columns will each expand by 3-4 pixels to use up the remaining 10 pixels.

With **android:stretchMode** set to *spacingWidth*, the two internal whitespaces will each grow by 5 pixels to consume the remaining 10 pixels.

Example 3: GridView Demo - Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="2dp"
    tools:context=".MainActivity" >
    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:textSize="24sp"
        android:textStyle="bold"
        android:textColor="#ffffffff"
        android:padding="2dip"    />

     <GridView
        android:id="@+id/grid"
        android:background="#77ffff00"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:verticalSpacing="5dip"
        android:horizontalSpacing="5dip"
        android:numColumns="auto_fit"
        android:columnWidth="100dip"
        android:stretchMode="spacingWidth"   />
</LinearLayout>
```



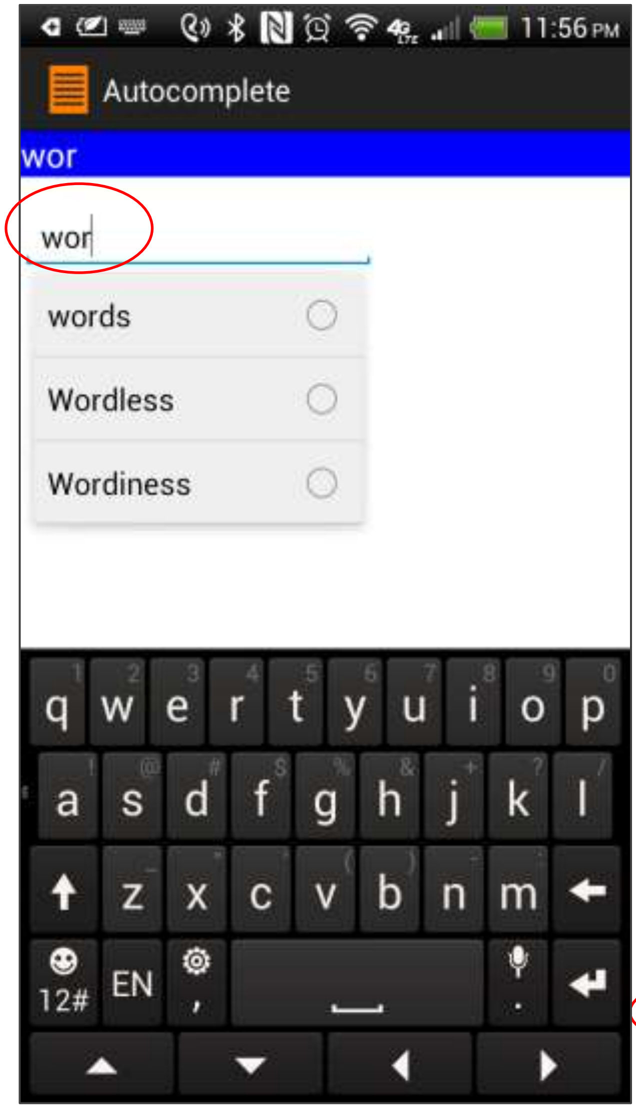
Example 3: GridView Demo - MainActivity

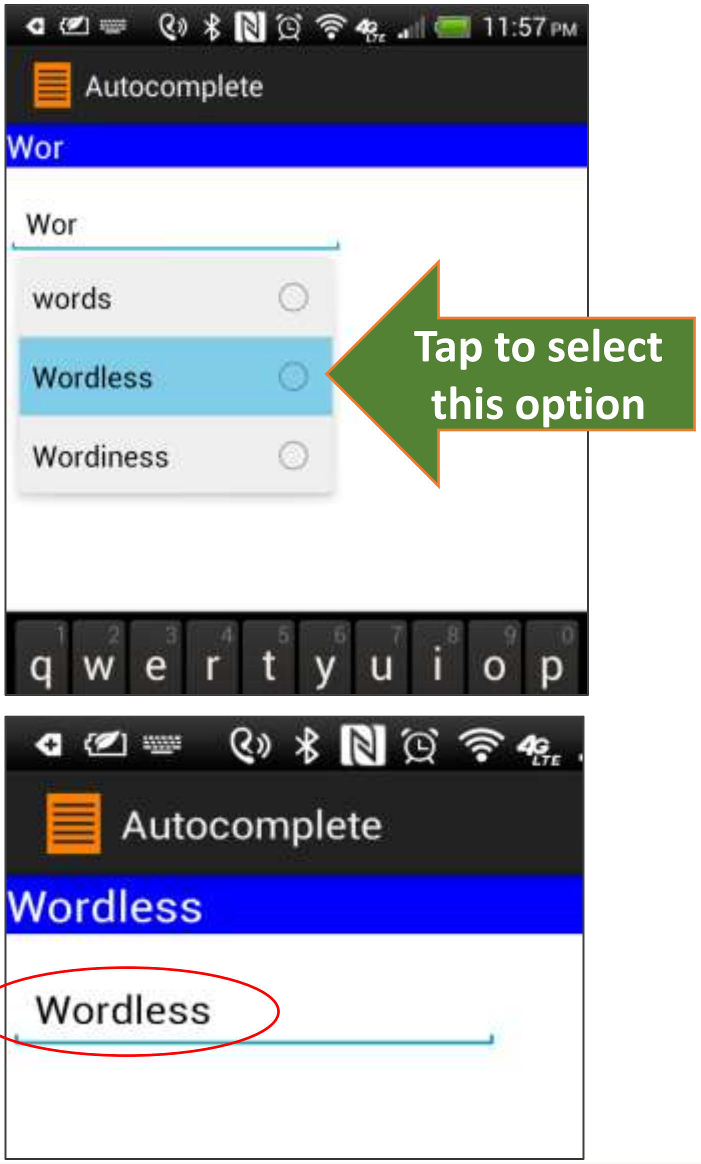
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val txtMsg: TextView = findViewById(R.id.txtMsg)

        val items: Array<String> = arrayOf("Data-0", "Data-1", "Data-2", "Data-3",
            "Data-4", "Data-5", "Data-6", "Data-7")
        val arrayAdapter: ArrayAdapter<String> = ArrayAdapter(
            this,
            android.R.layout.simple_list_item_1,
            items)

        findViewById<GridView>(R.id.grid).run {
            adapter = arrayAdapter
            onItemClickListener = object : OnItemClickListener {
                override fun onItemClick(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
                    txtMsg.text = items[p2]
                }
            }
        }
    }
}
```

The AutoCompleteTextView Widget

- An **AutoComplete** box is a more specialized version of the **EditText** view.
- Characters typed so far are compared with the beginning of words held in a user-supplied list of *suggested* values.
- Suggestions matching the typed prefix are shown in a *selection list.*
- The user can choose from the suggestion list or complete typing the word.
- The **`android:completionThreshold`** property is used to trigger the displaying of the suggestion list. It indicates the number of characters to watch for in order to match prefixes.

The AutoCompleteTextView Widget





Example 4.
A list of selected words beginning with "**wor**" or "**set**" is being watched.

If any of these prefixes (3 letters) are entered the TextWatcher mechanism shows an option list.

Example 4: AutoComplete Demo – Layout

```
<RelativeLayout 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" >

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:textColor="#ffffffff"
        android:background="#ff0000ff" >
      </TextView>

    <AutoCompleteTextView
        android:id="@+id/autoCompleteTextView1"
        android:hint="type here..."
        android:completionThreshold="3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtMsg"
        android:layout_marginTop="15dp"
        android:ems="10" />

</RelativeLayout>
```

Wait 3 chars to work

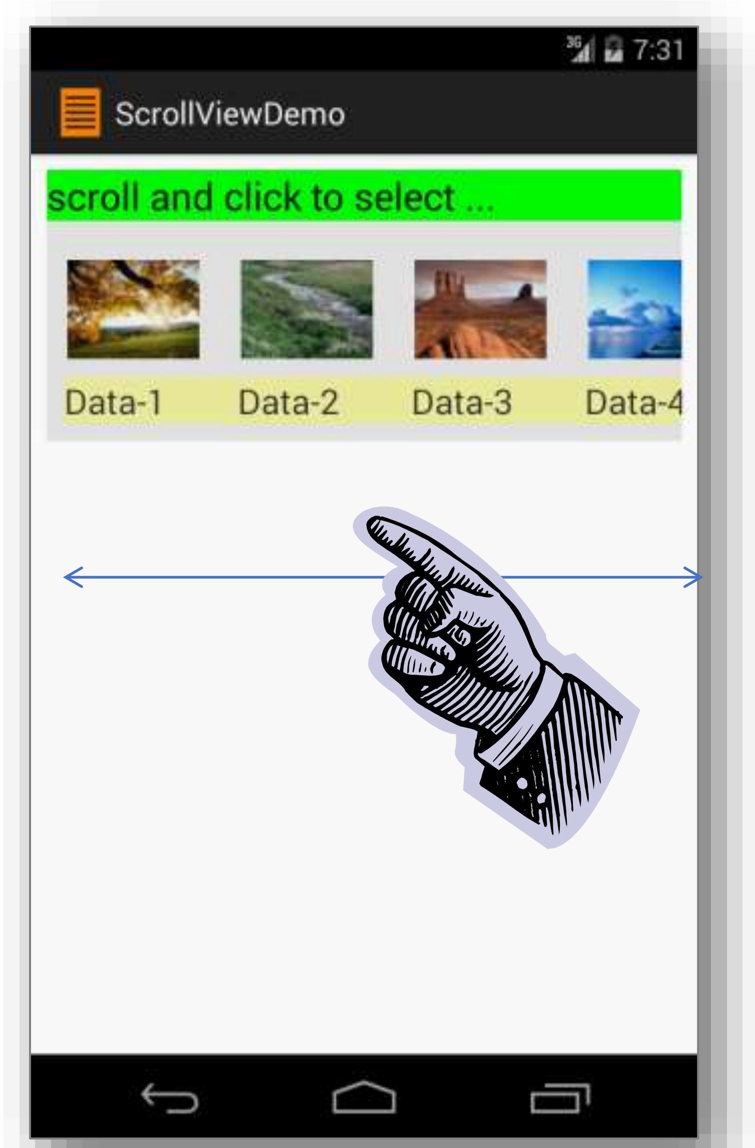
Example 4: AutoComplete Demo – Main Activity

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val txtMsg: TextView = findViewById(R.id.txtMsg)
        val items: Array<String> = arrayOf("words", "starting", "with", "set", "Setback", "Setline",
"Setoffs", "Setouts", "Setters", "Setting", "Settled", "Settler", "Wordless", "Wordiness", "Adios")
        val arrayAdapter: ArrayAdapter<String> = ArrayAdapter(this,
            android.R.layout.simple_list_item_1, items)
        findViewById<AutoCompleteTextView>(R.id.edit_keyword).run {
            setAdapter(arrayAdapter)
            addTextChangedListener(object : TextWatcher {
                override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
                    TODO("Not yet implemented")
                }

                override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
                    txtMsg.text = text
                }

                override fun afterTextChanged(p0: Editable?) {
                    TODO("Not yet implemented")
                }
            })
        }
    }
}
```

Example 4: AutoComplete Demo – Main Activity

```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val txtMsg: TextView = findViewById(R.id.txtMsg)

        val items: Array<String> = arrayOf("words", "starting", "with", "set",
            "Setback", "Setline", "Setoffs", "Setouts", "Setters", "Setting",
            "Settled", "Settler", "Wordless", "Wordiness", "Adios")
        val arrayAdapter: ArrayAdapter<String> = ArrayAdapter(
            this,
            android.R.layout.simple_list_item_1,
            items)

        findViewById<AutoCompleteTextView>(R.id.edit_keyword).run {
            setAdapter(arrayAdapter)
            addTextChangedListener(onTextChanged = {text, start, count, after ->
                txtMsg.text = text})
        }
    }
}
```

The HorizontalScrollView Widget



HorizontalScrollViews allow the user to graphically select an option from a set of small images called *thumbnails* +.

The user interacts with the viewer using two simple actions:

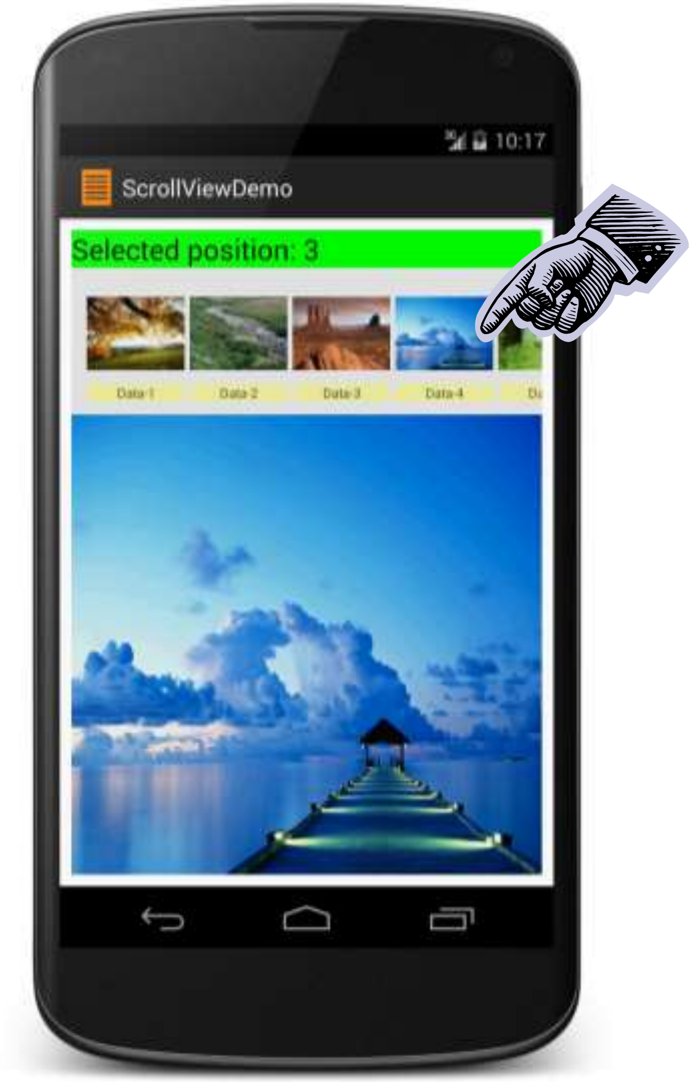
1. Scroll the list (left ⟷ right)
2. Click on a thumbnail to pick the option it offers.

In our example, when the user clicks on a thumbnail the app responds by displaying a high-resolution version of the image

+. A typical thumbnail size is 100x100 pixels (or less).

Example 5: HorizontalScrollView Demo

- In this example we place a **HorizontalScrollView** at the top of the screen, this view will show a set of thumbnail options.
- The user may scroll through the images and finally tap on a particular selection.
- A better quality version of the selected picture will be displayed in an **ImageView** widget placed below the horizontal scroller.



Example 5: Populating The HorizontalScrollView Widget

1. Our **HorizontalScrollView** will expose a list of frames, each containing an *icon* and a *caption* below the icon.

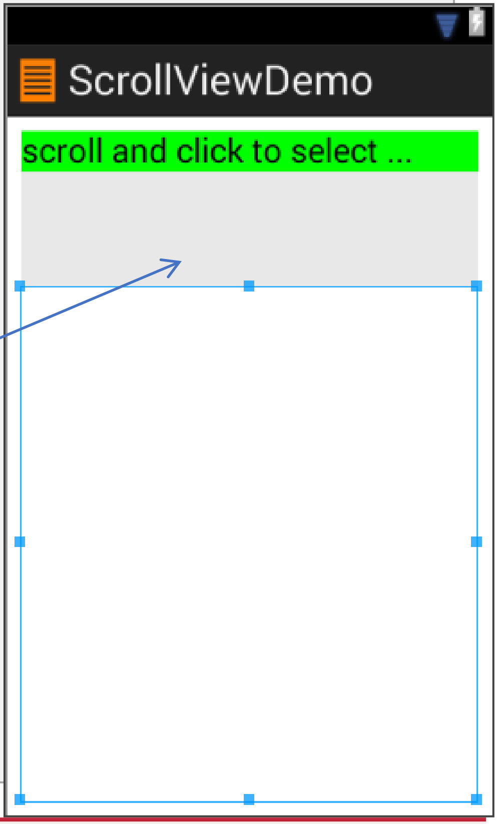
2. The *frame_icon_caption.xml* layout describes the formatting of icon and its caption. This layout will be **inflated** in order to create run-time GUI objects.

3. After the current *frame* is filled with data, it will be added to the growing set of views hosted by the *scrollViewgroup* container (scrollViewgroup is nested inside the horizontal scroller).

4. Each *frame* will receive an **ID** (its current position in the scrollViewgroup) as well as an individual **onClick** listener.

Example 5: HorizontalScrollView Demo – Layout 1 of 2

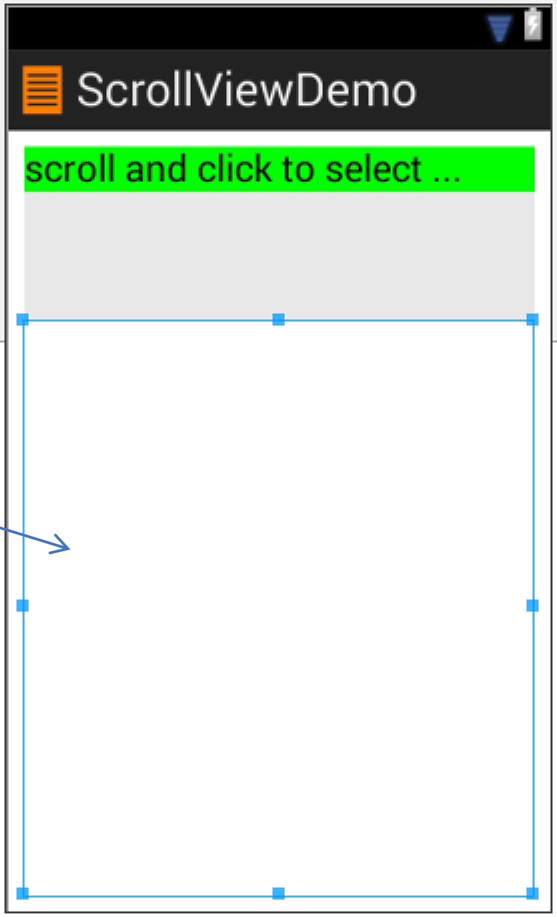
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffffffff"
    android:orientation="vertical"
    android:padding="2dp" >

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff00ff00"
        android:text="scroll and click to select ..."
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <HorizontalScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#44aaaaaa" >

        <LinearLayout
            android:id="@+id/viewgroup"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:padding="10dip" >
        </LinearLayout>

    </HorizontalScrollView>
```



scroll and click to select ...

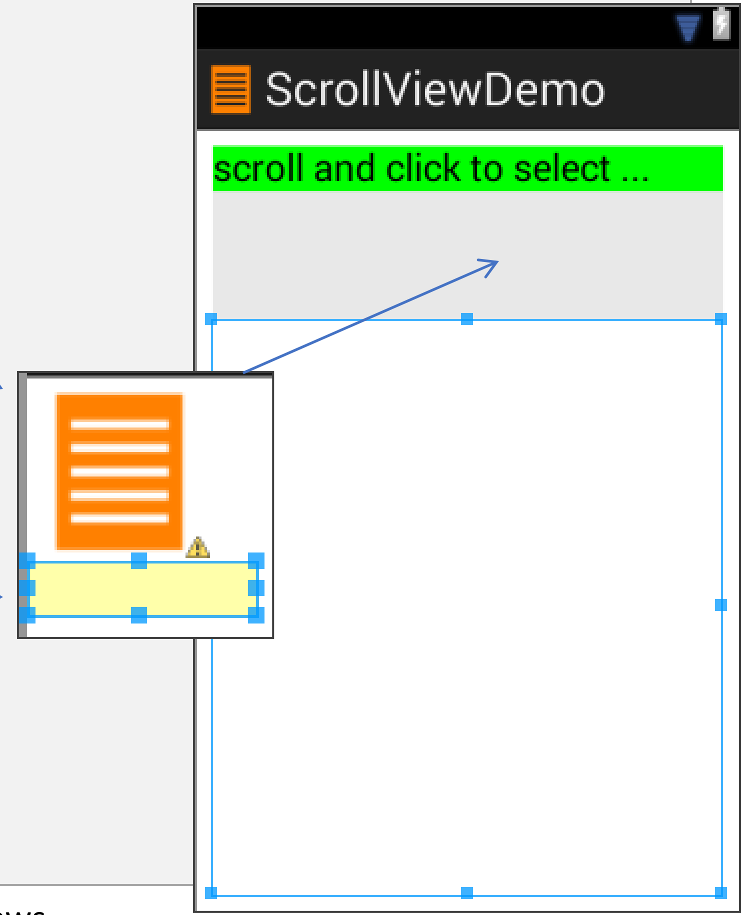
Example 5: HorizontalScrollView Demo – Layout 2 of 2

```
    <ImageView
        android:id="@+id/imageSelected"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2" />

</LinearLayout>
```



Example 5: Layout: *frame_icon_caption.xml*

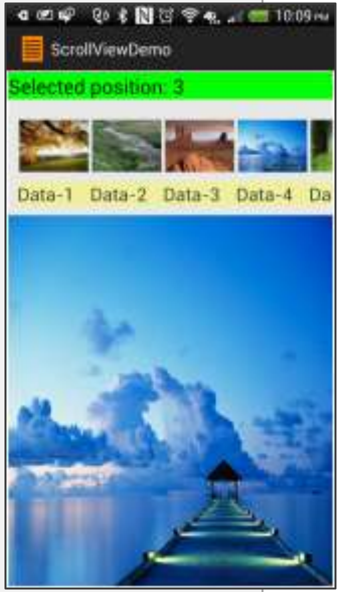
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:paddingLeft="2dp"
        android:paddingRight="2dp"
        android:paddingTop="2dp"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/caption"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#55ffff00"
        android:textSize="20sp" />

</LinearLayout>
```



This layout will be used by an **inflater** to dynamically create new views.
These views will be added to the linear layout contained inside the HorizontalScrollerView.

Example 5: HorizontalScrollView Demo – MainActivity 1 of 2

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val txtMsg: TextView = findViewById(R.id.txtMsg)
        val viewGroup: LinearLayout = findViewById(R.id.viewgroup)
        val imageSelected: ImageView = findViewById(R.id.imageSelected)

        // Prepare arrays of data
        val captionList: ArrayList<String> = ArrayList()
        val thumbList: ArrayList<Int> = ArrayList()
        val imageList: ArrayList<Int> = ArrayList()

        for (i in 1..27) {
            captionList.add("Data-$i")
            thumbList.add(resources.getIdentifier("thumb$i", "drawable", packageName))
            imageList.add(resources.getIdentifier("wall$i", "drawable", packageName))
        }
```



Example 5: HorizontalScrollView Demo – MainActivity 2 of 2

```
        // Populate the scroll view
        for (i in 0..<captionList.size) {
            val view: View = layoutInflater.inflate(R.layout.item_view, viewGroup, false)
            view.id = i

            view.findViewById<TextView>(R.id.caption).text = captionList[i]
            view.findViewById<ImageView>(R.id.icon).setImageResource(thumbList[i])

            viewGroup.addView(view)

            // Process click event
            view.setOnClickListener {
                txtMsg.text = "Selected position: ${view.id}"
                imageSelected.setImageResource(imageList[view.id])
            }
        }
    }
}
```

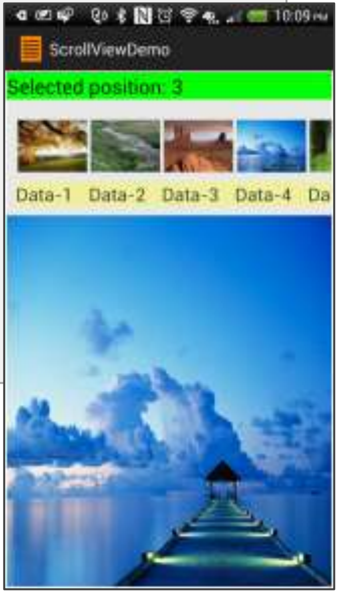
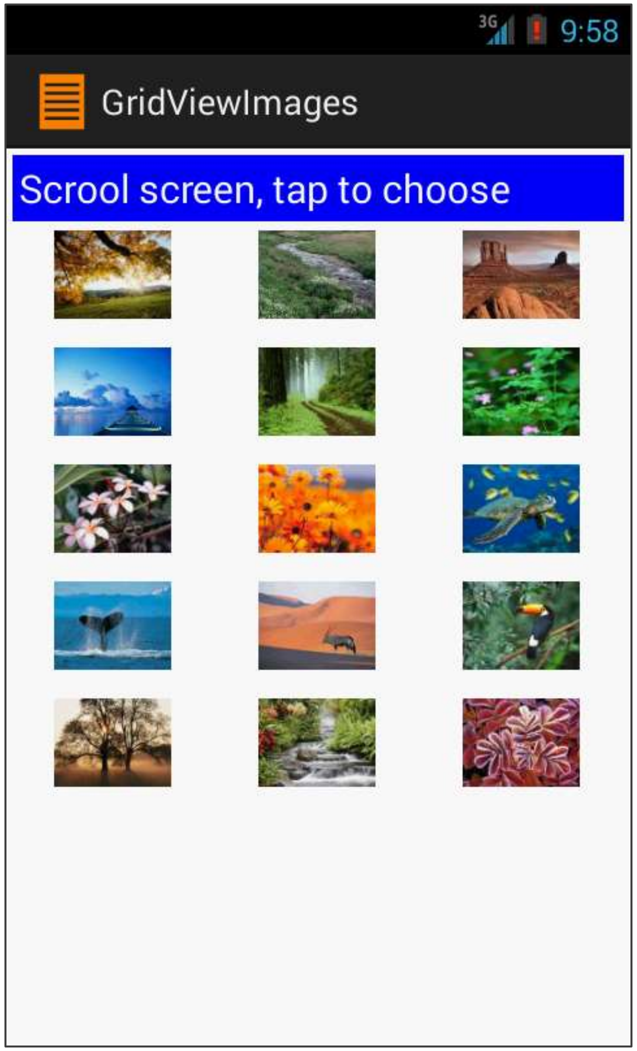


Image-based GridView

Perhaps a more interesting version of the `GridView` control involves the displaying of *images* instead of *text*.

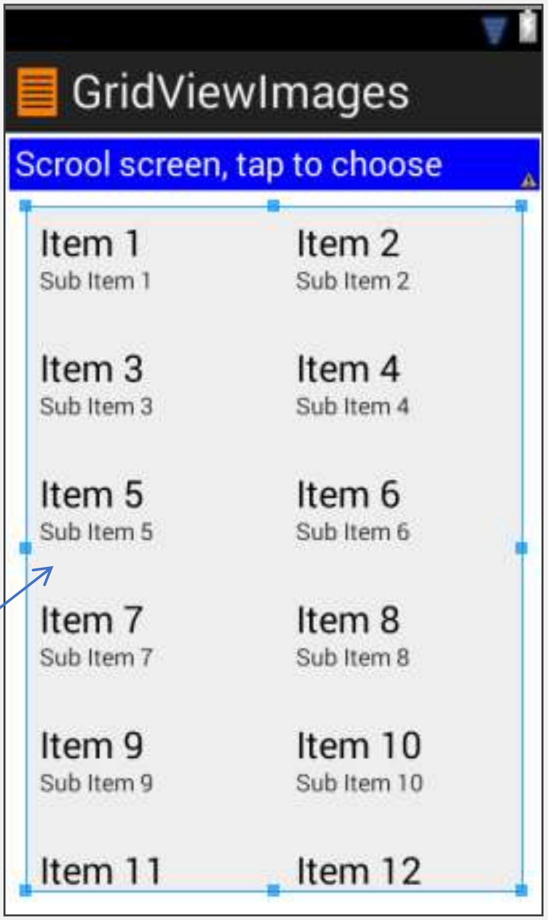
The following example illustrates how to use this control:

1. A screen shows an array of thumbnails.
2. The programmer must provide a custom data adapter to manage the displaying of thumbnails from the data set.

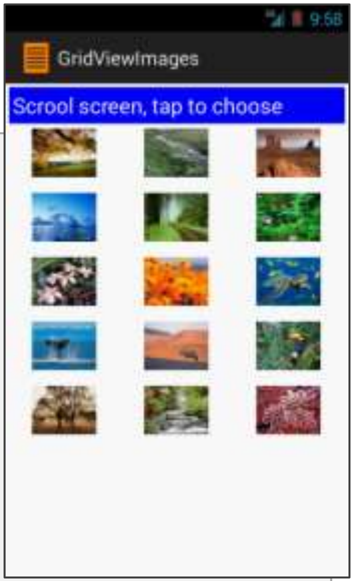


Example 6: GridView Images Demo – Layout (activity_main)

```
<?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"
    android:padding="3dp" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:padding="3dp"
        android:text="Scrool screen, tap to choose"
        android:textColor="#ffffffff"
        android:textSize="20sp" />

    <GridView
        android:id="@+id/gridview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="1dp"
        android:columnWidth="100dp"
        android:gravity="center"
        android:horizontalSpacing="5dp"
        android:numColumns="auto_fit"
        android:stretchMode="columnWidth"
        android:verticalSpacing="10dp" />
</LinearLayout>
```



Example 6: MainActivity

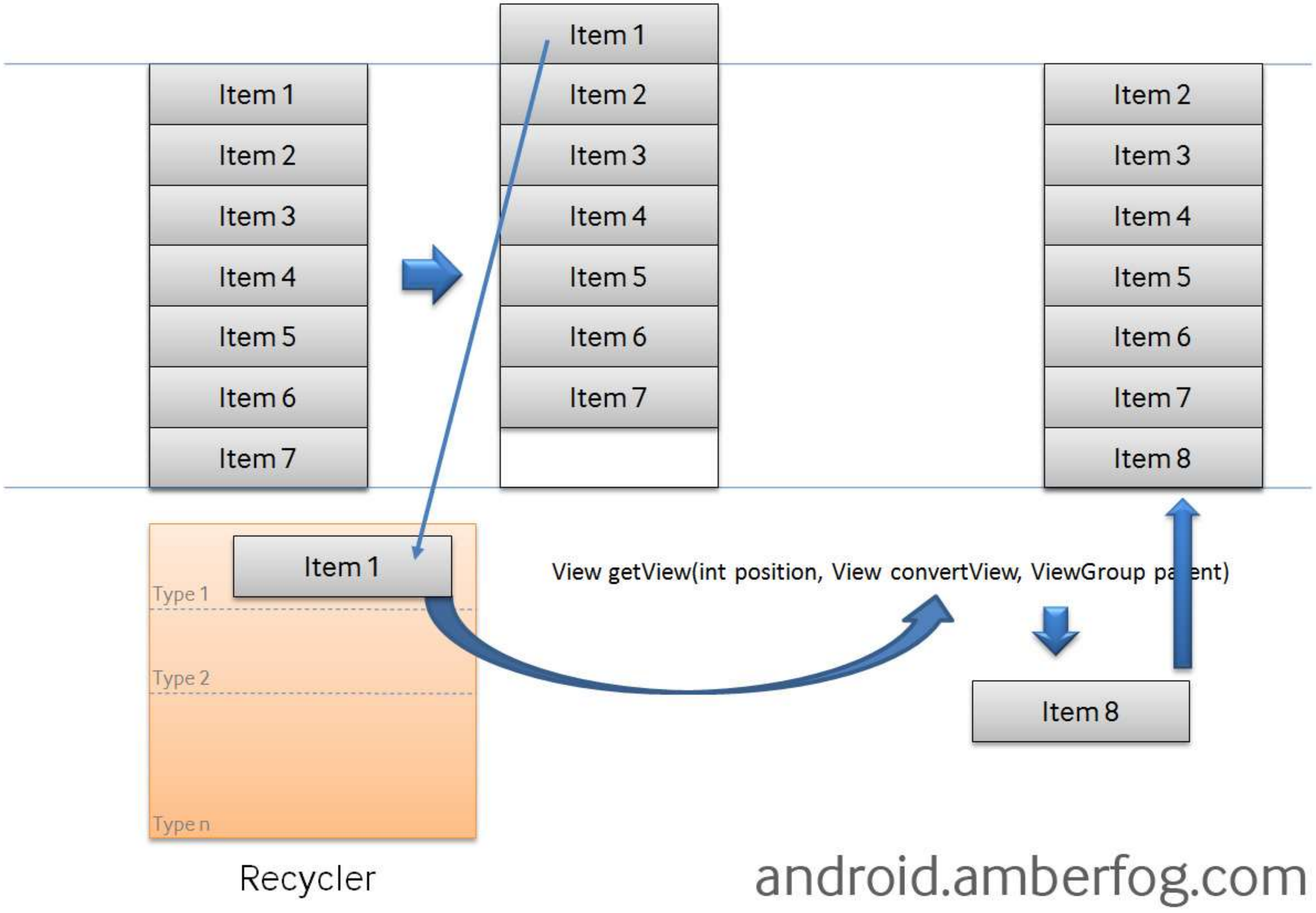


```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val thumbList = arrayListOf<Int>()
        val imageList = arrayListOf<Int>()

        // Prepare arrays of data
        for (i in 1..27) {
            thumbList.add(resources.getIdentifier("thumb$i", "drawable", packageName))
            imageList.add(resources.getIdentifier("wall$i", "drawable", packageName))
        }

        // Setup grid view
        val gridView = findViewById<GridView>(R.id.gridview)
        gridView.adapter = MyImageAdapter(thumbList)
    }
}
```

Example 6: Custom Adapter - MyImageAdapter

```
class MyImageAdapter(val thumbList: ArrayList<Int>) : BaseAdapter() {

    override fun getCount(): Int = thumbList.size

    override fun getItem(p0: Int): Any = thumbList[p0]

    override fun getItemId(p0: Int): Long = p0.toLong()

    override fun getView(p0: Int, p1: View?, p2: ViewGroup?): View {
        var imageView: ImageView
        if (p1 == null) {
            imageView = ImageView(p2?.context)
            imageView.layoutParams = LayoutParams(200, 200)
            imageView.scaleType = ImageView.ScaleType.CENTER_CROP
            imageView.setPadding(5)
        } else
            imageView = p1 as ImageView

        imageView.setImageResource(thumbList[p0])

        return imageView
    }
}
```

ListView's Recycler



Example 7: Defining your own ListViews

- Android provides several predefined row layouts for displaying simple lists (such as:
  `android.R.layout.simple_list_item_1`,
  `android.R.layout.simple_list_item_2`, etc).

- However, there are occasions in which you want a particular disposition and formatting of elements displayed in each list-row.

- In those cases, you should ***create your own subclass of a Data Adapter.***

- The next example shows how to do that.

Example 7: Create your own DataAdapter

In order to customize a Data Adapter, you need to:

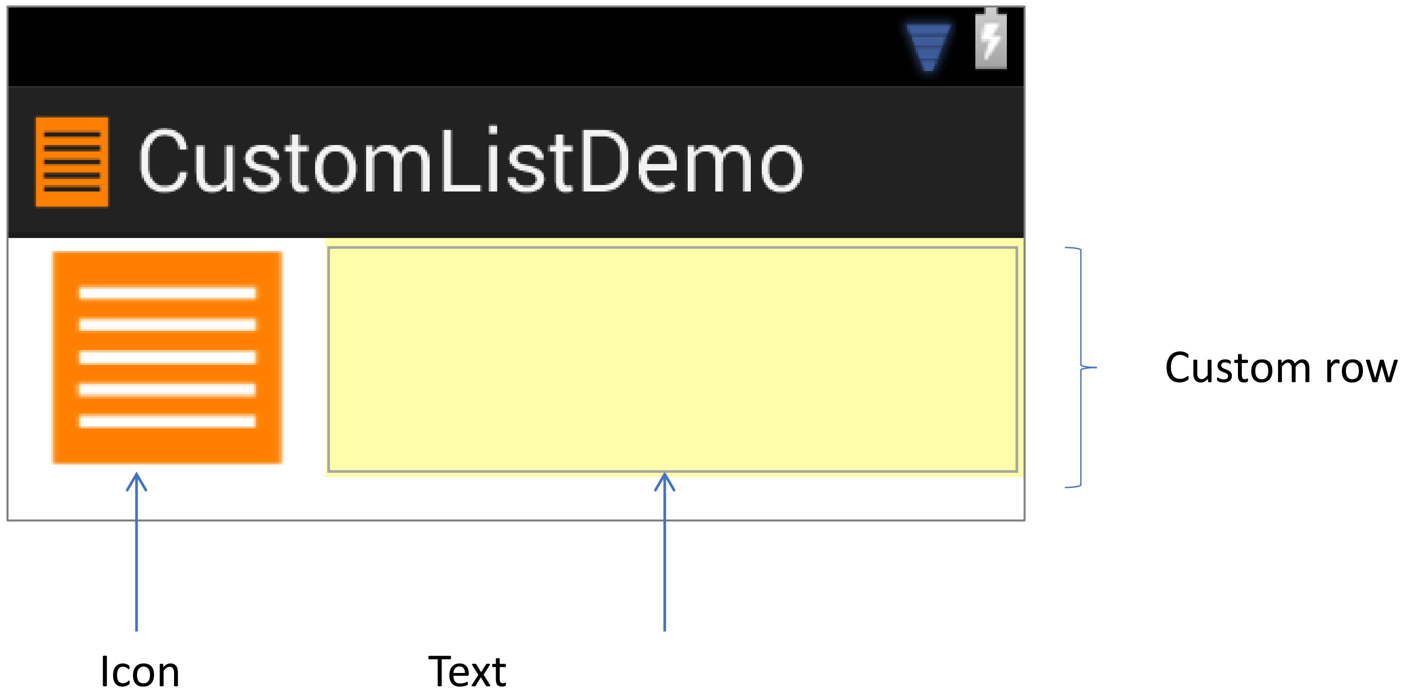
1. Create a class extending the concrete `ArrayAdapter class`
2. Override its **`getView()`**, and
3. Construct (inflate) your rows yourself.

```
class MyCustomAdapter :BaseAdapter() {
    override fun getCount(): Int {
        TODO("Not yet implemented")
    }

    override fun getItem(p0: Int): Any {
        TODO("Not yet implemented")
    }

    override fun getItemId(p0: Int): Long {
        TODO("Not yet implemented")
    }

    override fun getView(p0: Int, p1: View?, p2: ViewGroup?): View {
        TODO("Not yet implemented")
    }
}
```

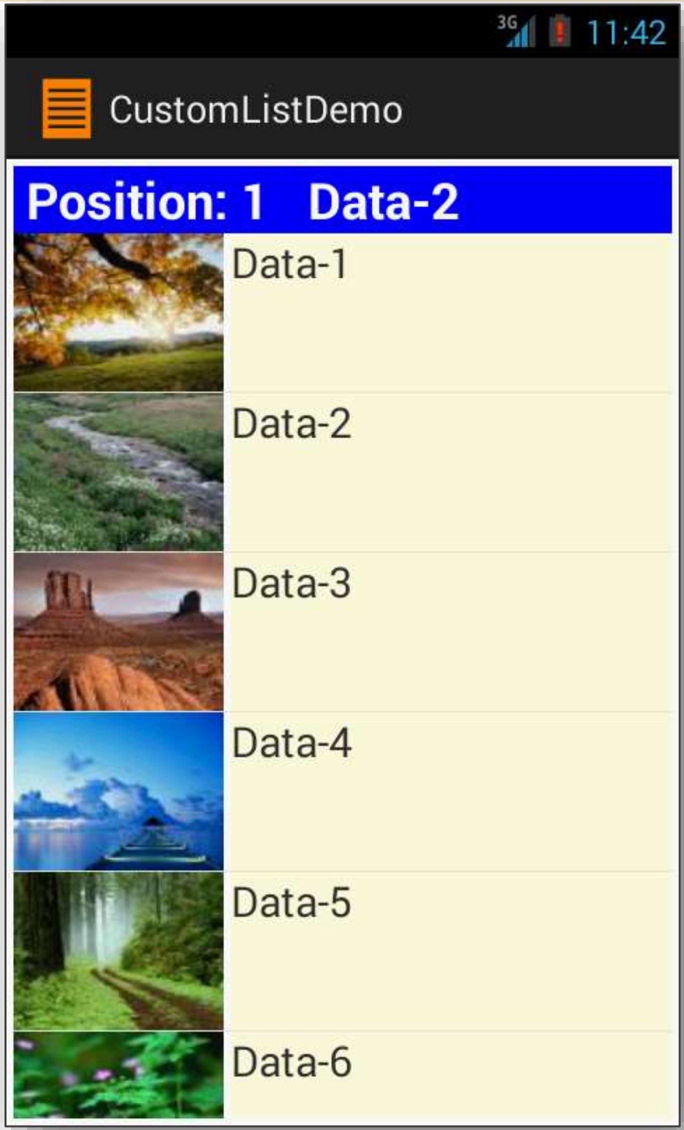
For each data element supplied by the adapter, the method **`getView()`** returns its 'visible' View.

Example 7: Designing Custom-Rows

In our example each UI row will show an icon (on the left side) and text following the icon to its right side.



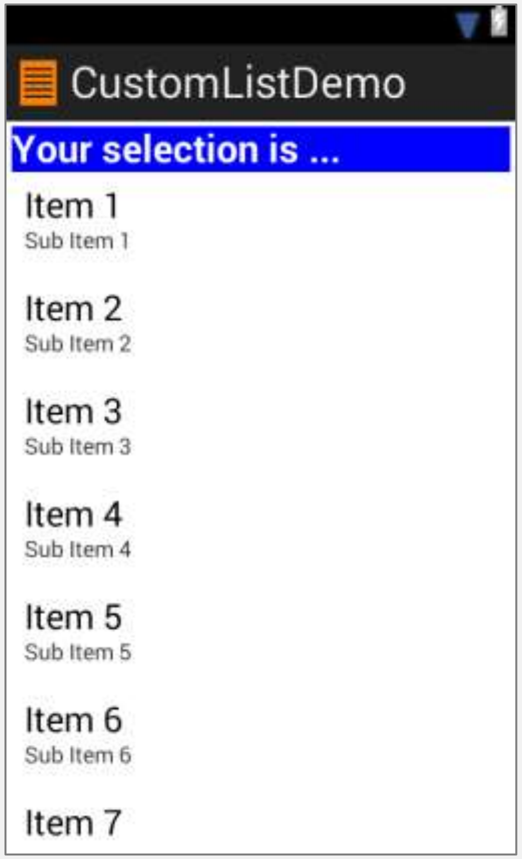
Example 7: Designing Custom-Rows



Custom row consists of icon & text

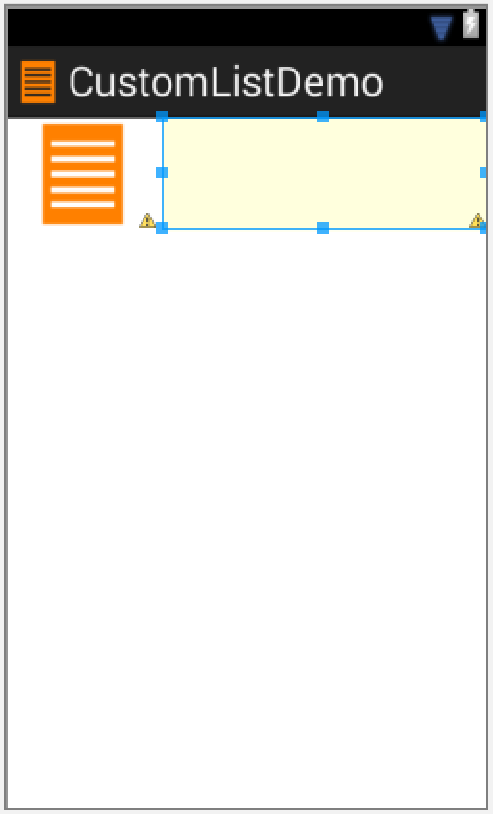
Example 7: Layout 1 – activity_main.xml

```
<?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:padding="3dp"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:text="Your selection is ..."
        android:textColor="#ffffffff"
        android:textSize="24sp"
        android:textStyle="bold" />
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```



Example 7: Layout 2 – custom_row_icon_label.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="wrap_content"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="100dp"
        android:layout_height="75dp"
        android:layout_marginRight="3dp"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="75dp"
        android:background="#22ffff00"
        android:textSize="20sp" />

</LinearLayout>
```



Example 7: MainAcitivity

```
class MainActivity : AppCompatActivity() {

    lateinit var txtMsg: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        txtMsg = findViewById<TextView>(R.id.txtMsg)

        // Prepare arrays of data
        val itemList = arrayListOf<ItemModel>()
        for (i in 1..27) {
            itemList.add(ItemModel("Data $i", resources.getIdentifier("thumb$i", 
"drawable", packageName)))
        }

        val listView = findViewById<ListView>(R.id.listView)
        listView.adapter = MyCustomAdapter(itemList)
        listView.setOnItemClickListener { adapterView, view, i, l ->  txtMsg.text = 
"Position $i: ${itemList[i]}"}
    }
}
```

Example 7: Custom ListView Demo – Adapter

```
data class ItemModel(val caption: String, val imageResource: Int)
```

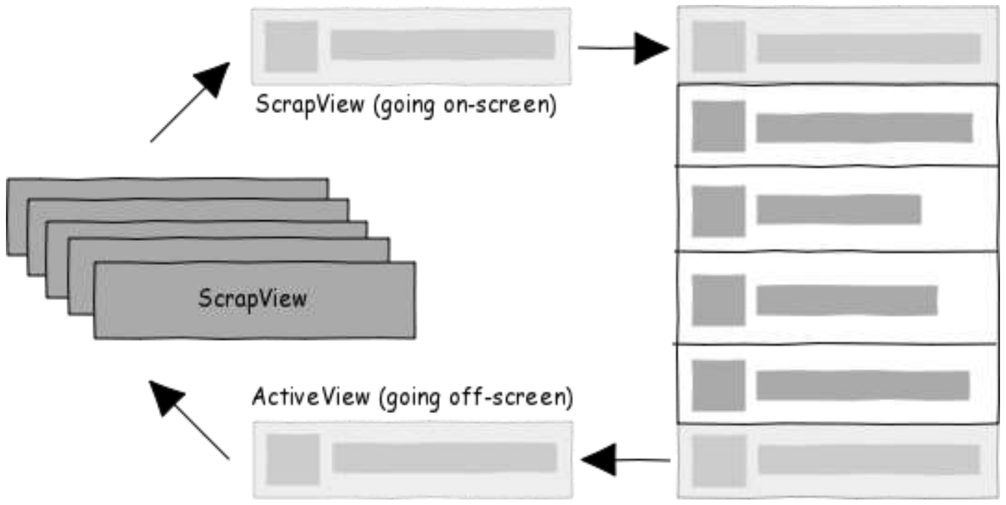
```
class MyCustomAdapter(val items: ArrayList<ItemModel>): BaseAdapter() {
    override fun getCount(): Int = items.size

    override fun getItem(p0: Int): Any = items[p0]

    override fun getItemId(p0: Int): Long = p0.toLong()

    override fun getView(p0: Int, p1: View?, p2: ViewGroup?): View {
        val row: View =
LayoutInflater.from(p2?.context).inflate(R.layout.custom_row_icon_label, p2,
false)

        val textView = row.findViewById<TextView>(R.id.label)
        val imageView = row.findViewById<ImageView>(R.id.icon)

        textView.text = items[p0].caption
        imageView.setImageResource(items[p0].imageResource)

        return row
    }
}
```

The LayoutInflater Class

- The **LayoutInflater** class converts an XML layout specification into an actual tree of View objects. The objects inflated by code are appended to the selected UI view. It typically works in cooperation with an ArrayAdapter.
- A basic **ArrayAdapter** requires three arguments: current **context**, **layout** on which output rows are shown, source data **items** (data to feed the rows).
    - The overridden **`getView()`** method inflates the row layout by custom allocating *icons* and *text* taken from data source in the user designed row.
    - Once assembled, the View (row) is returned.
    - This process is repeated for each item supplied by the ArrayAdapter.
    - Next example is a better built custom-adapter using the **ViewHolder** design strategy.

ViewHolder degisn pattern

The figure below is from "**Performance Tips for Android's ListView**" by Lucas Rocha http://lucasr.org/2012/04/05/performance-tips-for-androids-listview/ [Dec, 2014]. It shows a set of rows presented to the user inside a ListView container.



When a row gets out of sight, the memory of its layout is saved in a **scrapview** collection silently kept by the ListView.

If the row comes back to a visible state, you may reuse its scrapview skeleton instead of redoing the row from scratch.

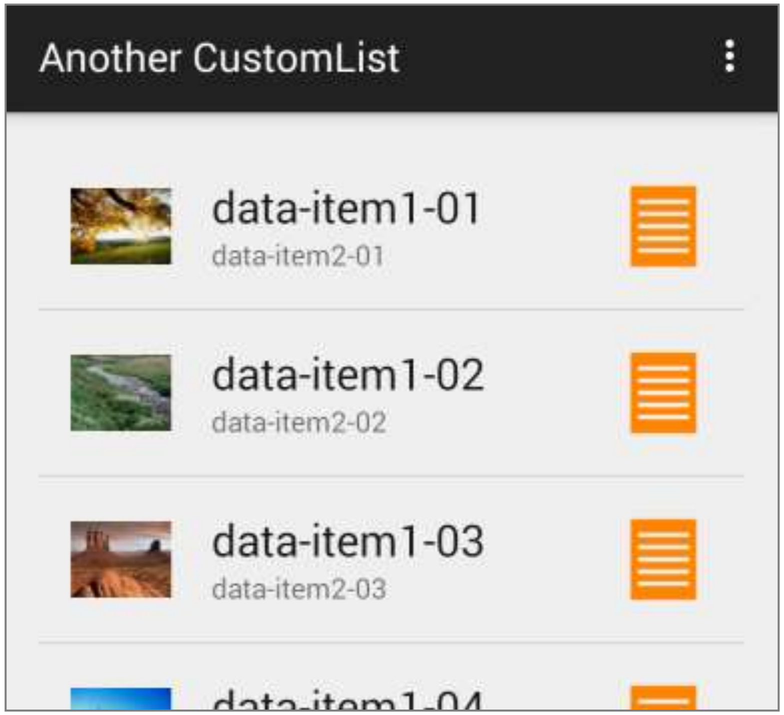
The strategy of reusing these scrapviews is known as the **ViewHolder Design Pattern.** It cuts down on the number of times you have to inflate a row-layout and then get access to its internal widgets by calling the 'findViewById()' method.

When reusing the scrapviews (made available as 'convertView') all you need to do is move the appropriate data to the internal widgets and set their onClick listeners.

ViewHolder design pattern

In this example a list holding rows showing multiple lines of text and images, is populated with a custom made BaseAdapter that uses the **ViewHolder** strategy for better performance.

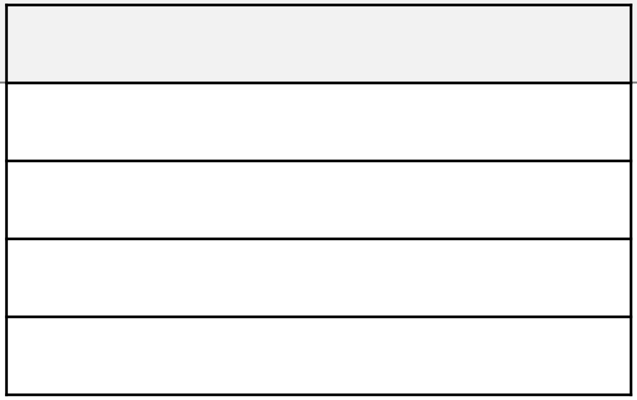
The app consists of two classes: **MainActivity** and **MyCustomAdapter**. It has two layouts: *activity_main* showing the list (see image on the right) and *custom_item_view* describing the structure of individual rows.



Example 8: Custom List – activity_main.xml

Layout *activity_main.xml* shows a ViewList.

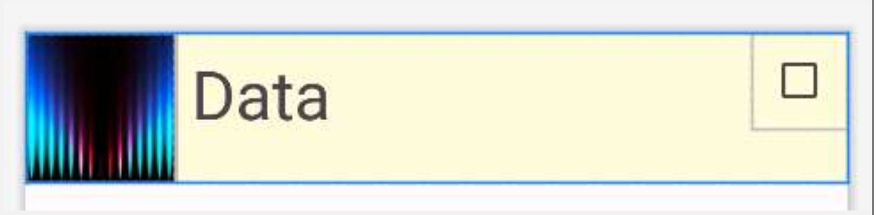
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="3dp"
    android:orientation="vertical" >
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>
</LinearLayout>
```



Example 8: Custom List – custom_item_view.xml

Layout *custom_item_view.xml* shows a custom-made row holding one image, one text and one checkbox

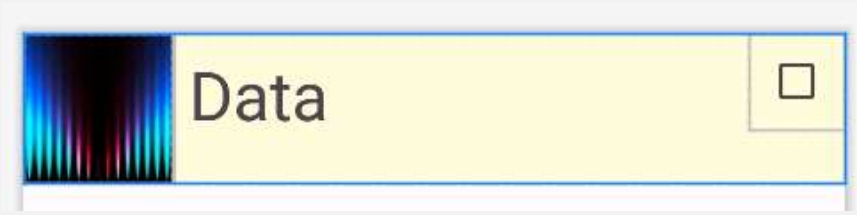
```xml
<?xml version="1.0" encoding="UTF-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/image_icon"
        android:layout_width="75dp"
        android:layout_height="75dp"
        android:src="@drawable/thumb1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```



Example 8: Custom List – custom_item_view.xml

Layout *custom_item_view.xml* shows a custom-made row holding one image, one text and one checkbox

```
    <TextView
        android:id="@+id/text_label"
        android:layout_width="0dp"
        android:layout_height="75dp"
        android:background="#22ffff00"
        android:padding="8dp"
        android:text="Data"
        android:textSize="34sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/image_icon"
        app:layout_constraintTop_toTopOf="parent" />

    <CheckBox
        android:id="@+id/check_select"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



Example 8: Custom List – MainActivity

The main activity exposes a ListView. A custom adapter is tied to the ListView. The adapter gets a reference to a test ‘database’ and the custom row layout.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Prepare arrays of data
        val itemList = arrayListOf<ItemModel>()
        for (i in 1..27) {
            itemList.add(ItemModel("Data $i",
                resources.getIdentifier("thumb$i", "drawable", packageName)))
        }

        val adapter = MyCustomAdapter(itemList)
        val listView = findViewById<ListView>(R.id.listView)
        listView.adapter = adapter
    }
}
```

Example 8: Custom List – MyCustomAdapter

The **getView** method in this extended BaseAdapter inflates a supplied row layout, gets access to its internal widgets, fills them with data and set listeners on some of them.

```
class MyCustomAdapter(val items: ArrayList<ItemModel>): BaseAdapter() {
    override fun getCount(): Int = items.size

    override fun getItem(p0: Int): Any = items[p0]

    override fun getItemId(p0: Int): Long = p0.toLong()

    override fun getView(p0: Int, p1: View?, p2: ViewGroup?): View {
        var viewHolder: MyViewHolder
        var itemView: View
        if (p1 == null) {
            itemView = LayoutInflater.from(p2?.context)
                .inflate(R.layout.custom_item_view, p2, false)
            viewHolder = MyViewHolder()
            viewHolder.textLabel = itemView.findViewById(R.id.text_label)
            viewHolder.imageIcon = itemView.findViewById(R.id.image_icon)
            viewHolder.checkSelect = itemView.findViewById(R.id.check_select)
            itemView.tag = viewHolder
        } else {
            itemView = p1
            viewHolder = itemView.tag as MyViewHolder
        }
```

Example 8: Custom List – MyCustomAdapter

The **getView** method in this extended BaseAdapter inflates a supplied row layout, gets access to its internal widgets, fills them with data and set listeners on some of them.

```
        viewHolder.textLabel.text = items[p0].caption
        viewHolder.imageIcon.setImageResource(items[p0].imageResource)
        viewHolder.checkSelect.isChecked = items[p0].selected

        viewHolder.checkSelect.setOnClickListener {
            items[p0].selected = !items[p0].selected
            notifyDataSetChanged()
        }

        return itemView
    }

    class MyViewHolder {
        lateinit var textLabel: TextView
        lateinit var imageIcon: ImageView
        lateinit var checkSelect: CheckBox
    }
}
```