Mobile Development
Lecture 4: Android GUI

Mahmoud El-Gayyar
elgayyar@ci.suez.edu.eg
Elgayyar.weebly.com

String Resources

- Declare constant strings and arrays in `res/values/strings.xml`:

```
<resources>
    <string name="name">value</string>
    <string name="name">value</string>

    <string-array name="arrayname">
        <item>value</item>
        <item>value</item>
        <item>value</item>
        ...
        <item>value</item>
    </string-array>
</resources>
```

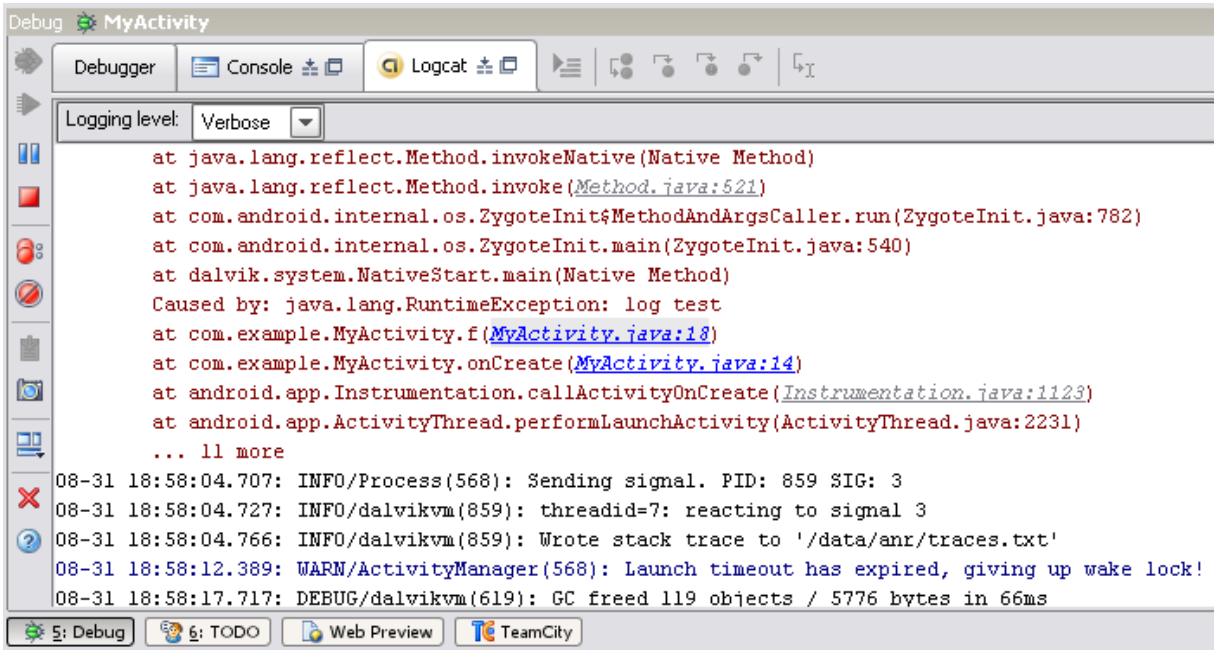
- Refer to them in Java code:
  - as a resource: `R.string.name`, `R.array.name`
  - as a string or array: `getResources().getString(R.string.name)`, `getResources().getStringArray(R.array.name)`

LogCat

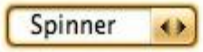
Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the Log class.

```
Log.i("MyActivity", "MyClass.getView() – get item number " + position);
```

```
Log.d(string, string); //debug
Log.e(string, string); //error
Log.w(string, string); //warning
```



GUI Widgets

Analog/DigitalClock 	Button 	Checkbox 	Date/TimePicker 
EditText 	Gallery 	ImageView/Button 	ProgressBar 
RadioButton 	Spinner 	TextView 	MapView,WebView 

TextView

Displays text to the user

key attributes:

`Layout:margins=[]`	Margins from the container layout (dp)
`android:id="@+id/`***`theID`***`"`	Unique ID for use in Java code
`android:textColor=`***`“color”`***	Color of the text
`android:Background=`***`“color”`***	Color of text background
`android:minWidth="`***`int`***`"`	Min width of the box
`android:text="`***`text`***`"`	Initial text to put in box(default empty)
`android:textSize="`***`size`***`"`	Size of font to use(e.g."20sp")
`android:padding="`***`size`***`"`	Space within object

Dp: Density independent Pixel (160 dp per inch)
Sp: Scale Independent Pixel

Button

A clickable widget with a text label

key attributes:

`android:clickable="`***`bool`***`"`	Set to false to disable the button
`android:id="@+id/`***`theID`***`"`	Unique ID for use in Java code
`android:onClick="`***`function`***`"`	Function to call in activity when clicked (must be public, void, and take a View arg)
`android:text="`***`text`***`"`	Text top of the button

- represented by `Button` class in Java code

```
Button b = (Button) findViewById(R.id.theID);
...
```

ImageButton

A clickable widget with an image label

key attributes:

`android:clickable="`***`bool`***`"`	Set to false to disable the button
`android:id="@+id/`***`theID`***`"`	Unique ID for use in Java code
`android:onClick="`***`function`***`"`	Function to call in activity when clicked (must be public, void, and take a View arg)
`android:src="@drawable/`***`img`***`"`	Image to put on the button (must correspond to an image resource)

- to set up an image resource:
  - put image file in project folder **app/src/main/res/drawable**
  - use `@drawable/foo` to refer to `foo.png`
    - use simple file names with only letters and numbers
  - Or (…) → Project → Drawable and select the image from properties window

ImageView

Displays an image

key attributes:

`android:id="@+id/`***`theID`***`"`	Unique ID for use in Java code
`android:onClick="`***`function`***`"`	Function to call in activity when clicked (must be public, void, and take a View arg)
`android:src="@drawable/`***`img`***`"`	Image source (must correspond to an image resource)

- to change the visible image, in Java code:
  - get the ImageView using findViewById
  - call its **setImageResource** method and pass R.drawable.***filename***

EditText

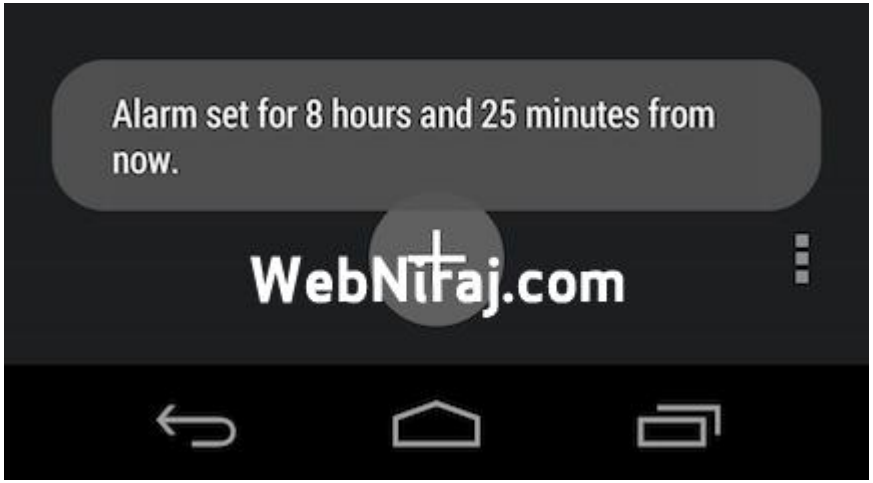
An editable text input box

key attributes:

`android:hint="`***`text`***`"`	Hint Text (in Gray to show what to type)
`android:id="@+id/`***`theID`***`"`	Unique ID for use in Java code
`android:inputType="`***`type`***`"`	Kind of input `number,phone,date,time,...`
`android:lines="`***`int`***`"`	Number of visible lines (rows) of input
`android:maxLines="`***`int`***`"`	Max lines to allow user to type in the box
`android:text="`***`text`***`"`	Initial text to put in box(default empty)
`android:textSize="`***`size`***`"`	Size of font to use(e.g."20sp")
`Layout:width="`**`string`**`"`	Wrap content – fill parent

Toast Message

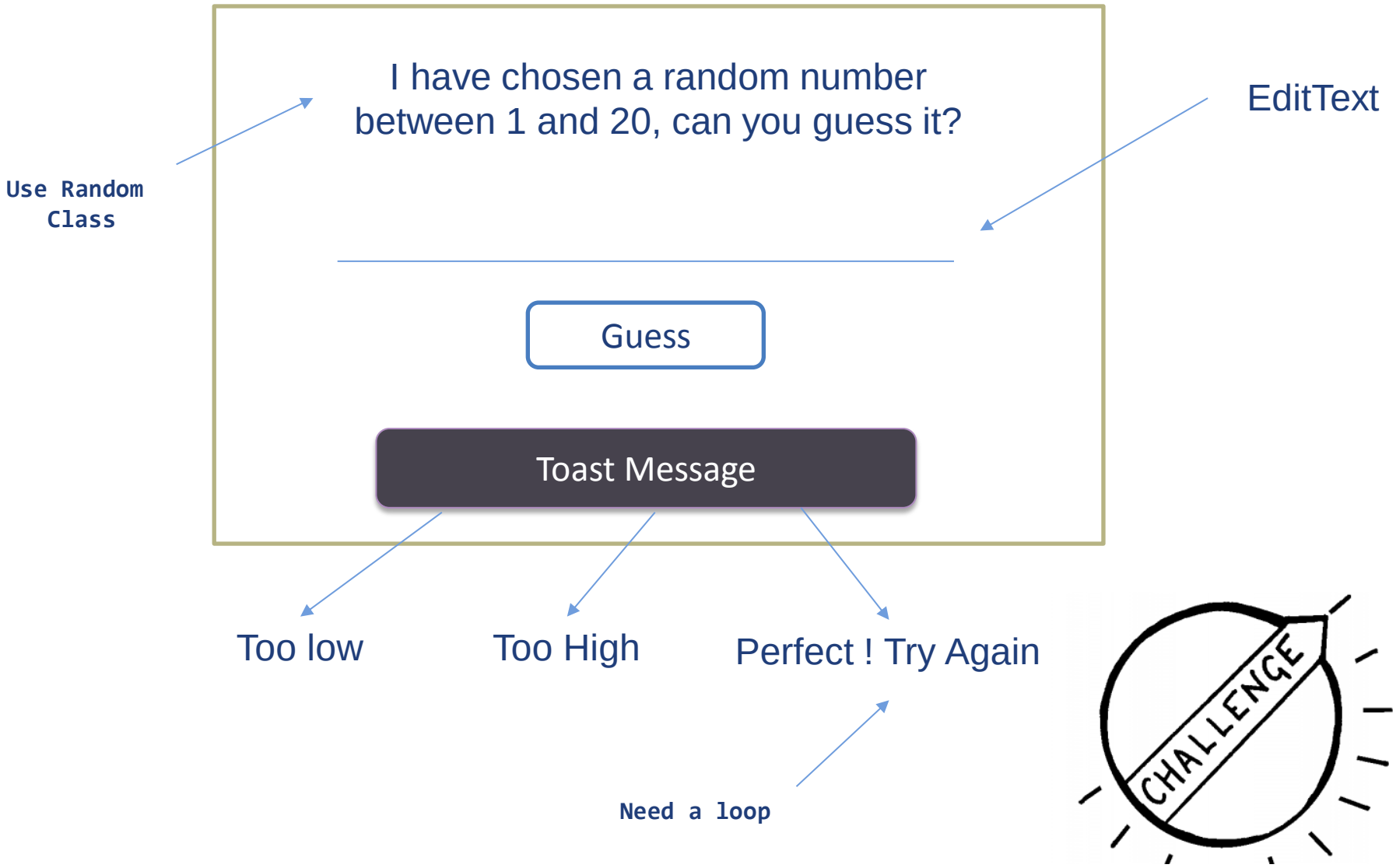
```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```



In Class Challenge

- Create a new Android App
- Display an icon image that contains a question mark
- Create a text field to allow users to enter their name
- They Press a Button
  - Get a pop-up (Toast) message Hello + Name.
  - The icon image shows a happy face instead of the question mark

Home Challenge



CheckBox

An individual toggleable on/off switch

key attributes:

`android:checked="`***`bool`***`"`	Set to true to make it initially checked
`android:clickable="`***`bool`***`"`	Set to false to disable it
`android:id="@+id/`***`theID`***`"`	Unique ID
`android:onClick="`***`function`***`"`	Function to call when clicked
`android:text="`***`text`***`"`	Text to put next the checkbox

- In Java code:

```
CheckBox cb = (CheckBox) findViewById(R.id.theID);
cb.toggle();
cb.setChecked(true);
cb.performClick();
```

RadioButton

A toggleable on/off switch; part of a group

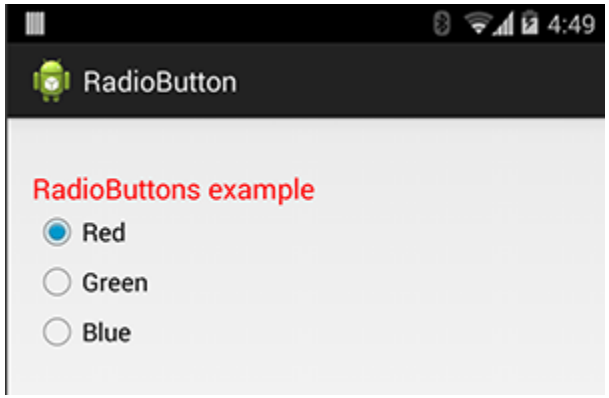
key attributes:

`android:checked="`***`bool`***`"`	Set to true to make it initially checked
`android:clickable="`***`bool`***`"`	Set to false to disable it
`android:id="@+id/`***`theID`***`"`	Unique ID
`android:onClick="`***`function`***`"`	Function to call when clicked
`android:text="`***`text`***`"`	Text to put next the checkbox

- need to be nested inside a `RadioGroup` tag in XML so that only one can be selected at a time

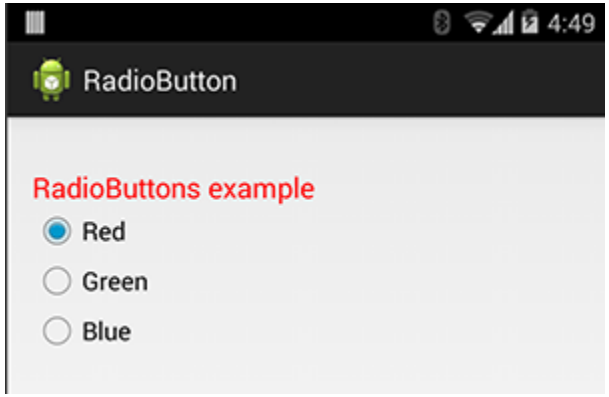
RadioGroup Example

```
<LinearLayout ...
        android:orientation="vertical"
        android:gravity="center|top">
    <RadioGroup ...
            android:orientation="horizontal">
        <RadioButton ... android:id="@+id/red"
                        android:text="Red"
                        android:onClick="radioClick" />
        <RadioButton ... android:id="@+id/green"
                        android:text="Green"
                        android:checked="true"
                        android:onClick="radioClick" />
        <RadioButton ... android:id="@+id/blue"
                        android:text="Blue"
                        android:onClick="radioClick" />
    </RadioGroup>
</LinearLayout>
```



Reusing onClick Handler

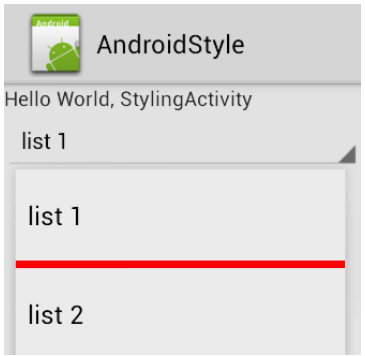
```
// in MainActivity.java
public class MainActivity extends Activity {

    public void radioClick(View view) {
        // check which radio button was clicked
        if (view.getId() == R.id.red) {
            // ...
        } else if (view.getId() == R.id.green) {
            // ...
        } else {
            // bears ...
        }
    }
```



Spinner

A drop-down menu of selectable choices

key attributes:



`android:clickable="`***`bool`***`"`	Set to false to disable it
`android:id="@+id/`***`theID`***`"`	Unique ID
`android:entries="@array/`***`array`***`"`	Set of options to appear in spinner (must match an array in strings.xml)

- also need to handle events in Java code
  - must get the Spinner object using findViewById
  - then call its setOnItemSelectedListener method

Spinner Example

```
<LinearLayout ...>
    <Spinner ... android:id="@+id/tmnt"
        android:entries="@array/turtles"
        android:prompt="@string/choose_turtle" />
    <TextView ... android:id="@+id/result" />
</LinearLayout>
```

in res/values/strings.xml:

```
<resources>
    <string name="choose_turtle">Choose a turtle:</string>
    <string-array name="turtles">
        <item>Leonardo</item>
        <item>Michelangelo</item>
        <item>Donatello</item>
        <item>Raphael</item>
    </string-array>
</resources>
```

Spinner Event Example


```
// in MainActivity.java

public class MainActivity extends Activity {
  ...
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Spinner spin = (Spinner) findViewById(R.id.tmnt);
    spin.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
      public void onItemSelected(AdapterView<?> spin, View v, int i, long id) {
        TextView result = (TextView) findViewById(R.id.turtle_result);
        result.setText("You chose " + spin.getSelectedItem());
      }
      public void onNothingSelected(AdapterView<?> parent) {}        // empty
    });
  }
}
```

ListView

An ordered collection of selectable choices

key attributes:

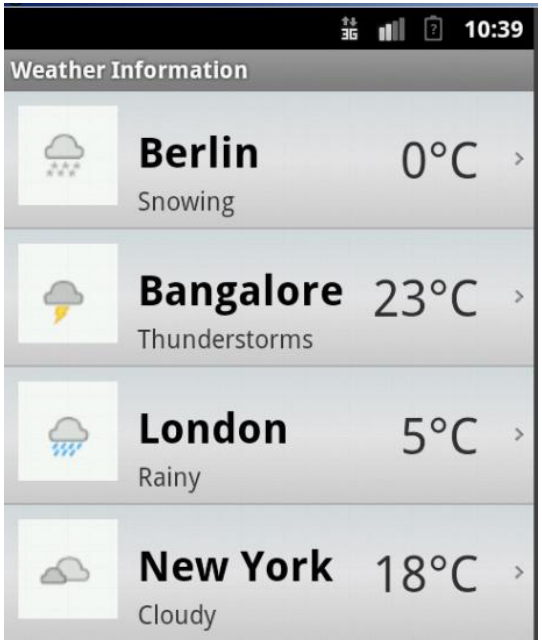
`android:clickable="`***`bool`***`"`	Set to false to disable it
`android:id="@+id/`***`theID`***`"`	Unique ID
`android:entries="@array/`***`array`***`"`	Set of options to appear in the list (must match an array in strings.xml)

- **static list**: Content is fixed and known before the app runs.
  - Declare the list elements in the **strings.xml** resource file.

Dynamic Lists

- **dynamic list**: Content is read or generated as the program runs.
  - Comes from a data file, or from the internet, etc.
  - Must be set in the Java code.
  - Suppose we have the following file and want to make a list from it:

```
// res/raw/oses.txt
Android
iPhone
...
Max OS X
```



List Adapters

- **adapter**: Helps turn list data into list view items.
  - common adapters: `ArrayAdapter`, `CursorAdapter`

- Syntax for creating an adapter:

```
ArrayAdapter<String> name =
    new ArrayAdapter<String>(activity, layout, array);
```

  - the ***activity*** is usually `this`
  - the default ***layout*** for lists is `android.R.layout.simple_list_item_1`
  - get the ***array*** by reading your file or data source of choice
    (it can be an array like `String[]`, or a list like `ArrayList<String>`)

- Once you have an adapter, you can attach it to your list by calling the `setAdapter` method of the `ListView` object in the Java code.

List Adapter Example

```
ArrayList<String> myArray = ...;                    // load data from file


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

ListView list = (ListView) findViewById(R.id.mylist);
list.setAdapter(myAdapter);
```

Handling List Events

- Unfortunately lists don't use a simple `onClick` event.
  - Several fancier GUI widgets use other kinds of events.
  - The event listeners must be attached in the Java code, not in the XML.
  - Understanding how to attach these event listeners requires the use of Java **anonymous inner classes**.

- **anonymous inner class**: A shorthand syntax for declaring a small class without giving it an explicit name.
  - The class can be made to extend a given super class or implement a given interface.
  - Typically the class is declared and a single object of it is constructed and used all at once.

Attaching Event Listener

```
<!-- activity_main.xml -->
<Button ... android:onClick="mybuttonOnClick" />
<Button ... android:id="@+id/mybutton" />

// MainActivity.java
public void mybuttonOnClick() { ... }
Button button = (Button) findViewById(R.id.mybutton);
button.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
            // code to run when the button gets clicked
      }
});
```

List Events

- List views respond to the following events:
  - **set**OnItemClickListener(AdapterView.OnItemClickListener)

    Listener for when an item in the list has been clicked.
  - **set**OnItemLongClickListener(AdapterView.OnItemLongClickListener)

    Listener for when an item in the list has been clicked and held.
  - **set**OnItemSelectedListener(AdapterView.OnItemSelectedListener)

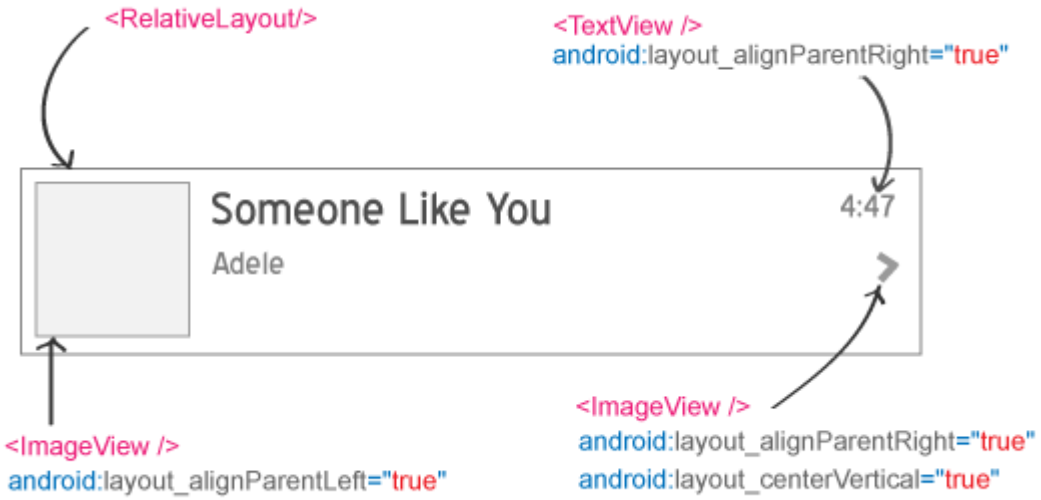
    Listener for when an item in the list has been selected.
- Others:
  - onDrag, onFocusChanged, onHover, onKey, onScroll, onTouch, ...

List Events Example

```
ListView list = (ListView) findViewById(R.id.id);
list.setOnItemClickListener(
     new AdapterView.OnItemClickListener() {
          @Override
          public void onItemClick(AdapterView<?> list,
                                  View row,
                                  int index,
                                  long rowID) {
               // code to run when user clicks that item
               ...
          }
     }
);
```

Custom List Layouts

- If you want your list to look different than the default appearance (of just a text string for each line), you must:
  - Write a short **layout XML file** describing the layout for **each row**.
  - Write a **subclass of `ArrayAdapter`** that overrides the **`getView`** method to describe what view must be returned for each row.



Custom List Layout XML

```
<!-- res/layout/mylistlayout.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ... android:orientation="horizontal">
      <ImageView ... android:id="@+id/list_row_image"
           android:layout_width="100dp"
           android:layout_height="100dp"
           android:src="@drawable/smiley" />

      <TextView ... android:id="@+id/list_row_text"
           android:textStyle="bold"
           android:textSize="22dp"
           android:text=""
           android:background="#336699" />
</LinearLayout>
```

Custom List Layout Java

```
// MyAdapter.java
public class MyAdapter extends ArrayAdapter<String> {
    private int layoutResourceId;
    private List<String> data;

    public MyAdapter(Context context, int layoutId, List<String> list) {
        super(context, layoutResourceId, data);
        layoutResourceId = layoutId;
        data = list;
    }

    @Override
    public View getView(int index, View row, ViewGroup parent) {
        row = getLayoutInflater().inflate(layoutResourceId, parent, false);
        TextView text = (TextView) row.findViewById(R.id.list_row_text);
        text.setText(data.get(index));
        return row;
    }
}
```

Recycled Views