Android
List-Based Selection Widgets

Victor Matos
Cleveland State University

Notes are based on:

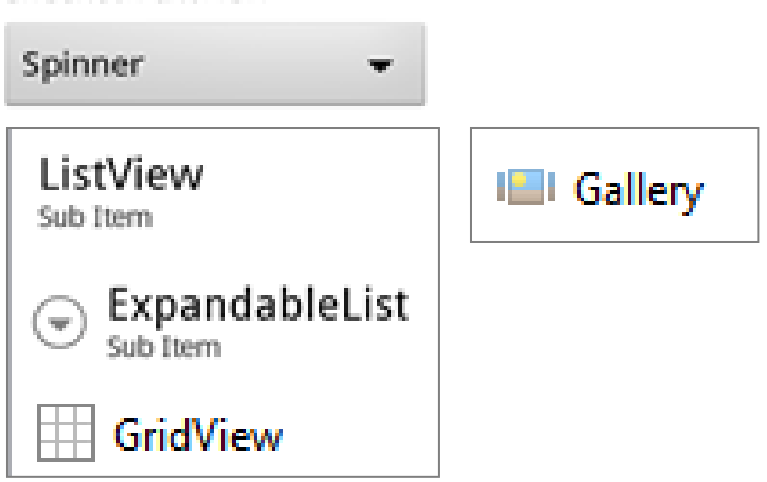
Android Developers
http://developer.android.com/index.html

List-Based Selection Widgets

- RadioButtons and CheckButtons are suitable for selecting options drawn from a *small* set of possibilities.

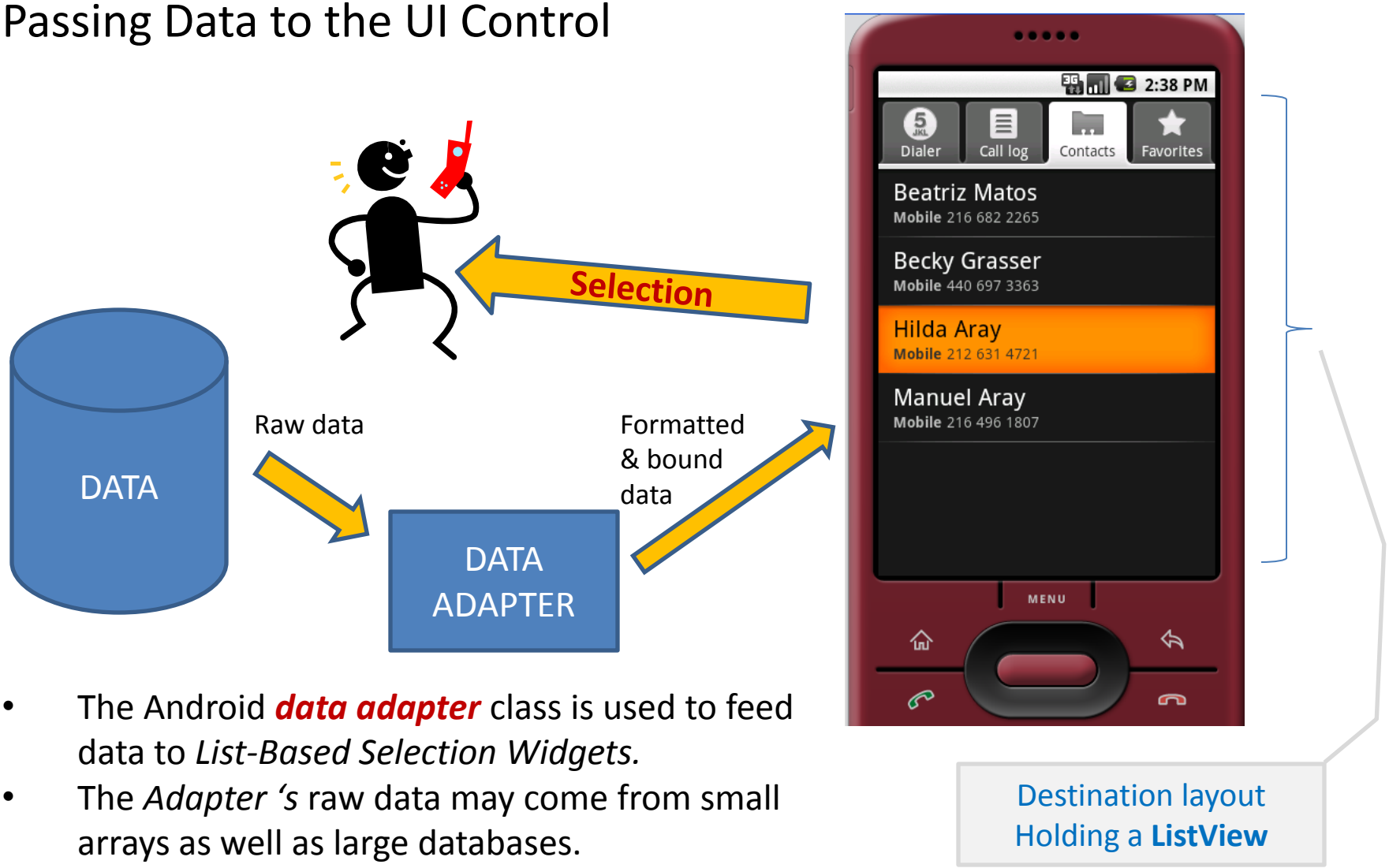
- For a larger set of choices, other Android list-based selection widgets are more appropriate.

- Example of list-based selection widgets include:
  - *ListViews,*
  - *Spinner,*
  - *GridView*
  - *Image Gallery*, etc.



List-Based Selection Widgets

Passing Data to the UI Control



- The Android ***data adapter*** class is used to feed data to *List-Based Selection Widgets.*
- The *Adapter 's* raw data may come from small arrays as well as large databases.

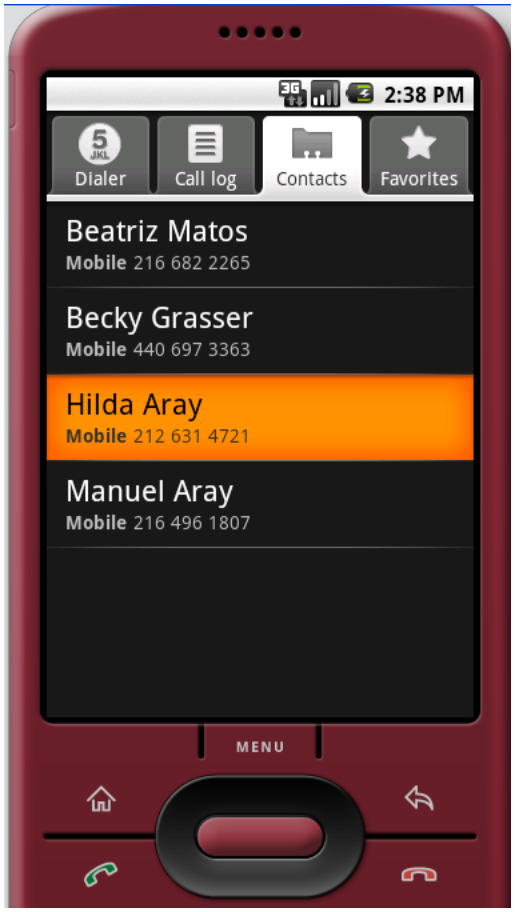
List-Based Selection Widgets

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

ListViews are scrollable, all rows from large data sets have a chance to be shown.

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 Pump

An **ArrayAdapter** can be used to wrap the contents of a Java **array** or **java.util.List** and supply data rows to the **UI.**

Raw Data

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

ArrayAdapter

```
ArrayAdapter<String> aa = new ArrayAdapter<String>(
                          this,
                          android.R.layout.simple_list_item_1,
                          android.R.id.text1,
                          items);
```

Parameters:

1. The current activity's ***Context***
2. The ListView showing the entire array (such as the built-in system resource : `android.R.layout.simple_list_item_1`).
3. The **TextView** to which an individual row is written (`android.R.id.text1`).
4. The actual **data source** (array or Java.List containing `items` to be shown).

List-Based Selection Widgets

```
<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>
```

Example 1: A simple list (1 of 4)

A list of array data will be displayed.
Rows consists of a simple line of text.
The txtMsg TextView will display the user's selection (after tapping a row)

Pay attention to the `@android:id/` predefined entries used here.

Android's built-in list layout

Used for empty lists

List-Based Selection Widgets

Example 1 : A simple list (2 of 4)

CAUTION:
A ListActivity is not a *common* Activity. It is bound to a ListView

```
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 = {};
```

Data Source

NOTE**: The **ListActivity** class is implicitly bound to an object identified by **@android:id/list

List-Based Selection Widgets

Example 1: A simple list (3 of 4)

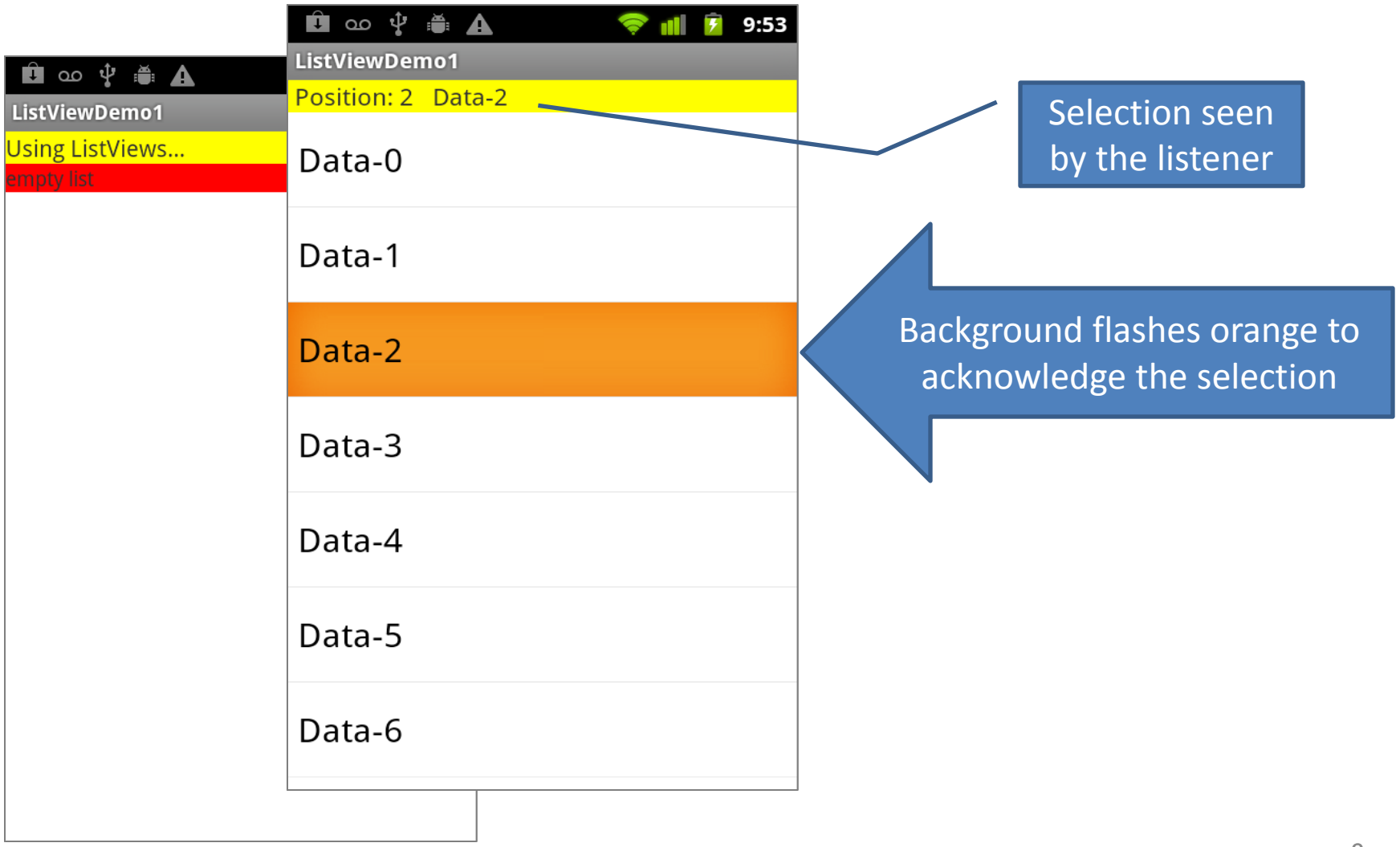
```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view_demo1);

        setListAdapter(new ArrayAdapter<String>(this,
                                                android.R.layout.simple_list_item_1,
                                                android.R.id.text1,
                                                items));

        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

List-Based Selection Widgets

Example 1: A simple list (4 of 4)



List-Based Selection Widgets

Comments on Example1.

This example uses a number of predefined Android components.

1. A *ListActivity* references its ViewGroup with the dedicated name: **android:id/list**
2. Later in the setting of the ArrayAdapter we use **android.R.layout.*simple_list_item_1*** (one row identified as ***text1*** displayed for each item supplied by the adapter)
3. The entry **android.R.id.text1** names the destination field in the UI for a single item.

Android SDK includes a number of predefined layouts/styles, they can be found in the folders

C:\ Your-Path \Android\android-sdk\platforms\android-xx\data\res\layout
C:\ Your-Path \Android\android-sdk\platforms\android-xx\data\res\values\styles.xml

*See **Appendix A** for more on this issue.*

List-Based Selection Widgets

Comments on Example1.

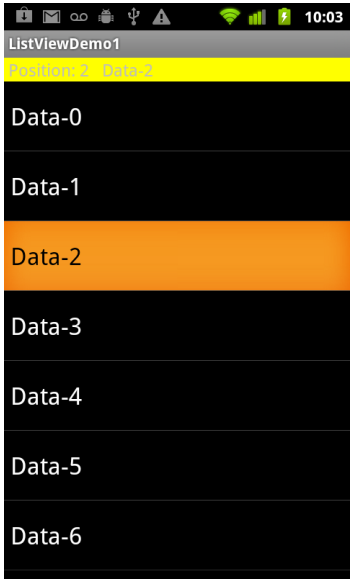
A little experiment

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. Run the application again. Observe its new look.



List-Based Selection Widgets

Comments on Example1.

A second little experiment

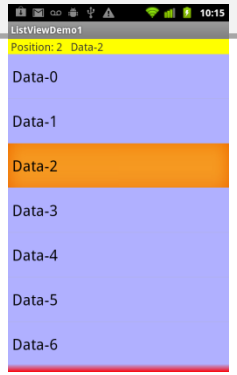
1. Open the AndroidManifest.xml file. Under the <Application> tag add the clause `android:theme="@style/CustomTheme"`

2. Open the file **res/values/styles/style.xml**. Add to the <resource> tag the following lines

```
<color name="custom_theme_color1">#b0b0ff</color>
<color name="custom_theme_color2">#ff0000</color>

<style name="CustomTheme" parent="android:Theme.Light">
    <item name="android:windowBackground">@color/custom_theme_color1</item>
    <item name="android:colorBackground">@color/custom_theme_color2</item>
</style>
```

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

4. Style specs are quite similar to CSS. More on styles later.



List-Based Selection Widgets

A Variation on Example1.

- You may use a common **Activity** class instead of a **ListActivity**.
- The Layout below uses a ListView identified as @+id/my_list (instead of **@android:id/list** used in the previous example).

```
<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>
```

List-Based Selection Widgets

A Variation on Example1.

Bind the Java ListView control to your XML *my_list* widget. Set the adapter to your new ListView control.

```
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_list_view_demo2);
          myListView = (ListView) findViewById(R.id.my_list);

          ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
                                    android.R.layout.simple_list_item_1,
                                    android.R.id.text1,
                                    items);
          myListView.setAdapter(aa);

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

List-Based Selection Widgets

A Variation on Example1 (*cont.*)

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

```
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);
        }
        });
```

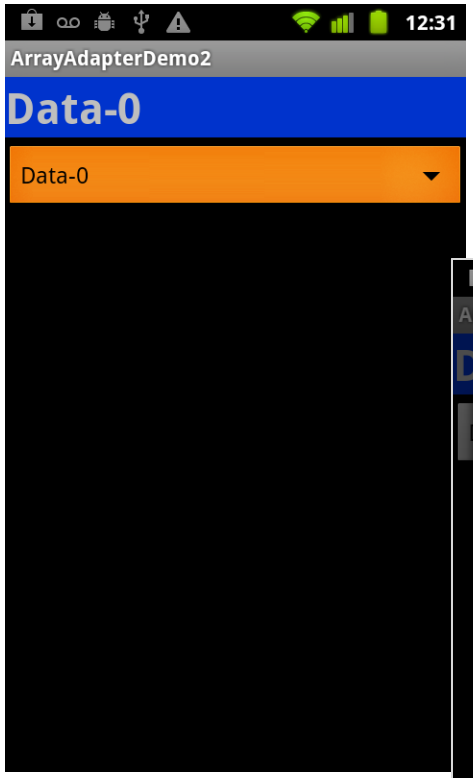
List-Based Selection Widgets

Spin Control

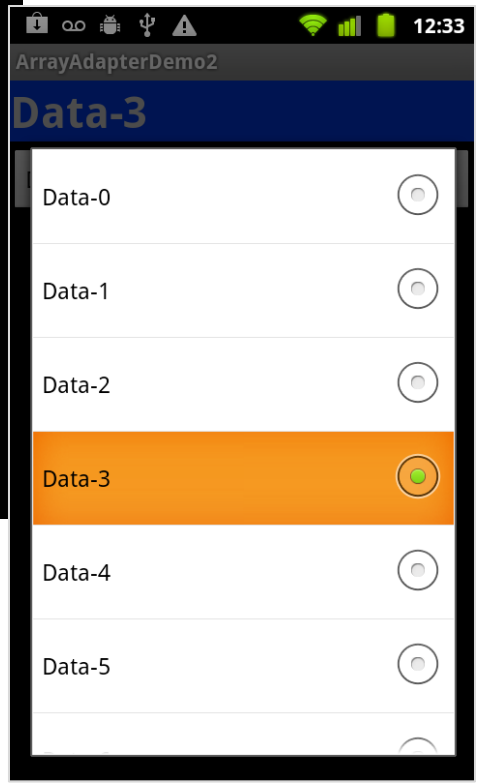
- Android's **Spinner** is equivalent to a *drop-down* selector.
- Spinners have the same functionality of a ListView but take less screen space.
- An Adapter is used to supply its data using ***setAdapter(…)***
- A listener captures selections made from the list with ***setOnItemSelectedListener***(…).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

List-Based Selection Widgets

Example 2. Using the Spinner



1. Click here



2. Select this option

3. Selected value

Images taken from a device running Gingerbread 2.3

List-Based Selection Widgets

Example 2. Using the Spinner

Image taken from a device running IceCream Sandw. 4.1

List-Based Selection Widgets

Example 2. Using the Spinner

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

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="3dp"
        android:background="#ff0033cc"
        android:textSize="30sp"
        android:textStyle="bold" >
    </TextView>

    <Spinner
        android:id="@+id/my_spinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >
    </Spinner>

</LinearLayout>
```

List-Based Selection Widgets

Example 2. Using the Spinner

```
public class ArrayAdapterDemo2 extends Activity
              implements AdapterView.OnItemSelectedListener {

   TextView txtMsg;

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


  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    setContentView(R.layout.main);
    txtMsg = (TextView) findViewById(R.id.txtMsg);
```

List-Based Selection Widgets

Example 2. Using the Spinner

```
    Spinner spin = (Spinner) findViewById(R.id.my_spinner);
    spin.setOnItemSelectedListener(this);

    // bind array to UI control to show one-line from array
    ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
                                                android.R.layout.simple_spinner_item,
                                                items);

    // showing the drop-down multi-line window
    aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    // bind everything together
    spin.setAdapter(aa);

  }// onCreate

  public void onItemSelected(AdapterView<?> parent, View v, int position, long id) {
    txtMsg.setText(items[position]);
  }

  public void onNothingSelected(AdapterView<?> parent) {
    txtMsg.setText("");
  }
}// class
```

List-Based Selection Widgets

Example 2B. Using the Spinner - A Code Variation

```
public class ArrayAdapterDemo2 extends Activity {
  TextView txtMsg;
  String[] items = { "Data-0", "Data-1", "Data-2", "Data-3",
                     "Data-4", "Data-5", "Data-6", "Data-7" };
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    txtMsg = (TextView) findViewById(R.id.txtMsg);

    Spinner spin = (Spinner) findViewById(R.id.my_spinner);
    spin.setOnItemSelectedListener(new OnItemSelectedListener() {
              @Override
              public void onItemSelected(AdapterView<?> container, View row,
                                         int position, long id) {
                txtMsg.setText(items[position]);
              }

              @Override
              public void onNothingSelected(AdapterView<?> container) {
                txtMsg.setText("");
              }
    });
```

List-Based Selection Widgets

Example 2B. Using the Spinner - A Code Variation

```
    // bind array to UI control to show one-line from array
    ArrayAdapter<String> aa = new ArrayAdapter<String>(
                                          this,
                                          android.R.layout.simple_spinner_item,
                                          items);

    // showing the drop-down multi-line window
    aa.setDropDownViewResource( android.R.layout.simple_spinner_dropdown_item );

    // bind everything together
    spin.setAdapter(aa);

  }// onCreate

}// class
```

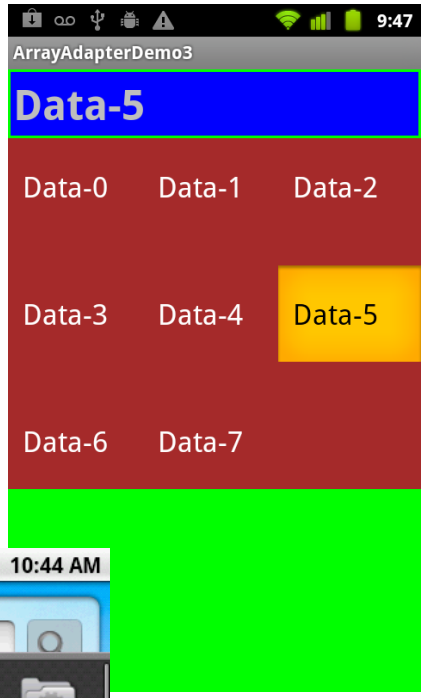
List-Based Selection Widgets

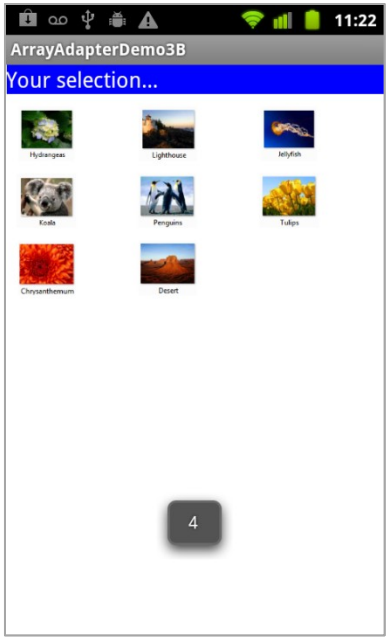
GridView

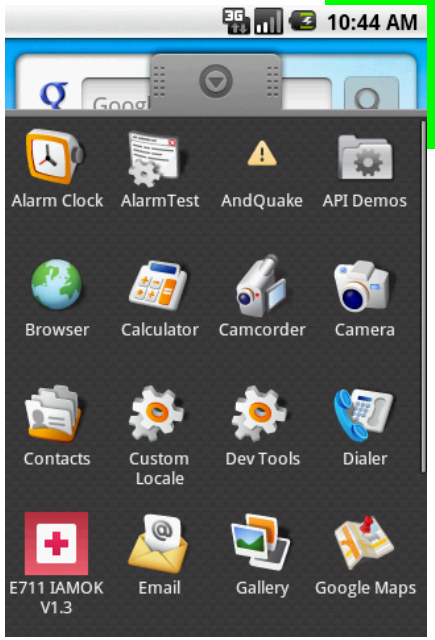
GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

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

Grid cells can show text and images







List-Based Selection Widgets

GridView

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 .

List-Based Selection Widgets

GridView

Example: Fitting the View
Suppose the screen is **320** (dip) pixels wide, and we have

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

Three columns would use **310** pixels (three columns of 100 pixels and two whitespaces of 5 pixels).

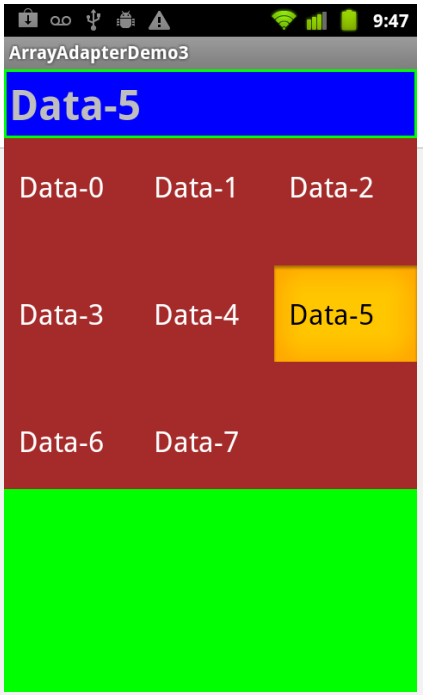
With *android:stretchMode* set to *columnWidth*, the three columns will each expand by 3-4 pixels to use up the remaining 10 pixels.

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

List-Based Selection Widgets

Example 3A. GridView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff00ff00"
    >
    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:textSize="14pt"
        android:textStyle="bold"
        android:layout_margin="2dip"
        android:padding="2dip"   />

    <GridView
        android:id="@+id/grid"
        android:background="#ffA52A2A"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:verticalSpacing="35dip"
        android:horizontalSpacing="5dip"
        android:numColumns="auto_fit"
        android:columnWidth="100dip"
        android:stretchMode="spacingWidth"  />
</LinearLayout>
```



List-Based Selection Widgets

Example 3A. GridView

```
public class ArrayAdapterDemo3 extends Activity  {

        TextView txtMsg;

        String[] items = { "Data-0", "Data-1", "Data-2", "Data-3",
                           "Data-4", "Data-5", "Data-6", "Data-7" };
        @Override
        public void onCreate(Bundle icicle) {
                super.onCreate(icicle);
                setContentView(R.layout.main);

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

List-Based Selection Widgets

Example 3A. GridView

```
      GridView grid = (GridView) findViewById(R.id.grid);

      ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            this,
            android.R.layout.simple_list_item_1,
            android.R.id.text1,
            items );

      grid.setAdapter(adapter);

      grid.setOnItemClickListener(new OnItemClickListener() {

         @Override
         public void onItemClick(AdapterView<?> container, View v,
                                 int position, long id) {
            txtMsg.setText(items[position]);
         }
      });

   }//onCreate

}// class
```

List-Based Selection Widgets

AutoCompleteTextView

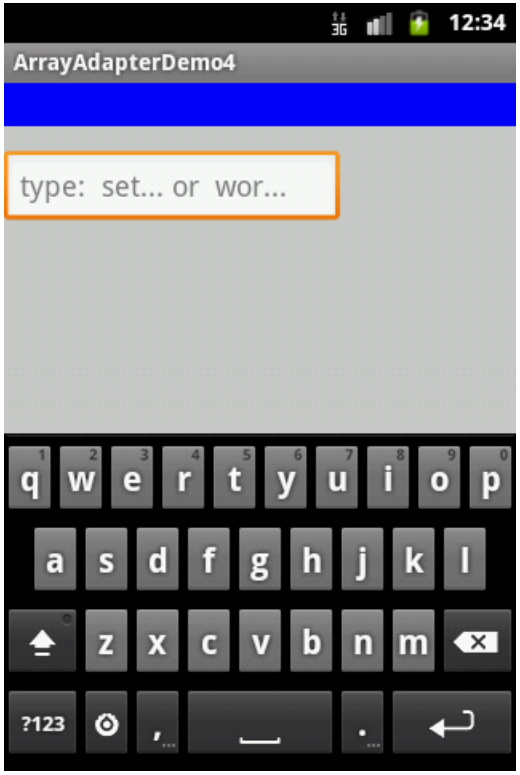
- Characters typed so far are compared with the beginning of words held in a user-supplied list of *suggested* values.
- Suggestions matching the typed prefix are shown in a *selection list.*
- The user can choose from the suggestion list or complete typing the word.
- The ***android:completionThreshold*** 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.

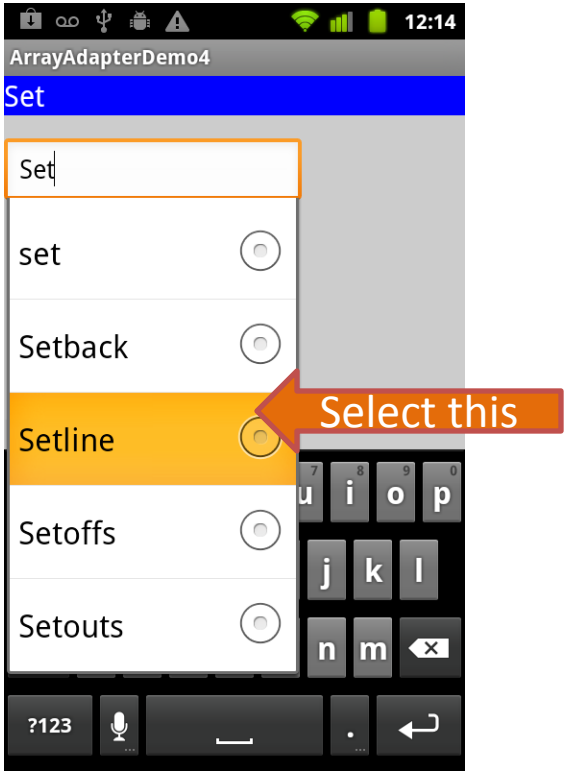
NOTE: For other features of the TextView control see Appendix B

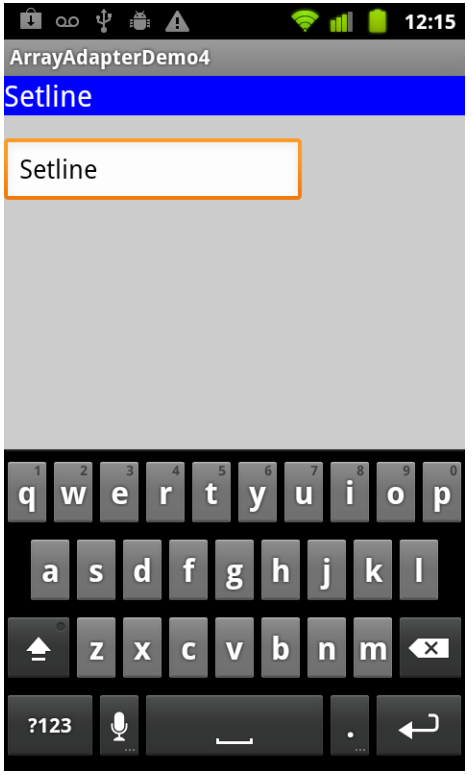
30

List-Based Selection Widgets

AutoCompleteTextView







List-Based Selection Widgets

Example 4. AutoCompleteTextView

```
<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"
    android:background="#ffcccccc">
    <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

List-Based Selection Widgets

Example 4. AutoCompleteTextView

```
public class ArrayAdapterDemo4 extends Activity implements TextWatcher {

  TextView txtMsg;

  AutoCompleteTextView txtAutoComplete;

  String[] items = { "words", "starting", "with", "set", "Setback",
      "Setline", "Setoffs", "Setouts", "Setters", "Setting",
      "Settled", "Settler", "Wordless", "Wordiness", "Adios" };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

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

new

List-Based Selection Widgets

Example 4. AutoCompleteTextView

```
  txtAutoComplete = (AutoCompleteTextView) findViewById(
                                                R.id.autoCompleteTextView1);
  txtAutoComplete.addTextChangedListener(this);

  txtAutoComplete.setAdapter(new ArrayAdapter<String>(
                            this,
                            android.R.layout.simple_list_item_single_choice,
                            items));
  }//onCreate

  public void onTextChanged(CharSequence s, int start, int before, int count) {
    txtMsg.setText(txtAutoComplete.getText());
  }

  public void beforeTextChanged(CharSequence s, int start, int count,int after) {
    // needed for interface, but not used
  }

  public void afterTextChanged(Editable s) {
    // needed for interface, but not used
  }

}//class
```

List-Based Selection Widgets

Gallery Widget

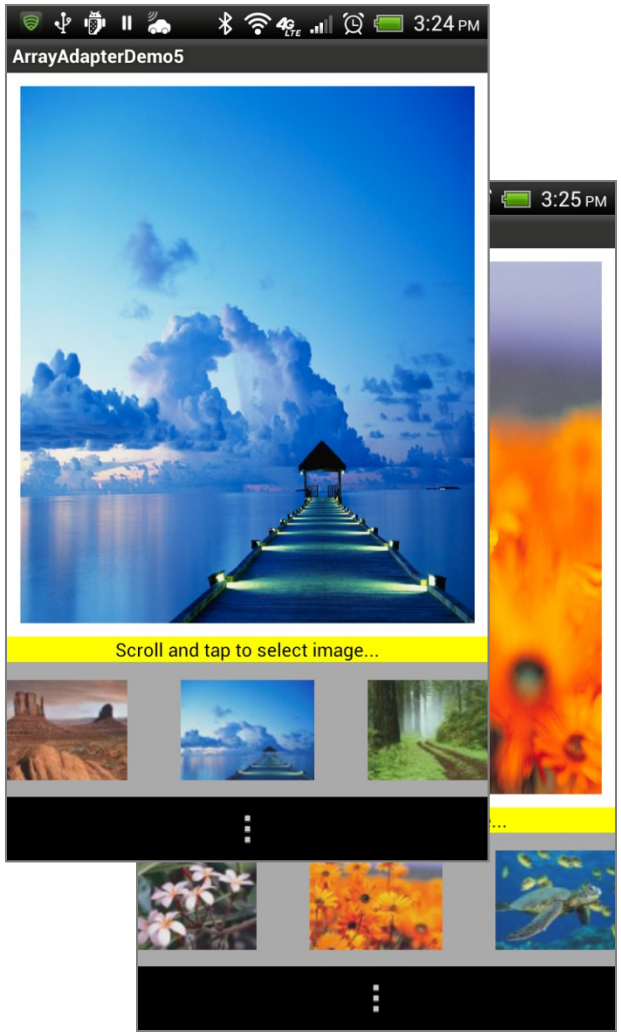
- The Gallery widget provides a set of options depicted as images.
- Image choices are offered on a contiguous horizontal mode, you may scroll across the image-set.

CAUTION ☹
Jelly-Bean 4.1x has deprecated this control. Instead the ViewPager and HorizontalScrollView are suggested (an example is provided at the end of this lesson).

List-Based Selection Widgets

Example 5 - Gallery Widget

- In this example we will place a Gallery view at the bottom of the screen
  (it is better to use small thumbnail images there)
- The user will scroll through the options and tap on a selection
- A 'better' (and larger) version of the selected picture will be displayed in an ImageView widget
- The programmer must supply a modified **BaseAdapter** to indicate what to do when an individual image is selected/clicked.



Note: For a free thumbnail-image making, visit the site: http://makeathumbnail.com

List-Based Selection Widgets

Example 5 - Gallery Widget

- The images used in this example were taken from the **Windows Vista** folder: **c:/Users/…/Documents/My Pictures**

- Equivalent 100x100 thumbnails were made visiting the site: http://makeathumb.com

List-Based Selection Widgets

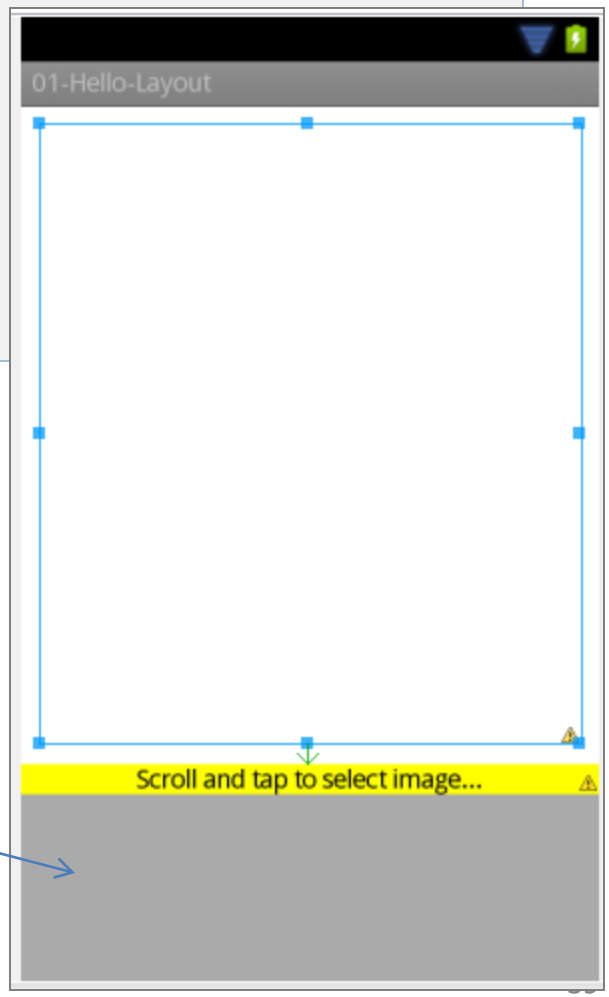
Example 5 - Gallery Widget

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/background_light"  >

    <ImageView
        android:id="@+id/selected_image"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_above="@+id/my_bottom_layout"
        android:layout_margin="10dip"  />

    <RelativeLayout
        android:id="@+id/my_bottom_layout"
        android:layout_width="fill_parent"
        android:layout_height="120dip"
        android:layout_alignParentBottom="true"
        android:background="@android:color/darker_gray" >

        <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:background="#ffffff00"
        android:text="Scroll and tap to select image..."
        android:textColor="#ff000000"
        android:gravity="center"/>
```

List-Based Selection Widgets

Example 5 - Gallery Widget

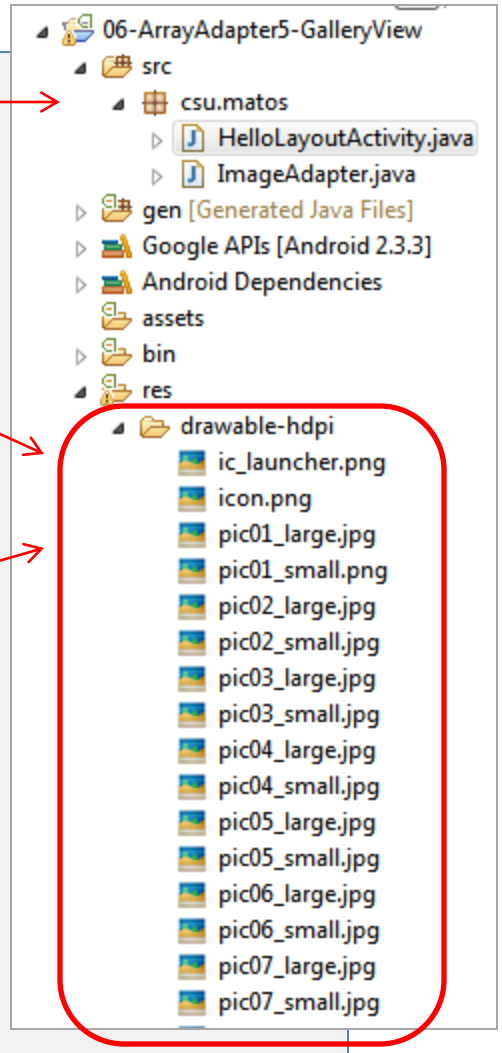
```
        <Gallery
            android:id="@+id/gallery"
            android:layout_width="match_parent"
            android:layout_height="150dip"
            android:layout_marginTop="20dip"
            android:layout_alignParentBottom="true"
            android:spacing="40dip" />

    </RelativeLayout>

</RelativeLayout>
```



List-Based Selection Widgets

Example 5 - Gallery Widget

```
public class HelloLayoutActivity extends Activity {
    // list of images copied to your res/drawable app folder
    Integer[] thumbNail = { R.drawable.pic01_small,
           R.drawable.pic02_small, R.drawable.pic03_small,
           R.drawable.pic04_small, R.drawable.pic05_small,
           R.drawable.pic06_small, R.drawable.pic07_small,
           R.drawable.pic08_small, R.drawable.pic09_small,
           R.drawable.pic10_small, R.drawable.pic11_small,
           R.drawable.pic12_small, R.drawable.pic13_small,
           R.drawable.pic14_small, R.drawable.pic15_small };

    Integer[] largeImages = { R.drawable.pic01_large,
           R.drawable.pic02_large, R.drawable.pic03_large,
           R.drawable.pic04_large, R.drawable.pic05_large,
           R.drawable.pic06_large, R.drawable.pic07_large,
           R.drawable.pic08_large, R.drawable.pic09_large,
           R.drawable.pic10_large, R.drawable.pic11_large,
           R.drawable.pic12_large, R.drawable.pic13_large,
           R.drawable.pic14_large, R.drawable.pic15_large };

    ImageView selectedImage;

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

        selectedImage = (ImageView) findViewById(R.id.selected_image);
```



List-Based Selection Widgets

Example 5 - Gallery Widget

```
        Gallery gallery = (Gallery) findViewById(R.id.gallery);

        // Set the adapter to our custom adapter
        gallery.setAdapter(new ImageAdapter(this, thumbNail));

        // Set an item click-listener and wait for user to make selection
        gallery.setOnItemClickListener(new OnItemClickListener() {

            public void onItemClick(AdapterView<?> parent, View v,int position, long id) {

                // draw selected image (if possible use better resolution);
                BitmapDrawable bitmapDrawable =
                        (BitmapDrawable) getResources().getDrawable( largeImages[position] );

                Bitmap bm = Bitmap.createScaledBitmap(bitmapDrawable.getBitmap(),
                        (int) (bitmapDrawable.getIntrinsicHeight() * 1.0),
                        (int) (bitmapDrawable.getIntrinsicWidth() * 1.0), false);

                selectedImage.setImageBitmap(bm);
                selectedImage.setScaleType(ScaleType.FIT_XY);
               //ABOVE SIMILAR TO ==> selectedImage.setImageResource(largeImages[position]);
            }
        });

    }// onCreate

}//class
```

List-Based Selection Widgets

Example 5 - Gallery Widget – Custom ImageAdapter

```
public class ImageAdapter extends BaseAdapter {

        private Context mContext;
        private Integer[] thumbNail;

        public ImageAdapter(Context c, Integer[] thumbNail) {
                mContext = c;
                this.thumbNail = thumbNail;
        }

        public int getCount() {
                return thumbNail.length;
        }

        public Object getItem(int position) {
                return position;
        }

        public long getItemId(int position) {
                return position;
        }

        public View getView(int position, View convertView, ViewGroup parent) {
                ImageView iv = new ImageView(mContext);
                iv.setImageResource(thumbNail[position]);
                iv.setLayoutParams(new Gallery.LayoutParams(150, 100));
                iv.setScaleType(ImageView.ScaleType.FIT_XY);
                return iv;
        }
}//class
```

A resource image whose name is taken from the thumbnail array will be used to fill a view. Our view is XY-fitted an set to 150x100 pixels

List-Based Selection Widgets

Example 5 - Gallery Widget – Your turn!

Optional little experiment!
Explore the **onTouch** event. Add the code below to your onCreate method.

```
// item touched
    gallery.setOnTouchListener(new OnTouchListener() {

      @Override
      public boolean onTouch(View v, MotionEvent event) {
         switch ( event.getAction() ) {
            case MotionEvent.ACTION_DOWN:
            // finger touches the screen
            break;

            case MotionEvent.ACTION_MOVE:
            // finger moves on the screen
            break;

            case MotionEvent.ACTION_UP:
            // finger leaves the screen
            break;
            }
         txtMsg.append ( "\n Touched... " + event.getAction() );
         return false;
      }
});
```

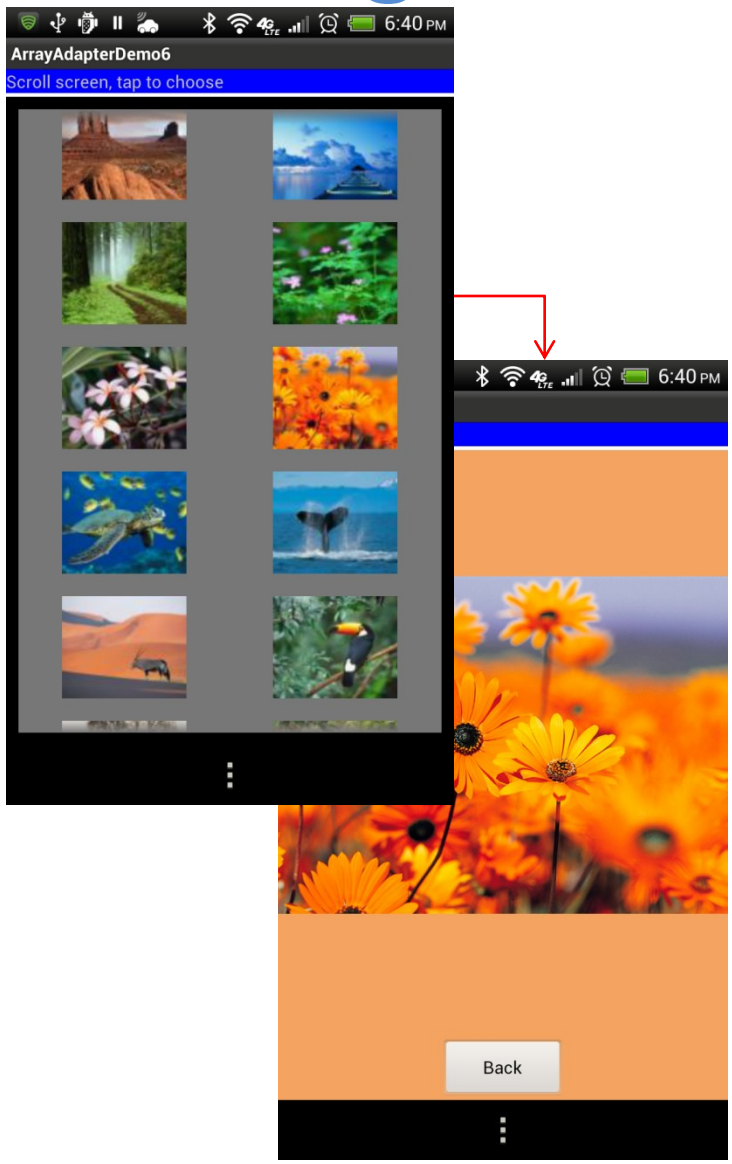
List-Based Selection Widgets

Example 6 - GridViews (again…)

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

The following example illustrates how to use this control:

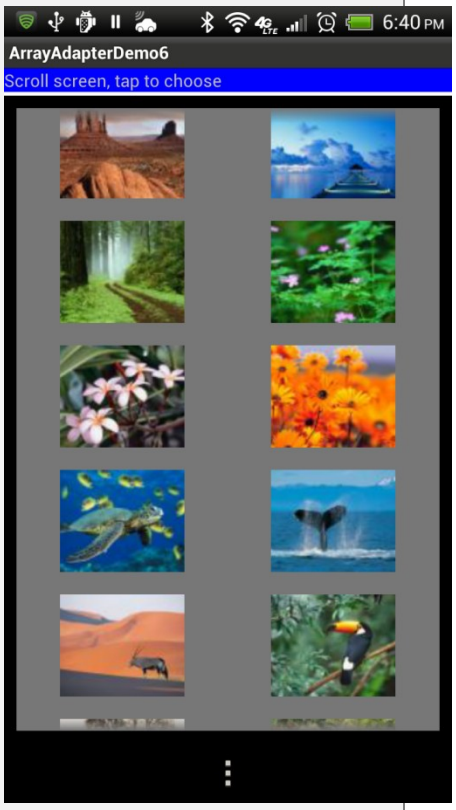
1. A screen shows an array of thumbnails.
2. The user taps on one of them and,
3. The app displays on a new screen a bigger/better image of the selected option.



List-Based Selection Widgets

Example 6 - GridView (again…) main.xml

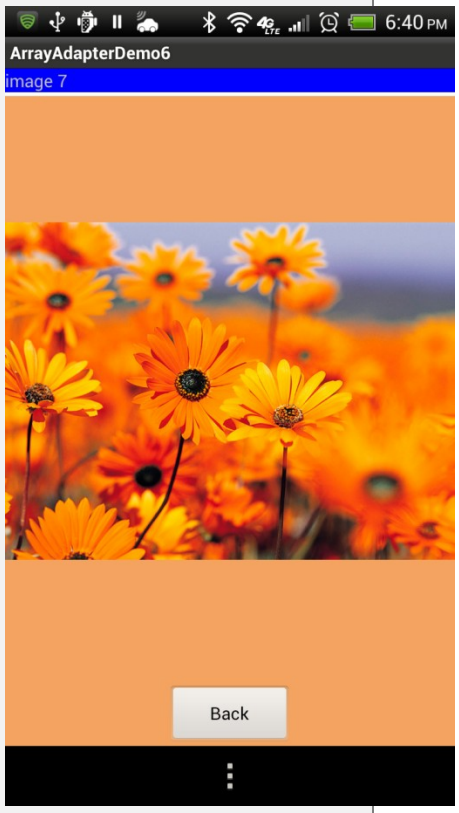
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"  >
<TextView
    android:text="Scrool screen, tap to choose"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#ff0000ff"  />
<View
    android:layout_width="fill_parent"
    android:layout_height="3dip"
    android:background="#ffffffff" />

<GridView
            android:id="@+id/gridview"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_margin="10dip"
            android:numColumns="2"
            android:verticalSpacing="10dp"
            android:horizontalSpacing="10dp"
            android:columnWidth="90dp"
            android:stretchMode="columnWidth"
            android:gravity="center"
            android:background="#ff777777" />
</LinearLayout>
```



List-Based Selection Widgets

Example 6 - GridView (again…) solo_picture.xml

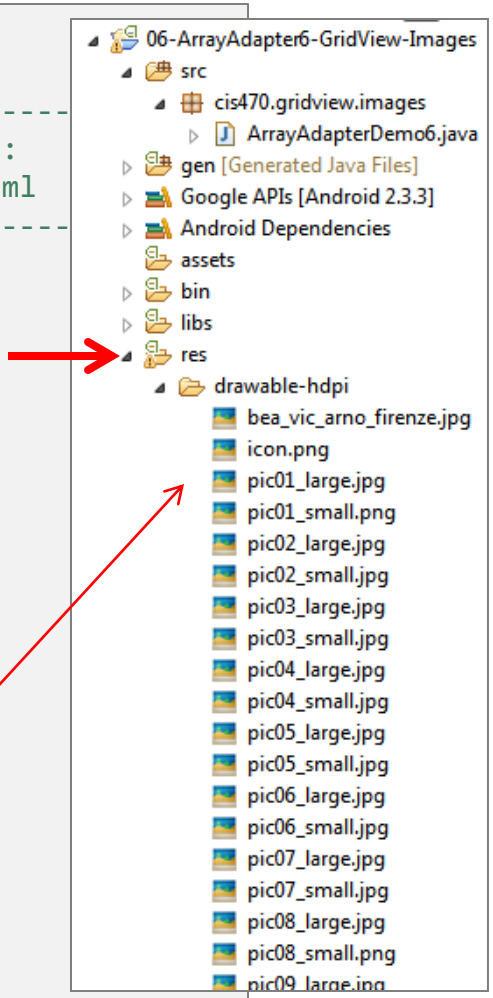
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ffF4A460"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtSoloMsg"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff" />

    <View
        android:layout_width="fill_parent"
        android:layout_height="3dip"
        android:background="#ffffffff" />

    <ImageView
        android:id="@+id/imgSoloPhoto"
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:layout_gravity="center|fill"
        android:layout_weight="2" />

    <Button
        android:id="@+id/btnBack"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Back" />
</LinearLayout>
```



List-Based Selection Widgets

Example 6 - GridView (again…)

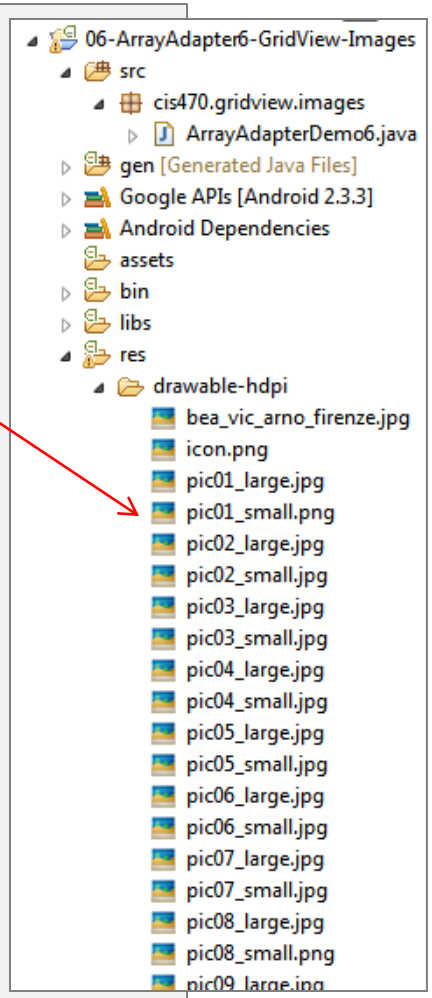
```
// REFERENCES:  Website on which you could make free thumbnails:
//              http://www.makeathumbnail.com/thumbnail.php
// ---------------------------------------------------------------------------
// displaying a number of pictures in a GridView, based on example from:
// http://developer.android.com/guide/tutorials/views/hello-gridview.html
// ---------------------------------------------------------------------------

public class ArrayAdapterDemo6 extends Activity
                               implements OnItemClickListener {
    GridView gridview;
    TextView txtSoloMsg;

    ImageView imgSoloPhoto;

    Button btnBack;
    Bundle myOriginalMemoryBundle;

// initialize array of small images (100x100 thumbnails)
    smallImages = new Integer[] { R.drawable.pic01_small,
            R.drawable.pic02_small, R.drawable.pic03_small,
            R.drawable.pic04_small, R.drawable.pic05_small,
            R.drawable.pic06_small, R.drawable.pic07_small,
            R.drawable.pic08_small, R.drawable.pic09_small,
            R.drawable.pic10_small, R.drawable.pic11_small,
            R.drawable.pic12_small, R.drawable.pic13_small,
            R.drawable.pic14_small, R.drawable.pic15_small
            };
```



List-Based Selection Widgets

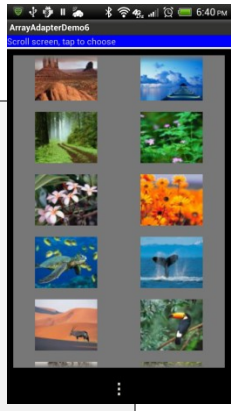
Example 6 - GridView (again…)

```
// initialize array of equivalent but better quality images
largeImages = new Integer[] { R.drawable.pic01_large,
        R.drawable.pic02_large, R.drawable.pic03_large,
        R.drawable.pic04_large, R.drawable.pic05_large,
        R.drawable.pic06_large, R.drawable.pic07_large,
        R.drawable.pic08_large, R.drawable.pic09_large,
        R.drawable.pic10_large, R.drawable.pic11_large,
        R.drawable.pic12_large, R.drawable.pic13_large,
        R.drawable.pic14_large, R.drawable.pic15_large
        };

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    myOriginalMemoryBundle = savedInstanceState;

    setContentView(R.layout.main);

    gridview = (GridView) findViewById(R.id.gridview);

    gridview.setAdapter(new MyImageAdapter(this));

    gridview.setOnItemClickListener(this);

}// onCreate
```



List-Based Selection Widgets

Example 6 - GridView (again…)




```
public class MyImageAdapter extends BaseAdapter {
      private Context mContext;

      public MyImageAdapter(Context c) { mContext = c;     }

      public int getCount() { return smallImages.length; }

      public Object getItem(int position) {  return null;  }

      public long getItemId(int position) {  return 0;     }


      // create a new ImageView for each grid item mentioned by the ArrayAdapter
      public View getView(int position, View convertView, ViewGroup parent) {
          ImageView imageView;
          if (convertView == null) {
              imageView = new ImageView(mContext);
              // new views - width 200, hwight 180, centered & cropped, 8dp padding
              imageView.setLayoutParams(new GridView.LayoutParams(200, 180))
              imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
              imageView.setPadding(8, 8, 8, 8);
          } else {
              imageView = (ImageView) convertView;
          }
          imageView.setImageResource( smallImages[position] );
          return imageView;
      }

   }// ImageAdapter
```

List-Based Selection Widgets

Example 6 - GridView (again…)

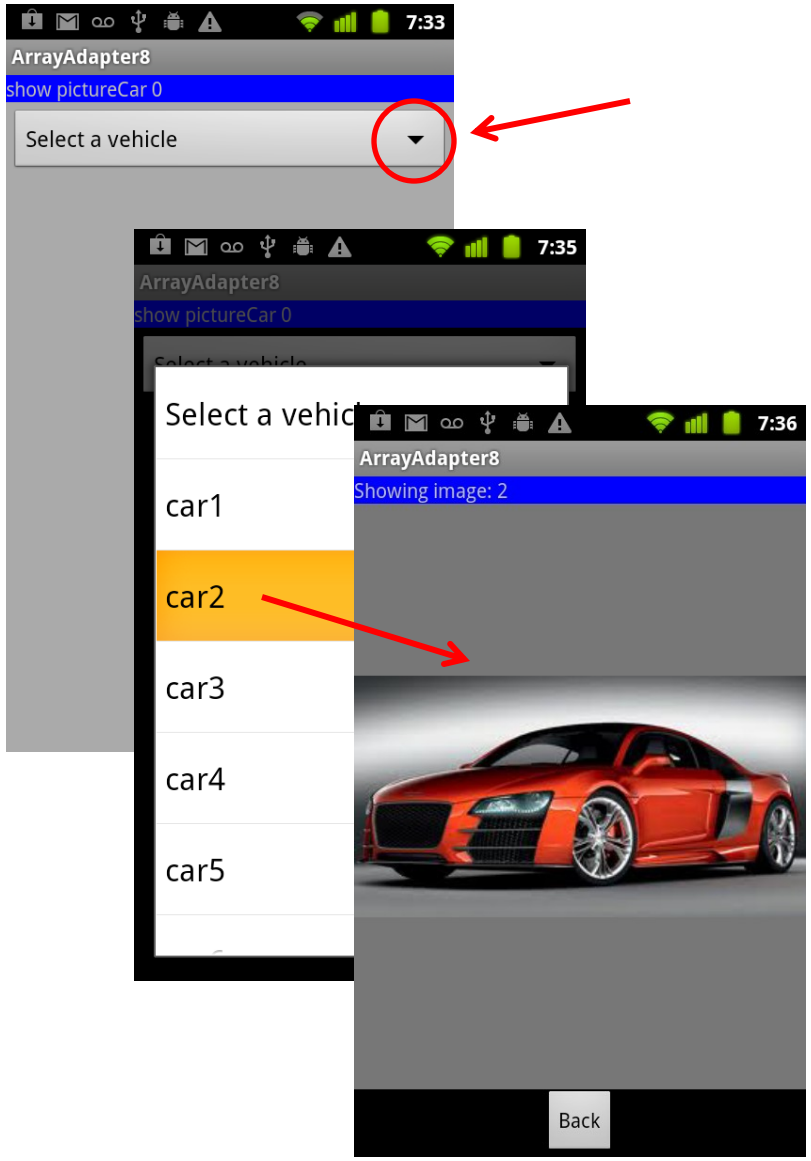
```
    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
        showBigScreen(position);
    }//onItemClick

    private void showBigScreen(int position) {
        // show the selected picture as a single frame
        setContentView(R.layout.solo_picture);
        txtSoloMsg = (TextView) findViewById(R.id.txtSoloMsg);
        imgSoloPhoto = (ImageView) findViewById(R.id.imgSoloPhoto);
        txtSoloMsg.setText("image " + position);

        imgSoloPhoto.setImageResource( largeImages[position] );

        btnBack = (Button) findViewById(R.id.btnBack);
        btnBack.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                // redraw the main screen beginning the whole app.
                onCreate(myOriginalMemoryBundle);
            }
        });

    }// showBigScreen

}// class
```

List-Based Selection Widgets

Example 7 - Spinner
(again…)

This is a simple variation of the previous example.

A list of choices is offered through a drop-down spinner control. The user taps on a row and an image for the selected choice is displayed on a new screen.



List-Based Selection Widgets

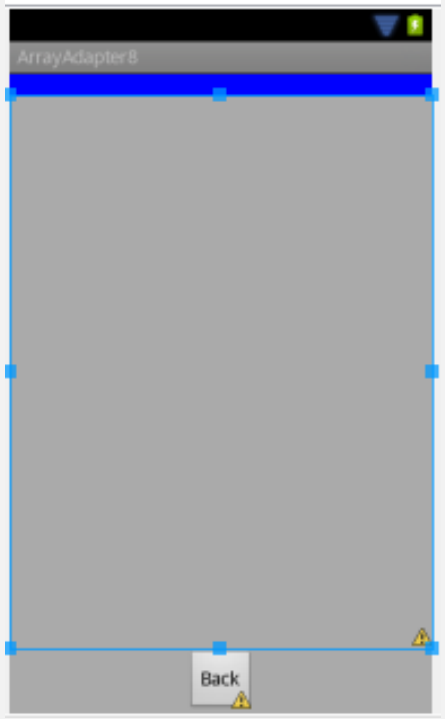
Example 7 - Spinner (again…) **screen1.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:color/darker_gray"
    >
<TextView
    android:text="Car selector"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#ff0000ff"
    />
<Spinner
          android:id="@+id/spinnerCar1"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:layout_margin="5dip"
          />
</LinearLayout>
```

List-Based Selection Widgets

Example 7 - Spinner (again…) **screen2.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:color/darker_gray"
    >
<TextView
    android:id="@+id/txtMsg2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#ff0000ff"
    />
<ImageView
            android:id="@+id/myImageView2"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_gravity="center|fill"
            android:layout_weight="1"/>

<Button
            android:text="Back"
            android:id="@+id/btnBack2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"/>
</LinearLayout>
```



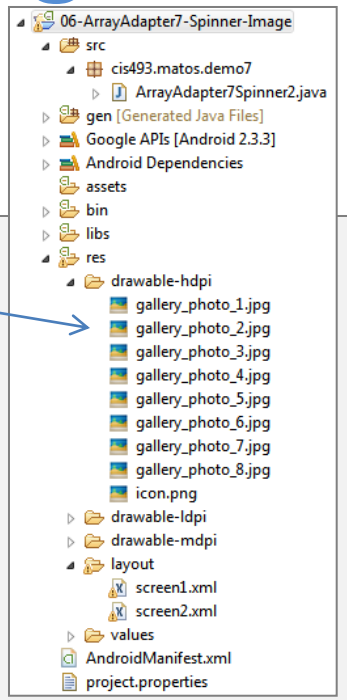
List-Based Selection Widgets

Example 7 - Spinner (again…)

```
public class ArrayAdapter7Spinner2 extends Activity
                              implements OnItemSelectedListener  {

    Spinner spinnerCar;

   String[] items = {  "Select a vehicle",
                    "car1", "car2", "car3", "car4",
                    "car5", "car6",  "car7", "car8" };

   Integer[] carImageArray;
   TextView txtMsg2;
   Button btnBack;
   Bundle myState;

   @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.screen1);

        myState = savedInstanceState;
```

List-Based Selection Widgets

Example 7 - Spinner (again...)




```
carImageArray  = new Integer[] {
          R.drawable.gallery_photo_1,   R.drawable.gallery_photo_2,
          R.drawable.gallery_photo_3,   R.drawable.gallery_photo_4,
          R.drawable.gallery_photo_5,   R.drawable.gallery_photo_6,
          R.drawable.gallery_photo_7,   R.drawable.gallery_photo_8,   };

    spinnerCar = (Spinner) findViewById(R.id.spinnerCar1);
    spinnerCar.setOnItemSelectedListener(this);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                          this,
                          android.R.layout.simple_spinner_item,
                          items);

    adapter.setDropDownViewResource(android.R.layout.simple_dropdown_item_1line);
    spinnerCar.setAdapter(adapter);
 }//onCreate

@Override
public void onItemSelected(AdapterView<?> parent, View v, int position, long id) {
   if (position>0)
      showScreen2(position);
}
@Override
public void onNothingSelected(AdapterView<?> parent) {
   // nothing TODO - stub is needed by the interface
}
```

List-Based Selection Widgets

Example 7 - Spinner (again…)

```
    public void showScreen2(int position) {

        setContentView(R.layout.screen2);

        txtMsg2 = (TextView) findViewById(R.id.txtMsg2);
        txtMsg2.setText("Showing image: " + position);

        ImageView imgCar = (ImageView)findViewById(R.id.myImageView2);

        imgCar.setImageResource(carImageArray[position-1]);

        Button btnBack = (Button) findViewById(R.id.btnBack2);
        btnBack.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                onCreate(myState);
            }
        });

    }//showScreen2

}//ArrayAdapter7Spinner2
```

List-Based Selection Widgets

Customized Lists

Android provides predefined row layouts for displaying simple lists. However, you may want more control in situations such as:

1. Not every row uses the same layout (e.g., some have one line of text, others have two)

2. You need to configure the widgets in the rows (e.g., different icons for different cases)

In those cases, the better option is to *create your own subclass of your desired Adapter.*

List-Based Selection Widgets

Based on example from:

The Busy Coder's Guide to Android Development
by Mark L. Murphy

ISBN: 978-0-9816780-0-9

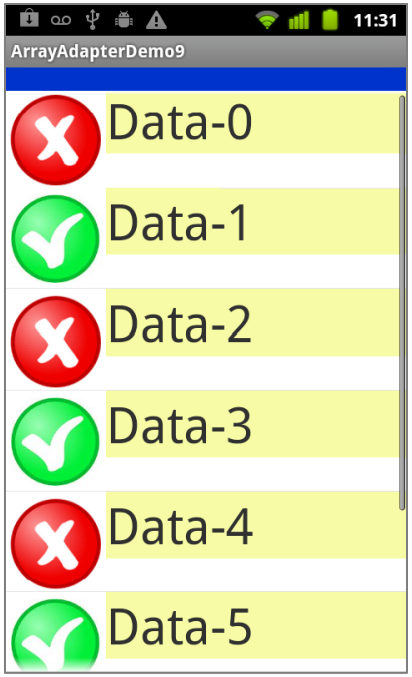
Example 8 - Customized Lists

In order to customize a Data Adapter, you need to

1. Override its **getView(),** and
2. construct (inflate) your rows yourself.

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

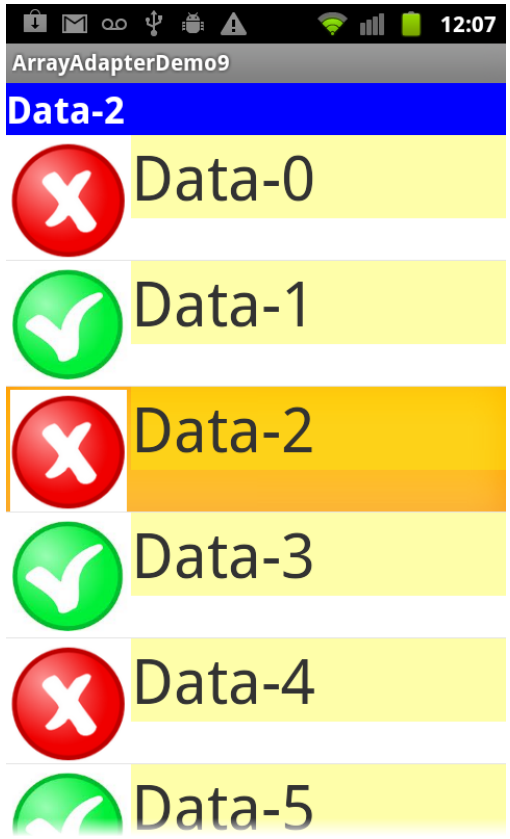
Example:
Each UI row will show an icon and text. Modify **getView()** to accommodate different icons at different row positions (odd rows show OK, even rows NOT-OK image).



List-Based Selection Widgets

Example 8 - Customized Lists main.xml

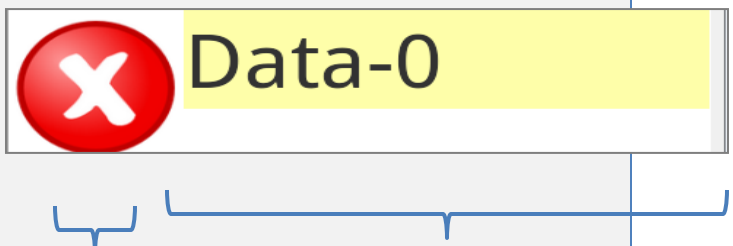
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="fill_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="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
    </ListView>

</LinearLayout>
```



List-Based Selection Widgets

Example 8 - Customized Lists 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="80dp"
        android:layout_height="80dp"
        android:paddingLeft="2dp"
        android:paddingRight="2dp"
        android:paddingTop="2dp"
        android:src="@drawable/not_ok" />

    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#55ffff00"
        android:textSize="40sp" />

</LinearLayout>
```



List-Based Selection Widgets

Example 8 - Customized Lists

```
public class ArrayAdapterDemo8 extends ListActivity {

   TextView txtMsg;

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

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      setContentView(R.layout.main);

      setListAdapter(new CustomIconLabelAdapter(this));

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

   public void onListItemClick(ListView parent, View v, int position, long id) {
      txtMsg.setText(items[position]);
   }
```

List-Based Selection Widgets

Example 8 - Customized Lists

```
class CustomIconLabelAdapter extends ArrayAdapter {
      Context context;

      CustomIconLabelAdapter(Context context) {
         super(context, R.layout.row_icon_label, items);
         this.context = context;
      }
      @Override
      public View getView(int position, View convertView, ViewGroup parent) {

         LayoutInflater inflater = ((Activity) context).getLayoutInflater();

         View row = inflater.inflate(R.layout.row_icon_label, null);

         TextView label = (TextView) row.findViewById(R.id.label);
         ImageView icon = (ImageView) row.findViewById(R.id.icon);
         label.setText(items[position]);

         if (position % 2 == 0)
            icon.setImageResource(R.drawable.notok);
         else
            icon.setImageResource(R.drawable.ok);
         return (row);
      }// getView
   }// IconicAdapter

}// ArrayAdapterDemo8
```

Calling a base ArrayAdapter

List-Based Selection Widgets

Customized Lists – LayoutInflater()

- **LayoutInflater** class converts an XML layout specification into an actual tree of View objects that are appended to the selected UI view.

- A basic *ArrayAdapter* requires three arguments: current **context**, **layout** to show the output rows, source data **items** (data to place in 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

List-Based Selection Widgets

Example 9
HorizontalScrollView

If you are developing apps using Release 4.x (or newer) you may want to follow the strategy suggested here as a way of replacing the now deprecated GalleryView control

main.xml

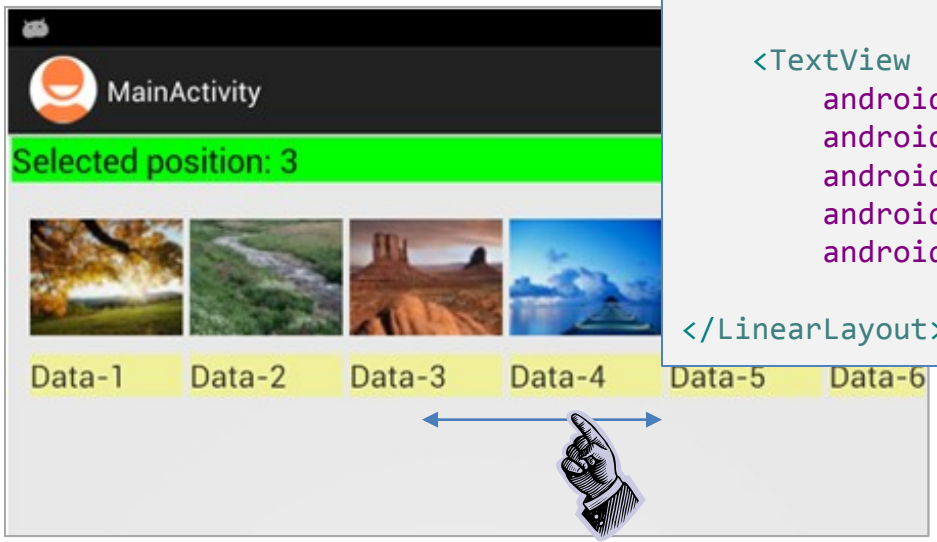
Inflated frames go here…

```
<?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="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="fill_parent"
        android:layout_height="wrap_content" >

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

    </HorizontalScrollView>

</LinearLayout>
```

List-Based Selection Widgets

Example 9 HorizontalScrollView

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.

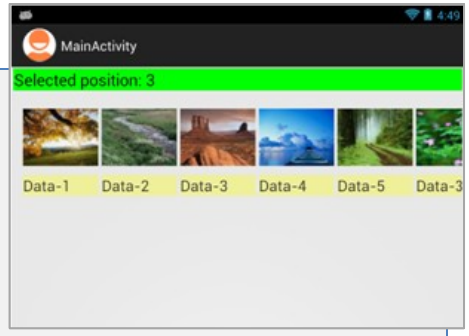


frame_icon_caption.xml

```
<?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="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>
```

List-Based Selection Widgets

Example 9 - HorizontalScrollView

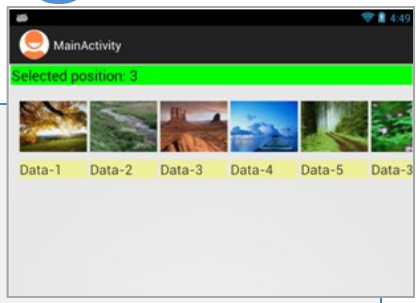



```
public class MainActivity extends Activity {

    TextView txtMsg;
    ViewGroup scrollViewgroup;
    ImageView icon;
    TextView caption;

    String[] items = { "Data-1", "Data-2", "Data-3", "Data-4", "Data-5",
           "Data-3", "Data-7", "Data-8", "Data-9", "Data-10", "Data-11",
           "Data-12", "Data-13", "Data-14", "Data-15" };

    Integer[] thumbnails = { R.drawable.pic01_small, R.drawable.pic02_small,
           R.drawable.pic03_small, R.drawable.pic04_small,
           R.drawable.pic05_small, R.drawable.pic06_small,
           R.drawable.pic07_small, R.drawable.pic08_small,
           R.drawable.pic09_small, R.drawable.pic10_small,
           R.drawable.pic11_small, R.drawable.pic12_small,
           R.drawable.pic13_small, R.drawable.pic14_small,
           R.drawable.pic15_small };

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
```

A set of images (stored in the res/drawable) is used to feed data to the horizontal scroller

List-Based Selection Widgets

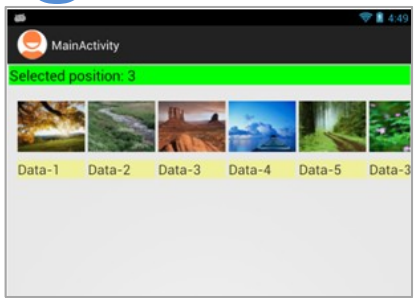
Example 9 - HorizontalScrollView




```
        // this layout is contained inside the horizontalScrollerView
        scrollViewgroup = (ViewGroup) findViewById(R.id.viewgroup);

1 →     for (int i = 0; i < items.length; i++) {

2 →         final View frame = getLayoutInflater().inflate( R.layout.frame_icon_label, null );

            TextView caption = (TextView) frame.findViewById(R.id.caption);
            ImageView icon = (ImageView) frame.findViewById(R.id.icon);

            icon.setImageResource( thumbnails[i] );
            caption.setText( items[i] );

3 →         scrollViewgroup.addView( frame );

            frame.setId(i);
4 →         frame.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    String text = "Selected position: " + frame.getId();
                    txtMsg.setText(text);
                }
            });// listener
        }// for
    }// onCreate
}// class
```

List-Based Selection Widgets

Example 9 - HorizontalScrollView



Comments

1. A collection of images and their associated text is used as data source.

2. The *frame_icon_caption layout* describing how an image and its caption should be displayed is inflated and stored in the *frame* view.

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

4. Each *frame* is given an ID (its current position in the scrollViewgroup). Finally a CLICK listener is attached to each frame.

List-Based Selection Widgets

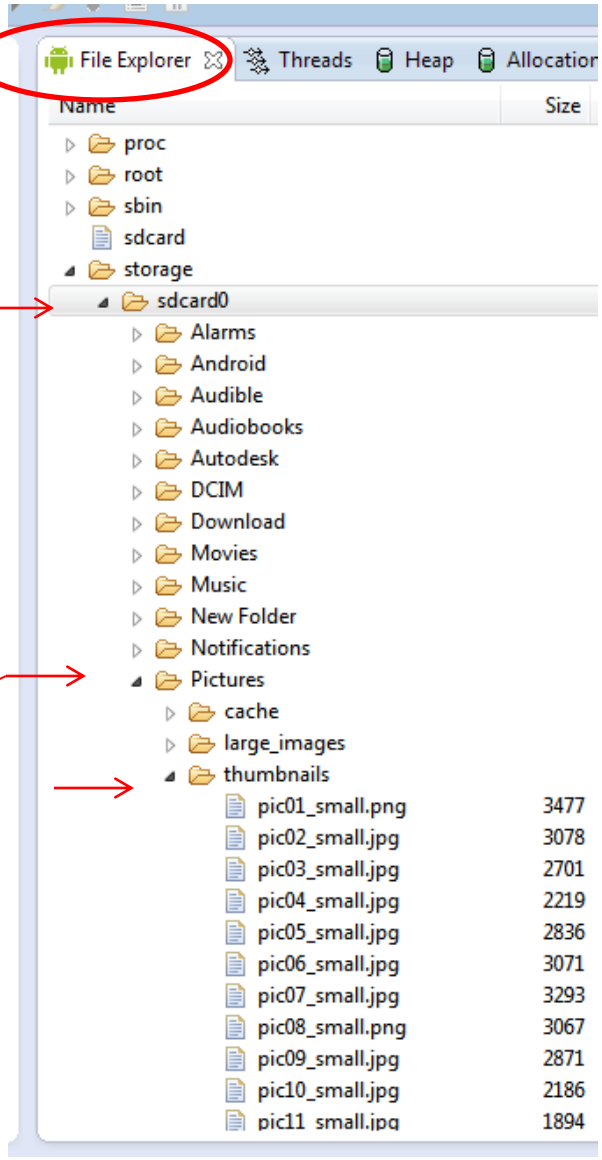
Example 10 - HorizontalScrollView

A Variation on the previous Example.

The previous example stored a set of pictures in memory.

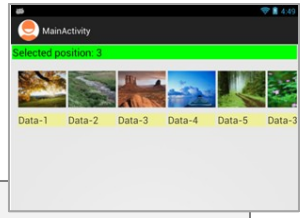
This time we will use the external disk (**SD-card**) to keep the data.

More precisely, the user has already transferred a copy of the pictures to the SD folder **/Pictures/thumbnails/**



List-Based Selection Widgets

Example 10 – HorizontalScrollView 1 of 3

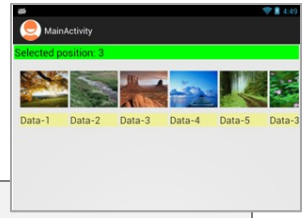


We use in this app the same layouts introduced in Example-9

```
public class MainActivity extends Activity {

    ViewGroup scrollViewgroup;
    ImageView icon;
    TextView caption;
    TextView txtMsg;
    int index;

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

        // this layout is contained inside the horizontalScrollerView
        scrollViewgroup = (ViewGroup) findViewById(R.id.scrollViewgroup);

        try {

            String absolutePath2SdCard = Environment.getExternalStorageDirectory()
                                                    .getAbsolutePath();

            String path2PicturesOnSdCard = absolutePath2SdCard + "/Pictures/thumbnails/";

            File sdPictureFiles = new File(path2PicturesOnSdCard);
            File[] files = sdPictureFiles.listFiles();
```

List-Based Selection Widgets

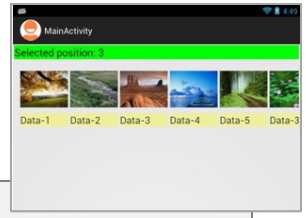
Example 10 – HorizontalScrollView 2 of 3




```
File sdPictureFiles = new File(path2PicturesOnSdCard);
File[] files = sdPictureFiles.listFiles();
txtMsg.append("\nNum files: " + files.length);

File file;

for (index = 0; index < files.length; index++) {
   file = files[index];

   final View frame = getLayoutInflater().inflate(
         R.layout.frame_icon_label, null);

   TextView caption = (TextView) frame.findViewById(R.id.caption);
   ImageView icon = (ImageView) frame.findViewById(R.id.icon);

   // convert (jpg, png,...) file into an acceptable bitmap
   BitmapFactory.Options bmOptions = new BitmapFactory.Options();
   bmOptions.inSampleSize = 1; // one-to-one scale
   Bitmap bitmap = BitmapFactory.decodeFile(
                  file.getAbsolutePath(),
                  bmOptions);

   icon.setImageBitmap(bitmap);
   caption.setText("File-" + index);
```

List-Based Selection Widgets

Example 10 – HorizontalScrollView 3 of 3




```
            scrollViewgroup.addView(frame);

            frame.setId(index);
            frame.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    String text = "Selected position: " + frame.getId();
                    txtMsg.setText(text);
                }
            }); // listener

        }// for

    } catch (Exception e) {

        txtMsg.append("\nError: " + e.getMessage());
    }

  } // onCreate
} // class
```

List-Based Selection Widgets

Example 10 – HorizontalScrollView

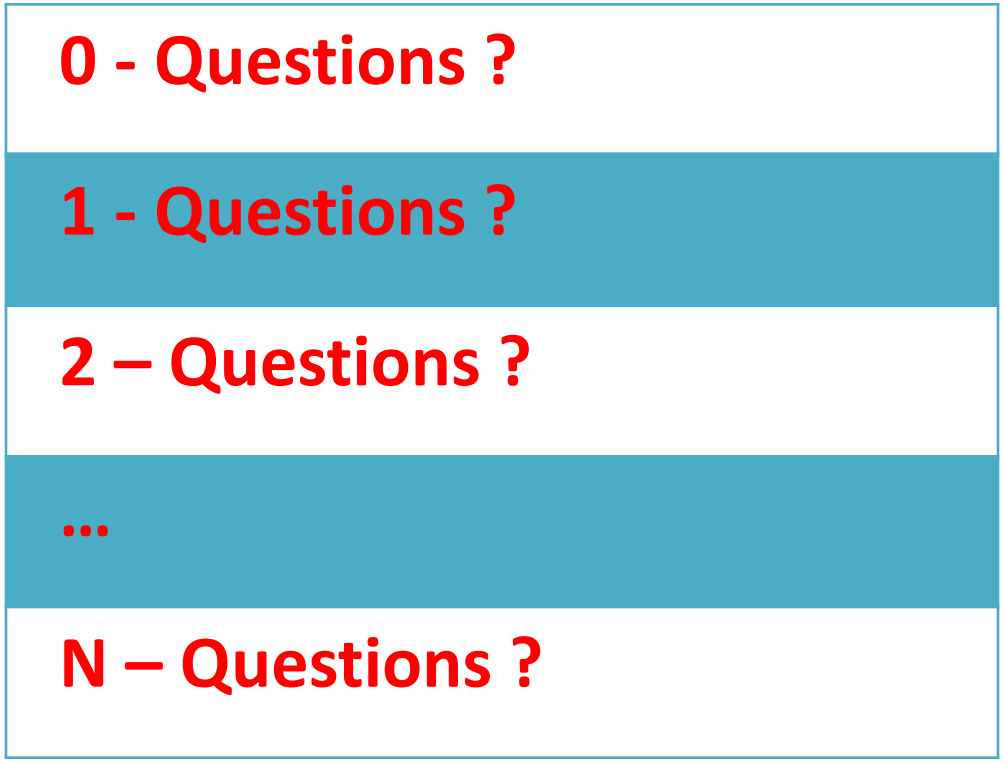
BitmapFactory
Creates Bitmap objects from various sources, including files, streams, and byte-arrays.

inSampleSize Option

- If set to a value > 1, requests the decoder to subsample the original image, returning a smaller image to save memory.
- The sample size is the number of pixels in either dimension that correspond to a single pixel in the decoded bitmap. For example, inSampleSize == 4 returns an image that is 1/4 the width/height of the original, and 1/16 the number of pixels.
- Any value <= 1 is treated the same as 1.

Reference:
http://developer.android.com/reference/android/graphics/BitmapFactory.Options.html

List-Based Selection Widgets



Appendix A: List-Based Selection Widgets

Android's Predefined Layouts

This is the definition of: ***simple_list_item_1 .*** It consists of a single TextView field named "**text1**" centered, large font, and some padding.


```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:minHeight="?android:attr/listPreferredItemHeight"
    android:paddingLeft="6dip"
    android:textAppearance="?android:attr/textAppearanceLarge" />
```

Appendix A: List-Based Selection Widgets

Android's Predefined Layouts

This is the definition of: ***simple_spinner_dropdown_item*** in which a single row holding a radio-button and text is shown.

Link: http://code.google.com/p/pdn-slatedroid/source/browse/trunk/eclair/frameworks/base/core/res/res/layout/simple_spinner_dropdown_item.xml?r=44

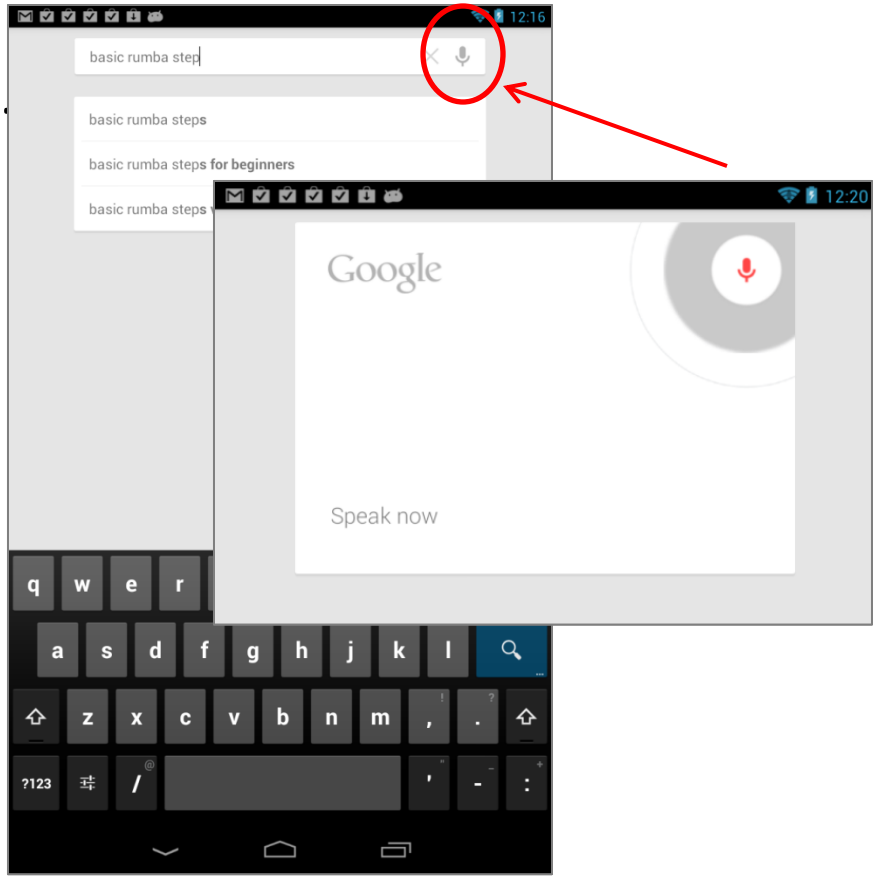
```
<?xml version="1.0" encoding="utf-8"?>
<!--
** Copyright 2008, The Android Open Source Project
** etc...
-->
<CheckedTextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    style="?android:attr/spinnerDropDownItemStyle"
    android:singleLine="true"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:ellipsize="marquee" />
```

Appendix B: EditText Boxes & Keyboarding

Keyboarding data into Android's applications is functionally dependent of the hardware present in the actual device.

Sliding Window in this unit exposes a hard keyboard.

Device has a permanently exposed hard keyboard and Stylus pen appropriate for handwriting



Input accepted from Virtual keyboard and/or voice recognition

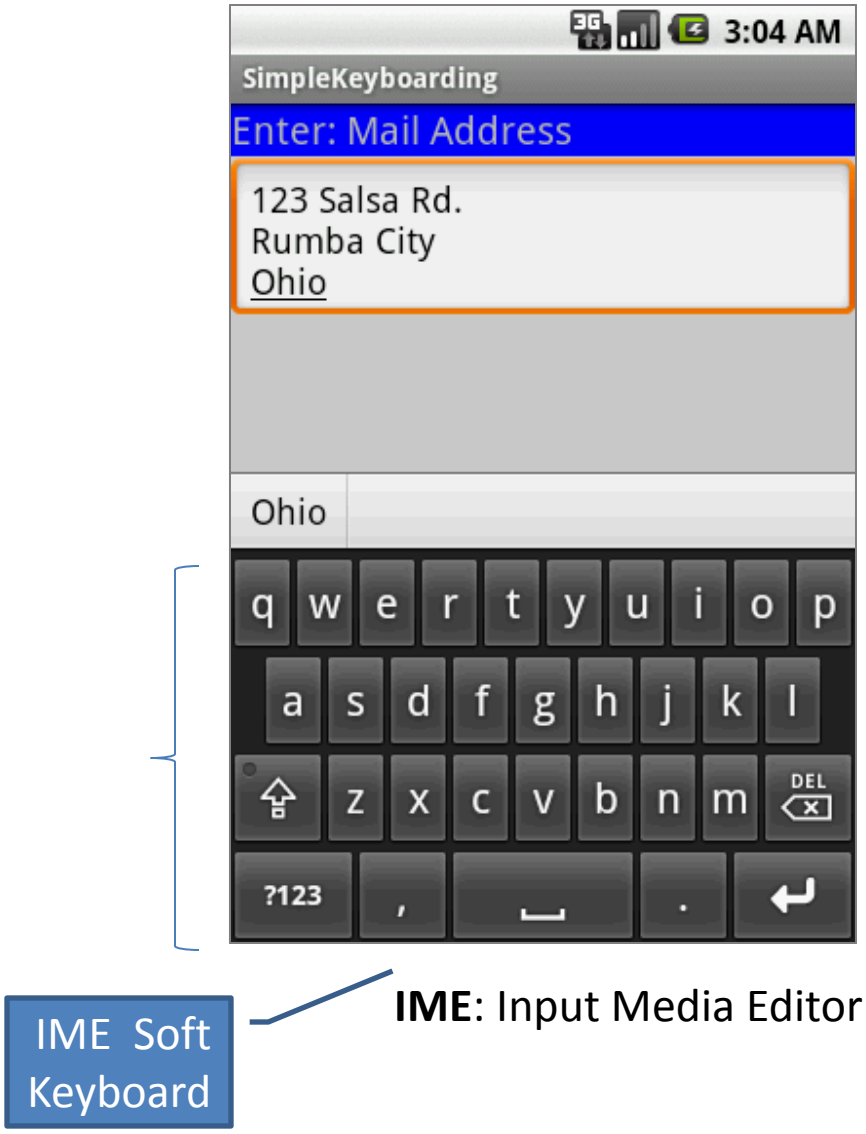
Appendix B: EditText Boxes & Keyboarding

When the user taps on an **EditText** box, the Input Media Framework (**IMF**) provides access to

1. a hard (or *real)* keyboard (if one is present) or

2. a soft (or virtual) keyboard known as IME that is the most appropriated for the current input type.

You may close the virtual keyboard by tapping the hardware **BackArrow** key.



Appendix B: EditText Boxes & Keyboarding

Telling Android what data to expect

TextViews can use either **XML** elements or **Java** code to tell the type of textual data they should accept. For example:

XML

```
android:inputType="phone"
```

Java

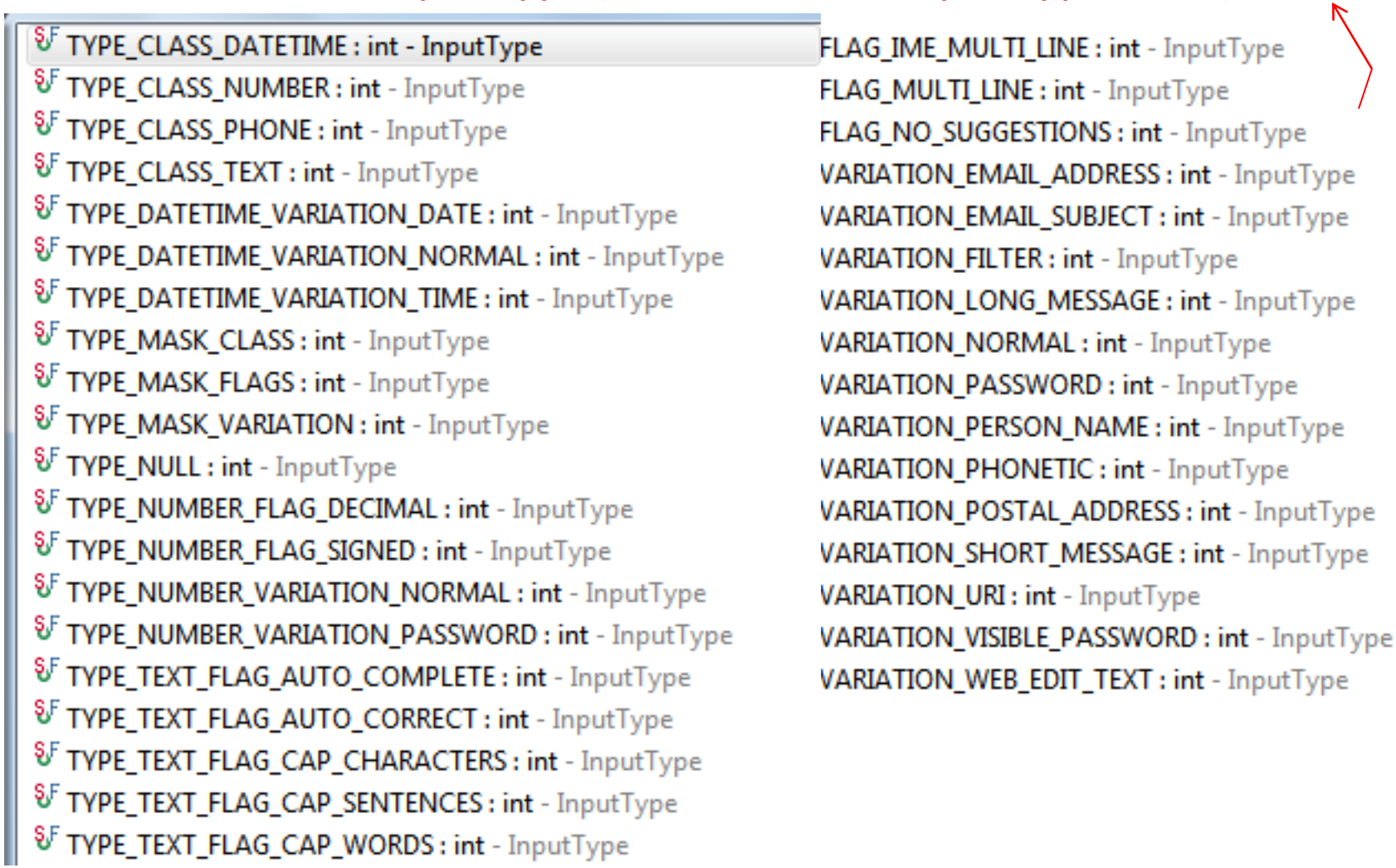
```
editTextBox.setInputType(
        android.text.InputType.TYPE_CLASS_PHONE);
```

Knowing the inputType has an impact on virtual keyboards (the software can expose the *best* layout for the current input class)

Appendix B: EditText Boxes & Keyboarding

Java Usage – inpuType Classes

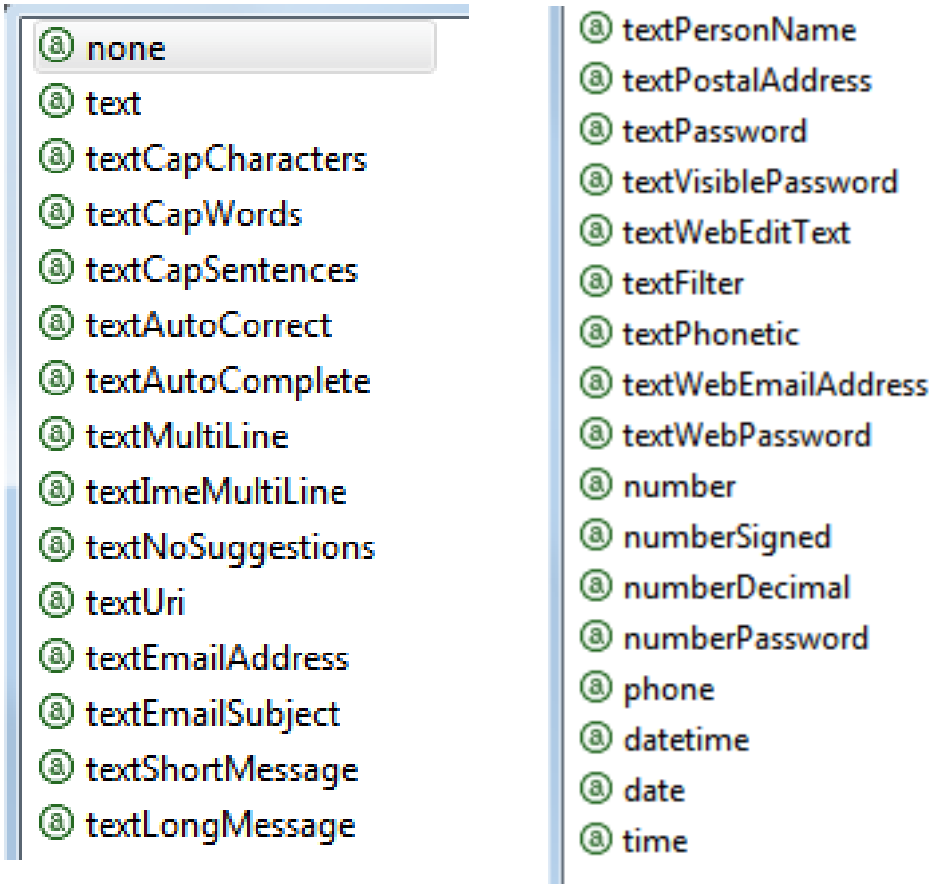
editTextBox.setInputType(android.text.InputType.XXX);



Appendix B: EditText Boxes & Keyboarding

XML Usage – inputType Classes

```
<EditText
        …
        android:inputType="numberSigned|numberDecimal"
        … />
```



Reference:
http://developer.android.com/reference/android/R.styleable.html#TextView_inputType

Appendix B: EditText Boxes & Keyboarding

Example11: Using multiple XML attributes

android:inputType="text|textCapWords"

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

<TextView
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:background="#ff0000ff"
     android:text="inputType: text|textCapWords"
     android:textStyle="bold"
     android:textSize="22sp" />

<EditText
     android:id="@+id/editTextBox"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:padding="5dip"
     android:textSize="18sp"
     android:inputType="text|textCapWords"   />
</LinearLayout>
```

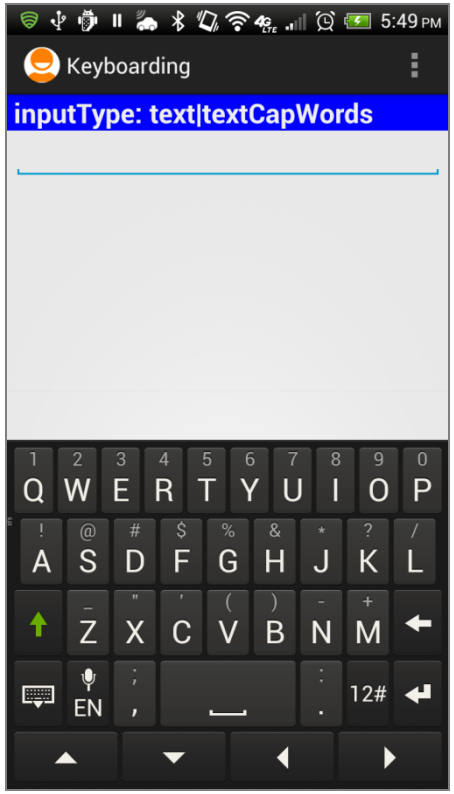
Use "pipe" symbol **|** to separate the options.

In this example a soft text keyboard will be used.

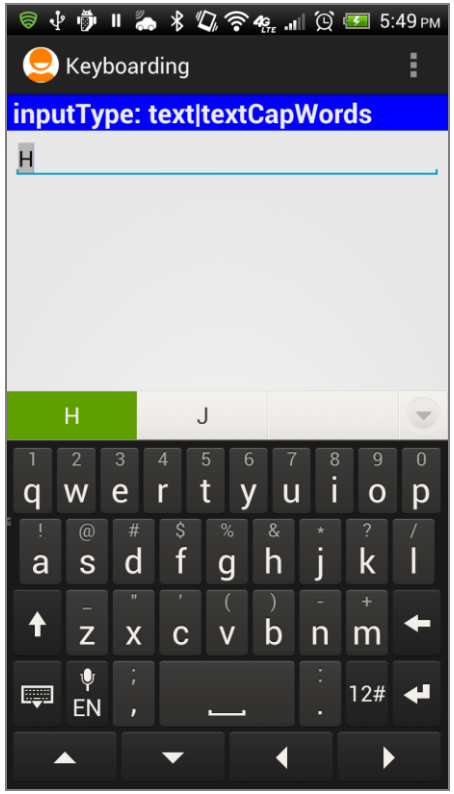
Each word will be capitalized.

Appendix B: EditText Boxes & Keyboarding

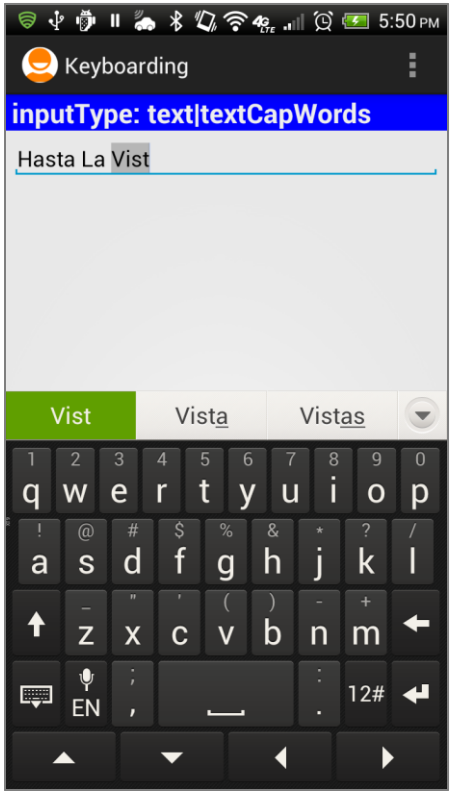
Example11: Using android:inputType= *" text|textCapWords"*



After tapping the EditText box to gain focus, a soft keyboard appears showing CAPITAL letters



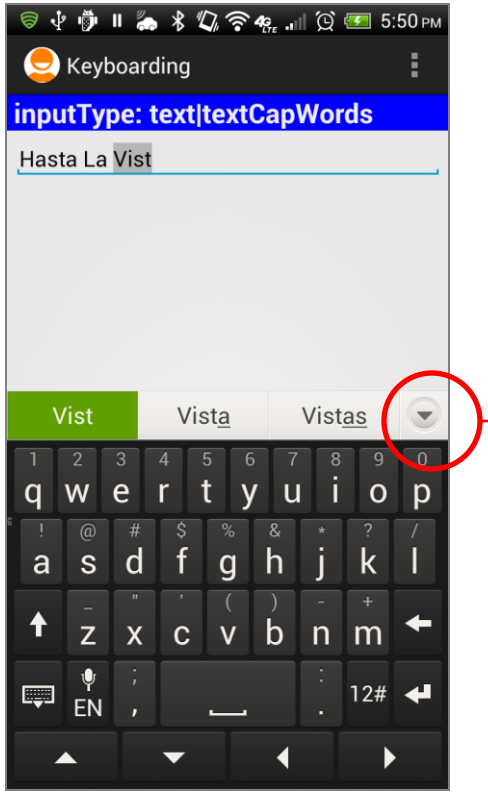
After first letter is typed the keyboard automatically switches to LOWER case mode



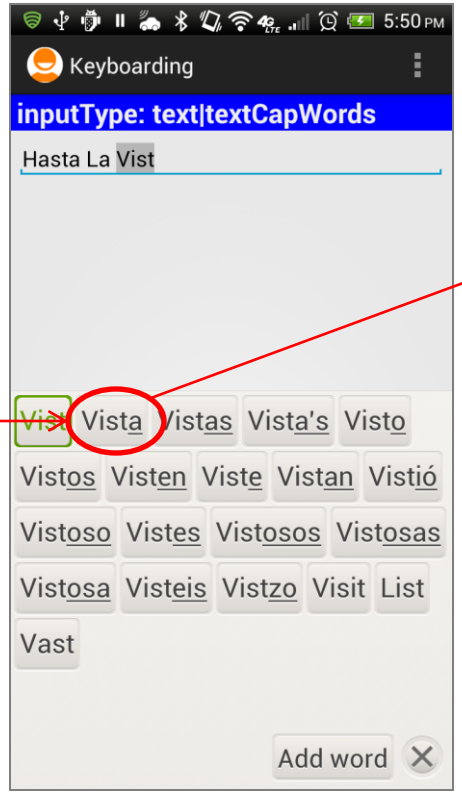
After entering *space* the keyboard repeats cycle beginning with UPPER case, then LOWER case letters.

Appendix B: EditText Boxes & Keyboarding

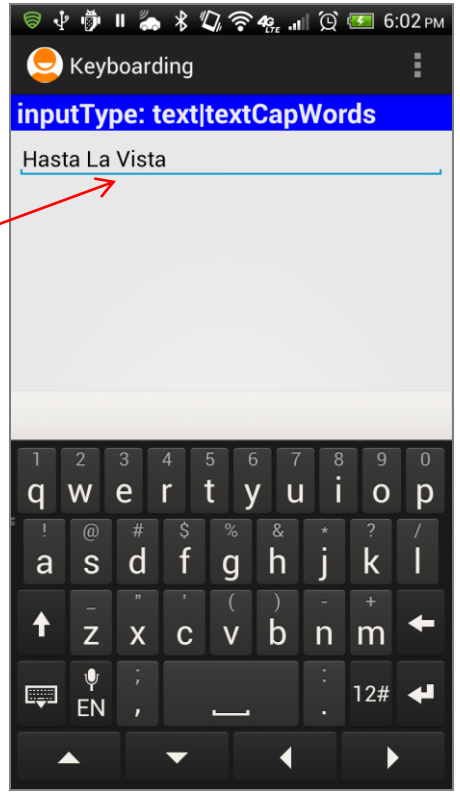
Example11: Using android:inputType= *" text|textCapWords"*



English and Spanish are the user's selected languages in this device



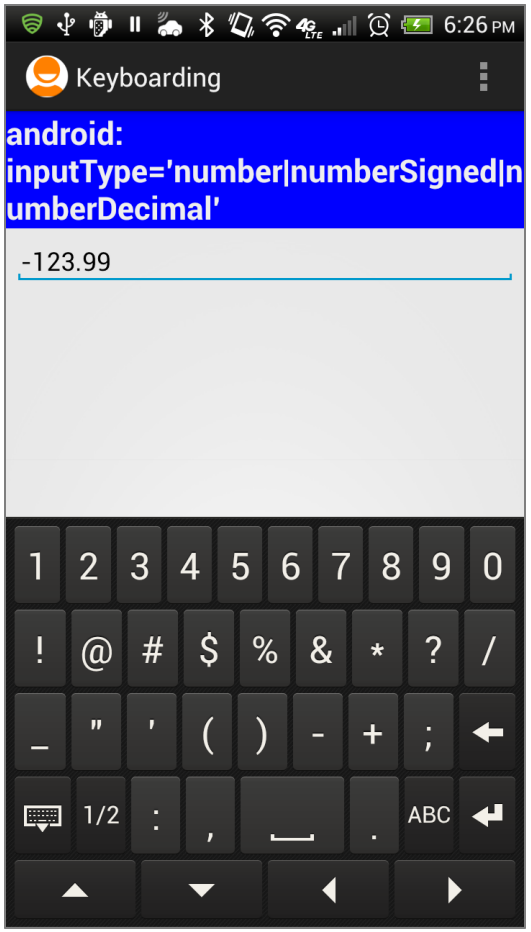
You may speed up typing by tapping on an option from the list of suggested words (bilingual choices)



Selected word is introduced in the EditText box

Appendix B: EditText Boxes & Keyboarding

Example 12: Using
android:inputType="*number|numberSigned|numberDecimal*"



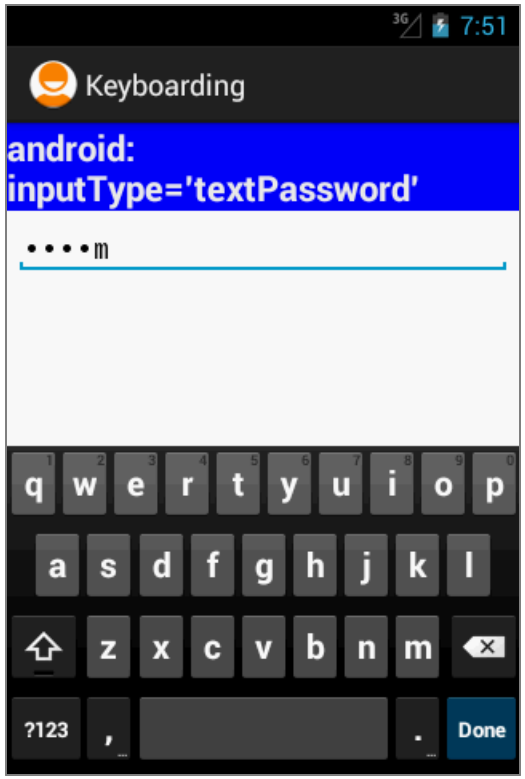
1. The keyboard displays numbers.
2. *Non-numeric* keys (such as !@#$%&*?/_) are visible but disable.
3. Only valid numeric expressions can be entered.
4. Type **number|numberSigned** accepts integers.
5. Type **numberDecimal** accepts real numbers.

Assume the EditText field is named: **editTextBox**, In Java code we could at run-time set the input method by issuing the command:

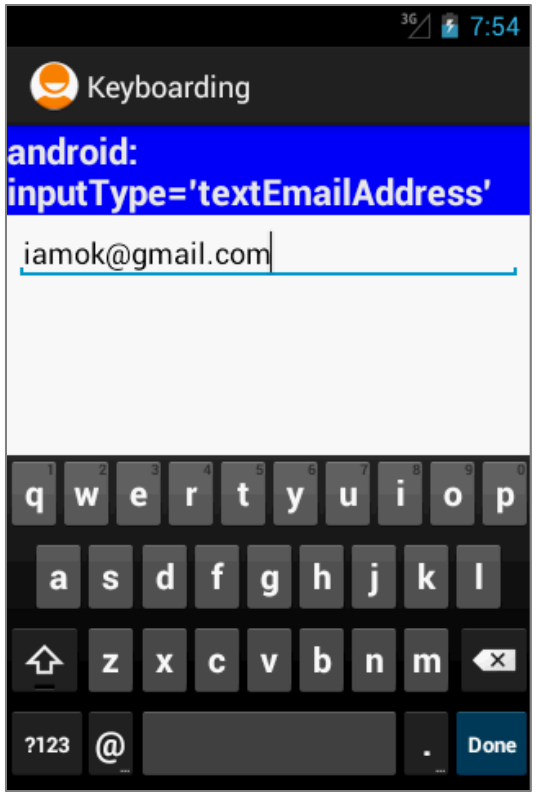
```
editTextBox.setInputType(
  android.text.InputType.TYPE_CLASS_NUMBER |
  android.text.InputType.TYPE_NUMBER_FLAG_SIGNED);
```

Appendix B: EditText Boxes & Keyboarding

Example 13: Using android:inputType="*textPassword*"



Example 14: Using android:inputType="*textEmailAddress*"



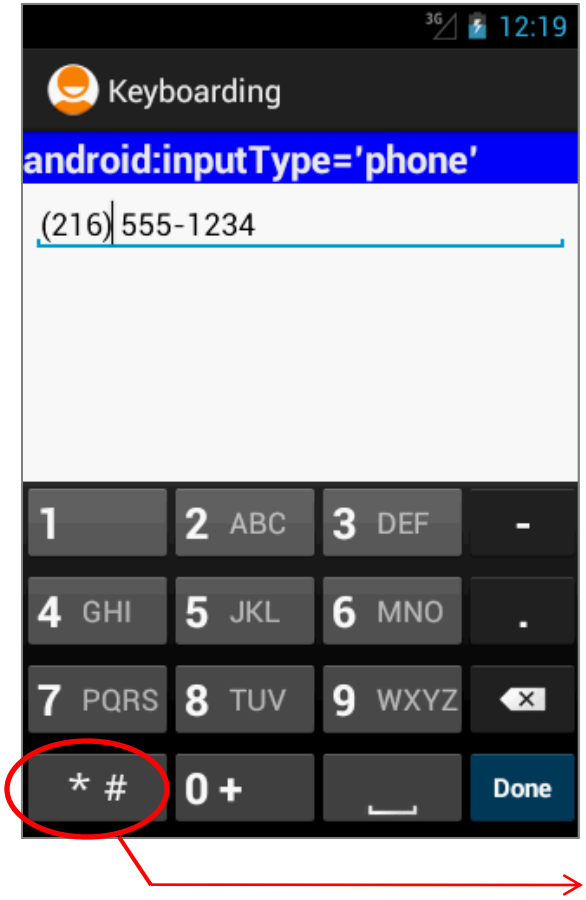
Soft keyboard shows characters used in email addresses (such as letters, **@,** dot**).**

Click on [**?123**] key (lower-left) for additional characters

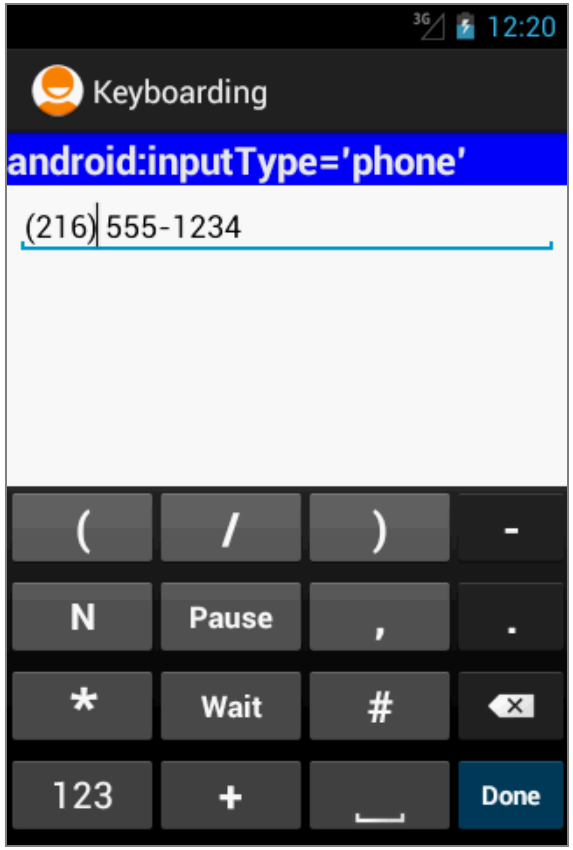
- The keyboard displays all possible keys.
- Current character is briefly displayed for verification purposes.
- The current character is hidden and a *heavy-dot* is displayed.

Appendix B: EditText Boxes & Keyboarding

Example 15: Using android:inputType="*phone*"



Additional symbols



Soft keyboard displays the layout of a typical *phone keypad* plus additional non digit symbols such as:

( ) . / Pause Wait # - +

Appendix B: EditText Boxes & Keyboarding

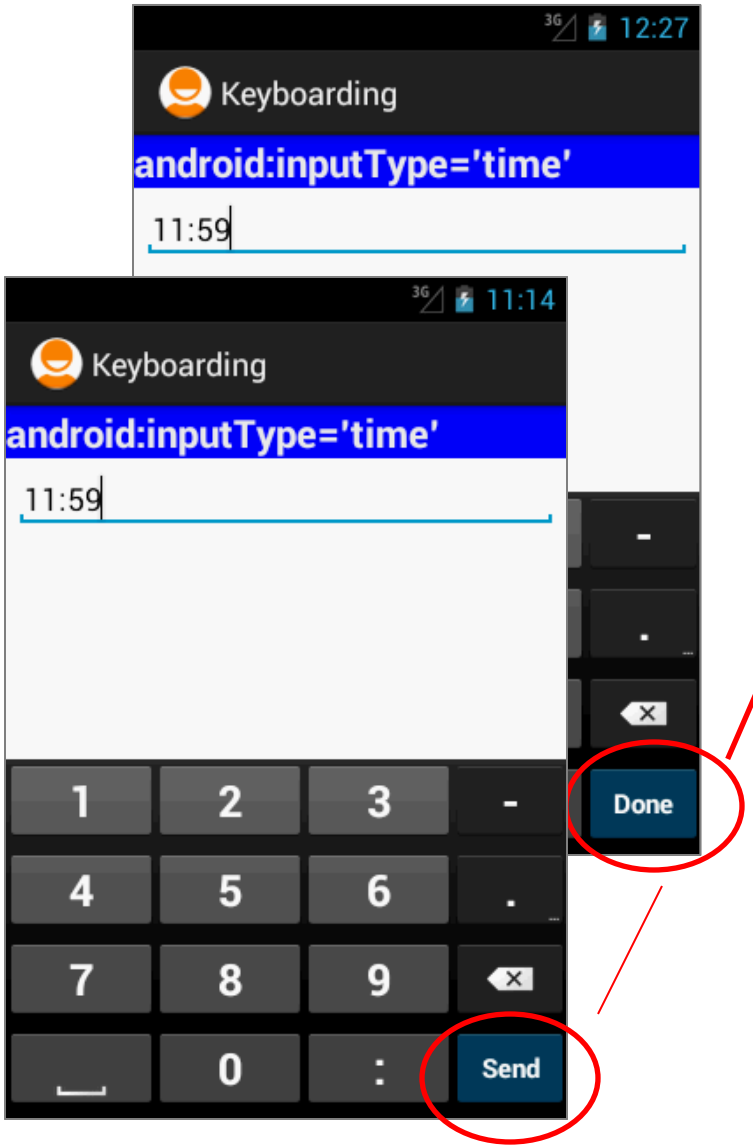
Example 16: Using android:inputType=*"time"*

Soft keyboard displays a numerical layout.

Only digits and colon-char "**:**" can be used.

Appendix B: EditText Boxes & Keyboarding

Example 16: Using android:inputType="*time*"



When clicked, the **Auxiliary** button **DONE** will removed the keyboard from the screen (default behavior).

You may change the button's caption and set a listener to catch the click event on the Auxiliary button. To do that add the following entry to your XML specification of the EditText box:

```
android:imeAction="actionSend"
```

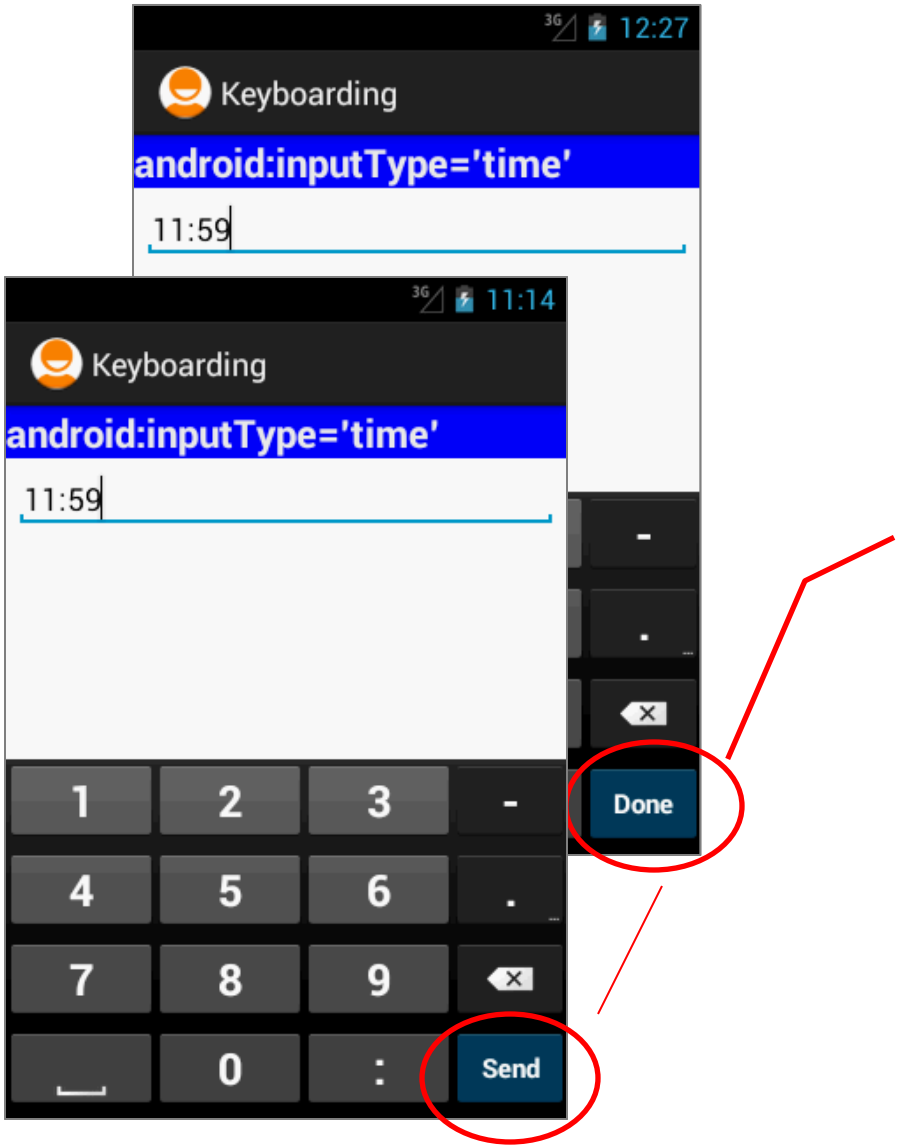
Later, your Java code could provide an implementation of the method:

```
editTextBox
   .setOnEditorActionListener()
```

to do something with the event.

Appendix B: EditText Boxes & Keyboarding

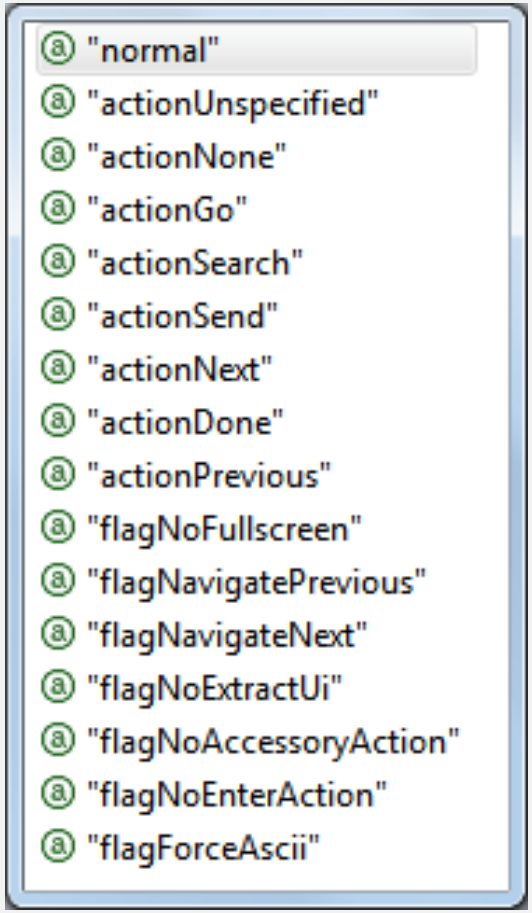
Example 16: Using android:inputType=*"time"*



Other options for

`android:imeAction=". . ."`

are



Appendix B: EditText Boxes & Keyboarding

Example 17: Using android:inputType="*datetime*"

Soft keyboard displays a numerical layout.

Only digits and date/time valid characters are allowed.

Examples of valid dates are:

```
12/21/2012  12:12
12/31/2011
12:30
```

Appendix B: EditText Boxes & Keyboarding

Disable Soft Keyboarding on an EditText View 

Assume *editTextBox1* is an EditText box. To **disable** the action of the soft keyboard on an EditText you should set its input type to null, as indicated below:

```
editTextBox.setInputType( InputType.TYPE_NULL );
```

Appendix B: EditText Boxes & Keyboarding

TextWatcher Control

Assume *txtBox1* is an **Editable** box. A listener of the type **onKeyListener** could be used to follow the actions made by the hardware keyboard; however *it will not properly work with the **Virtual Keyboard***.

A solution to this problem is to attach to the Editable control a **TextWatcher** and let its methods be called when the Editable text is changed.

The main methods of a **TextWatcher** are:

```
public void afterTextChanged (Editable theWatchedText)
public void beforeTextChanged ( … )
public void onTextChanged ( … )
```

Appendix B: EditText Boxes & Keyboarding

Example 18: TextWatcher Demo

EditText box uses **addTextChangedListener**

IME suggestions

Appendix B: EditText Boxes & Keyboarding

Example 18: TextWatcher Demo

```
<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="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:textColor="#ffffffff"
        android:background="#ff0000ff"
        tools:context=".MainActivity" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="text"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="41dp“  />

</RelativeLayout>
```

Appendix B: EditText Boxes & Keyboarding

Example 18: TextWatcher Demo

```
public class MainActivity extends Activity {
    EditText editText1;
    TextView txtMsg;
    int keyCount = 0;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);

        editText1 = (EditText) findViewById(R.id.editText1);

        editText1.addTextChangedListener(new TextWatcher() {
            public void afterTextChanged(Editable theWatchedText) {
                String msg = "count: “ + editText1.getText().toString().length() + " "
                        + theWatchedText.toString();
                txtMsg.setText(msg);
            }

            public void beforeTextChanged(CharSequence arg0, int arg1,
                    int arg2, int arg3) {
            }
            public void onTextChanged(CharSequence arg0, int arg1, int arg2,
                    int arg3) {
            }
        }); // addTextChangedListener
    } // onCreate
}//class
```