ICT PROJECT DEVELOPMENT AND IMPLEMENTATION

SESSION #2

GUI (I)

OUTLINE

DESIGNING THE USER INTERFACE

- Layouts
- Widgets
- UI Events
- Resources
- Localization
- Menus

GUI

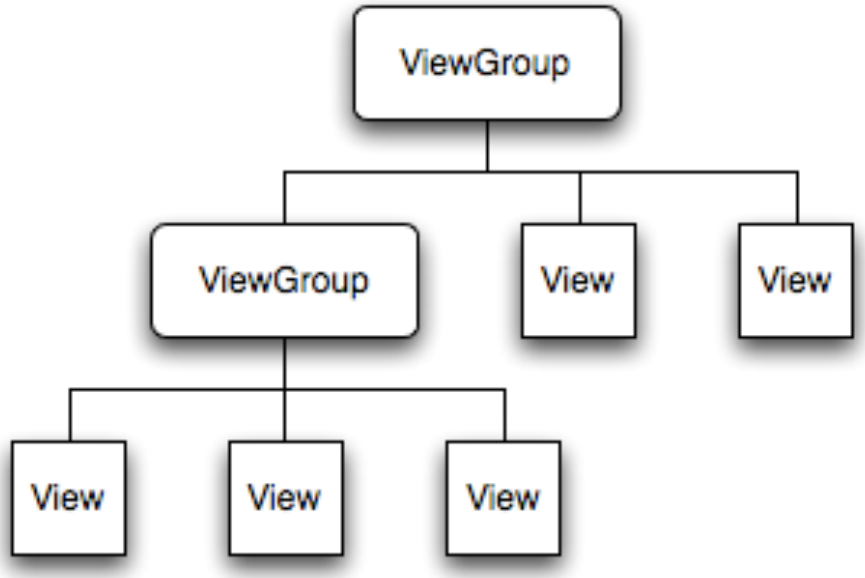
In an Android application, the user interface is built using View and ViewGroup

- **View objects** are the basic units of user interface expression on the Android platform.
- The **View class** serves as the base for subclasses called "**widgets**," which offer fully implemented UI objects, like text fields and buttons.
- The **ViewGroup class** serves as the base for subclasses called "**layouts**," which offer different kinds of layout architecture.

The Activity's UI is defined using a hierarchy of View and ViewGroup nodes.

In order to attach the view hierarchy tree to the screen for rendering, your Activity must call the **setContentView** method and pass a reference to the root node object.

e.g. **setContentView(R.layout.*main*);**



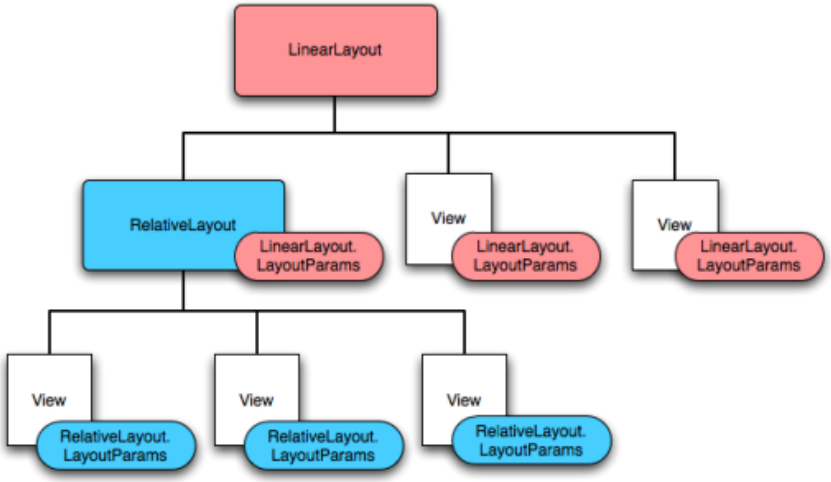
GUI

- **Layouts:**
  - A type of resource that defines what is drawn on the screen. A layout resource is simply a template for a user interface screen, **a container**.
  - A type of View class whose primary purpose is to **organize other controls** (LinearLayout, RelativeLayout, WebView, etc.).
- **Widgets:**
  - A widget is a **View** object that serves as an interface for interaction with the user (buttons, checkboxes, labels, text-entry fields, etc.).
- **Input Events:**
  - Is the way android inform the application (activity) about the **user's interactio**n with them, so it can perform actions.
- **Menus:**
  - Menus offers a reliable interface that reveals **application functions and settings**.

LAYOUTS

The layout is the **architecture** for the user interface in an Activity.

It defines the layout structure and **holds all the elements** that appear to the user.



You can declare your layout in two ways:

- **Declare UI elements in XML**. Android provides a XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime**. Your application can create View and ViewGroup objects (and manipulate their properties) from code.

LAYOUTS

Each layout file must contain exactly one **root element** (View or ViewGroup object)

You can add additional **layout objects** or **widgets** as child elements to gradually build a View hierarchy that defines your layout.

When the application is compiled, each XML layout file is compiled into a View resource. You should load the layout resource in your activity.onCreate() callback, by calling setContentView(), passing it your layout resource in the form of: R.layout.*layout_file_name*

LAYOUTS PARAMETERS

- XML layout attributes named layout_*something* define layout parameters for the View.
  - All view groups include a width and height parameter (layout_width and layout_height), and each view is required to define them. Many layout parameters also include optional margins and borders.
  - You can specify width and height with exact or relative to parent measurements:
    - wrap_content tells your view to size itself to the dimensions required by its content.
    - match_parent (former called fill_parent) tells your view to become as big as its parent view group will allow.

LAYOUTS PARAMETERS

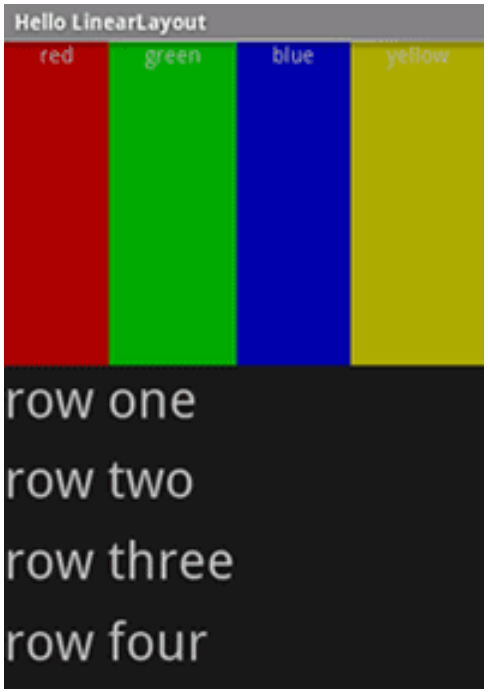
- Manage parameters from code:
  - **Layout Position**: A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a width and height.
    - The location (relative to its parent) can be retrieved invoking getLeft() and getTop(), getRight() and getBottom() methods.

  - **Size, Padding and Margins**
    - The size is expressed with a width and a height, and can be obtained by calling getWidth() and getHeight() for drawing dimensions.
    - To measure its dimensions, a view takes into account its padding. Padding can be set using the setPadding(int, int, int, int) method and queried by calling getPaddingLeft(), getPaddingTop(), getPaddingRight() and getPaddingBottom().

LAYOUTS PARAMETERS

ATTRIBUTE	DESCRIPTION
layout_width	Specifies the width of the View or ViewGroup (match_parent, 50dp…)
layout_height	Specifies the height of the View or ViewGroup (wrap_content, 50px…)
layout_marginTop	Specifies extra space on the top side of the View or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the View or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the View or ViewGroup
layout_marginRight	Specifies extra space on the right side of the View or ViewGroup
layout_gravity	Specifies how child Views are positioned (left, right,…)
layout_weight	Assigns an "importance" value to a view in terms of how much space it should occupy on the screen.
layout_x	Specifies the x-coordinate of the View or ViewGroup
layout_y	Specifies the y-coordinate of the View or ViewGroup

LINEAR LAYOUT



LinearLayout is a ViewGroup that displays child View elements in a linear direction, either **vertically** or **horizontally**.

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

  <LinearLayout
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_weight="1">
      <TextView
          android:text="red"
          android:gravity="center_horizontal"
          android:background="#aa0000"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="green"
          android:gravity="center_horizontal"
          android:background="#00aa00"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="blue"
          android:gravity="center_horizontal"
          android:background="#0000aa"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="yellow"
          android:gravity="center_horizontal"
          android:background="#aaaa00"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
  </LinearLayout>
```

```
  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">
    <TextView
        android:text="row one"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="row two"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="row three"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="row four"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
  </LinearLayout>

</LinearLayout>
```

RELATIVE LAYOUT


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

RelativeLayout is a ViewGroup that displays child View elements in relative positions. The position of a View can be specified as **relative to sibling** elements (such as to the left-of or below a given element) or in positions relative to the RelativeLayout parent area (such as **aligned to** the bottom, left of center).

RELATIVE LAYOUT

- RelativeLayout lets child views specify their position relative to the parent view or to each other.
- So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.
- By default, all child views are drawn at the top-left of the layout.

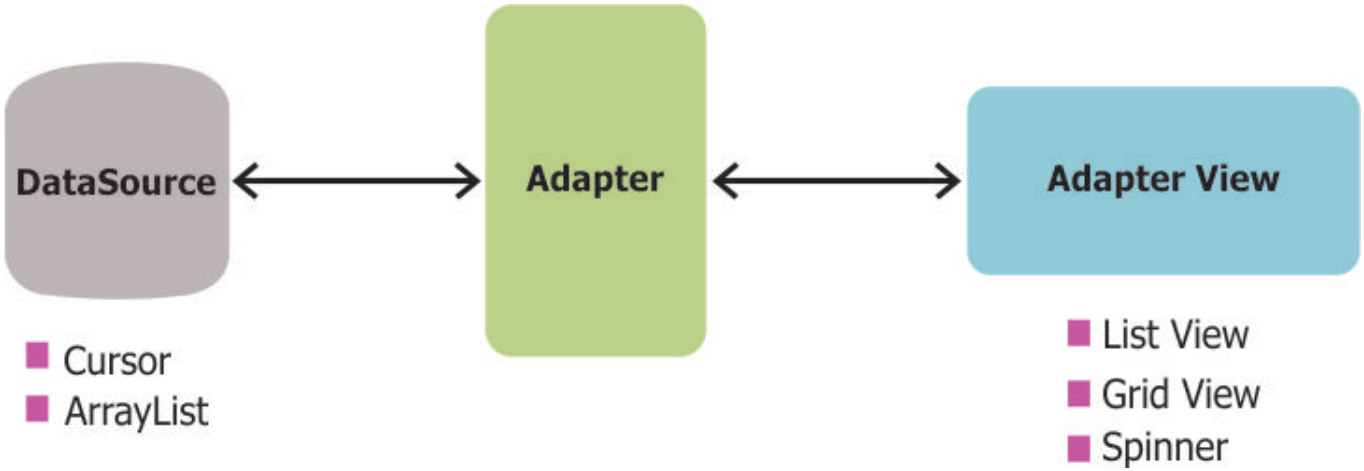
- Some examples:
  - layout_alignParentTop: makes the top edge of this view match the top edge of the parent.
  - layout_centerVertical: centers this child vertically within its parent.
  - layout_below: positions the top edge of this view below the view specified with a resource ID.
  - layout_toRightOf: positions the left edge of this view to the right of the view specified with a resource ID.

- RelativeLayout parameters
  - https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html

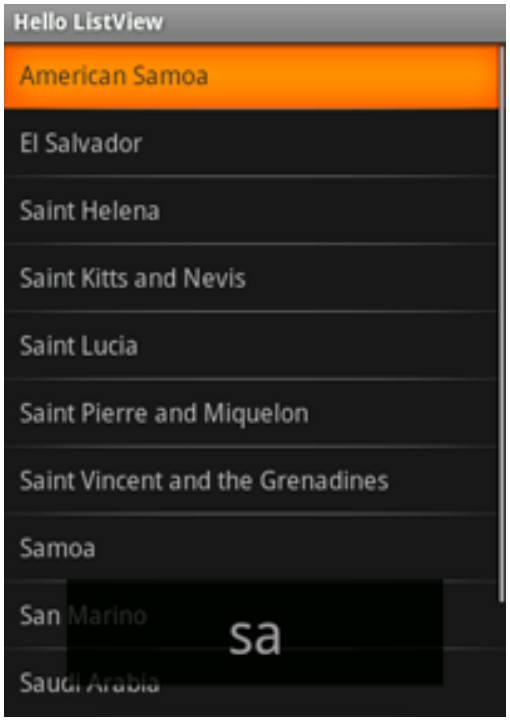
ADAPTER

An adapter loads the information to be displayed from a data source, such as an array or database query and creates a view for each item. Then it inserts the views into the ListView.

The adapter acts as the middle man between the ListView and data source, or its provider



LIST VIEW



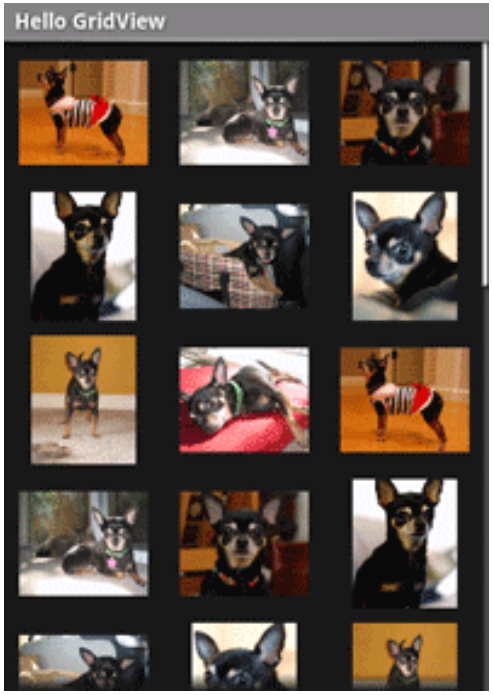
ListView is a ViewGroup that creates a list of scrollable items. The list items are automatically inserted to the list using a **ListAdapter**.

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

```
public class HelloListView extends ListActivity {
```

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

  setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item, COUNTRIES));

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);

  lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
      // When clicked, show a toast with the TextView text
      Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
          Toast.LENGTH_SHORT).show();
    }
  });
}
```

GRID VIEW



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

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position, Toast.LENGTH_SHORT).show();
        }
    });
}
```

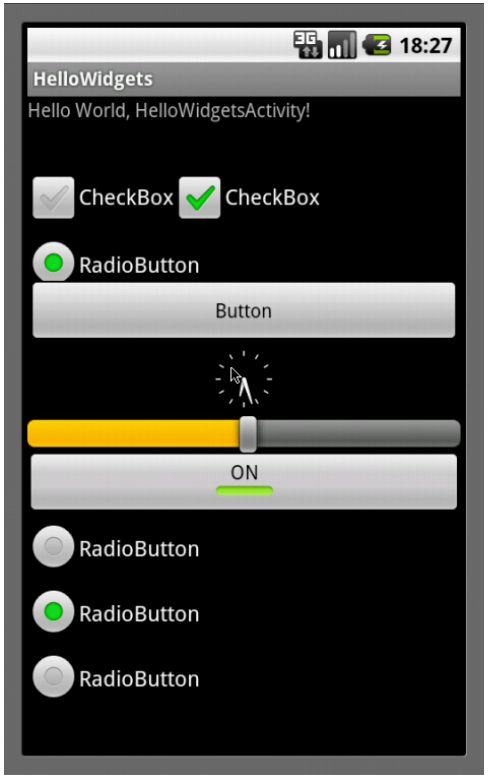
IMAGE ADAPTER CLASS

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

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

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

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

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

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {  // if it's not recycled, initialize some
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
```

```
    // references to our images
    private Integer[] mThumbIds = {
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7
    };
}
```

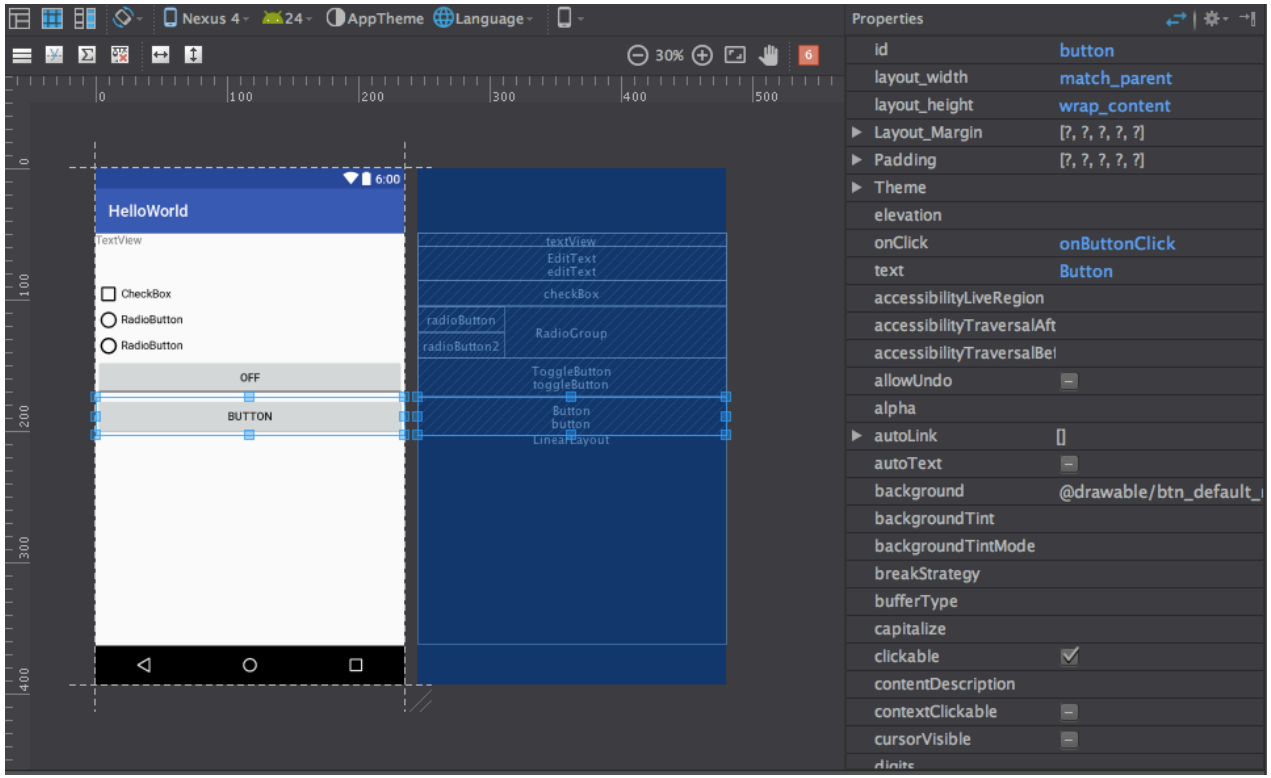
WIDGETS

- A widget is a View object that serves as an interface for interaction with the user.
- Android provides a rich set of widgets: button, checkbox, text field, date picker, ...
- Android is not limited to its own widgets. You can customized and create your own actionable elements by defining your own View object or by extending and combining existing widgets.
- Android Studio incorporates an UI editor allows you to create the UI via drag and drop or by means of XML source code.



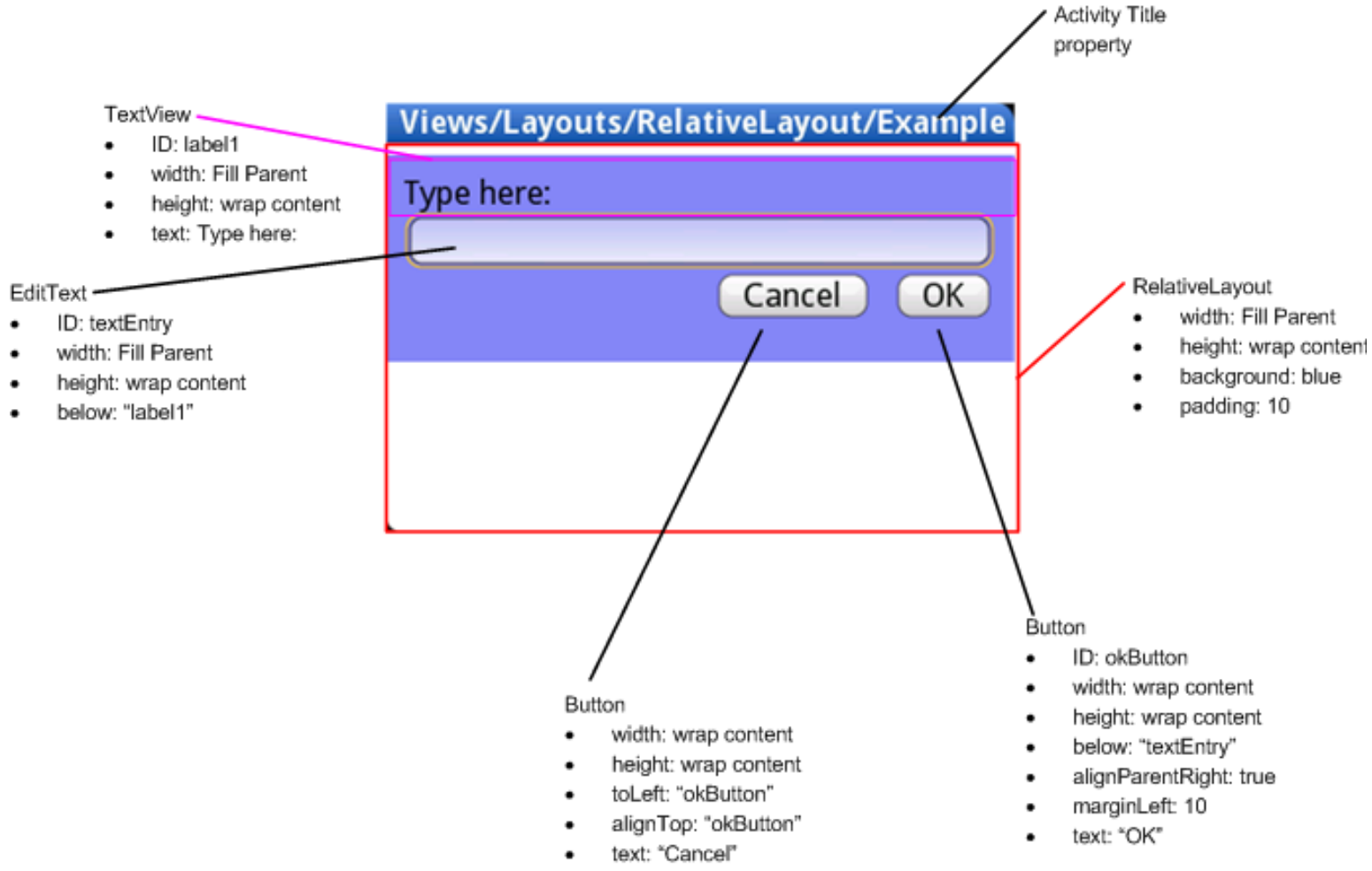
EDITING UI

- The properties of a UI element can be changed via the properties view.
- Most of the properties can also be changed via the right mouse menu.



USER INTERFACE

UI LAYOUTS ARE IN JAVA AND XML



setContentView(R.layout.hello_activity); //will load the XML UI file

INPUT EVENTS

- Android provides different methods to intercept the events from a user's interaction with an application.
- View classes may notice several public callback methods that look useful for UI events.
  - These methods are called by the Android framework as default **event handlers**, when the respective action occurs on that object. (For instance: When a View is touched, the onTouchEvent() method is called on that object)
  - This method requires extending every View object in order to handle such an event, which would not be practical.
- View class also contains a collection of nested interfaces with callbacks that you can much more easily define.
- These interfaces, called **event listeners**, allow to capture the user interaction with the user interface.

EVENT LISTENERS

- An event listener is an interface in the View class that contains a single callback method.
- These methods will be called by the framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

CALLBACK	FROM	DESCRIPTION
onClick()	View.OnClickListener	It is called when the user either touches the item, or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball
onLongClick()	View.OnLongClickListener	It is called when the user either touches and holds the item, or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).
...	...	...

EVENT LISTENERS

CALLBACK	FROM	DESCRIPTION
onFocusChange()	**View.OnFocusChangeListener**	This is called when the user navigates onto or away from the item.
onKey()	**View.OnKeyListener**	This is called when the user is focused on the item and presses or releases a key on the device.
onTouch()	**View.OnTouchListener**	This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).
onCreateContextMenu()	View.OnCreateContextMenuListener	This is called when a Context Menu is being built (as the result of a sustained "long click")

WIDGETS EXAMPLE

LAYOUT

- Using a vertical LinearLayout.
- Create the widgets.
- Assign properties.
- Define the event listeners.
  - Static way:
    - Layout:
      android:onClick="onButtonClick”
    - Activity:

      public void onButtonClick(View v) {… }
  - Dinamic way:
    - Activity: final Button button =
      (Button) findViewById(R.id.button);

      button.setOnClickListener(new
      View.OnClickListener() {
      @Override
      public void onClick(View v) { } });

WIDGETS EXAMPLE

EDIT TEXT

Add a Text Field for user input, using the EditText widget. Once text has been entered, the "Enter" key will display the text in a toast message.

1. Add the EditText element to the main.xml layout:

```
    <EditText
        android:id="@+id/edittext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
```

2. Add the OnKey listener to the end of the onCreate() method to show a toast message when the user press the ENTER key:

```
final EditText edittext = (EditText) findViewById(R.id.edittext);
edittext.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        // If the event is a key-down event on the "enter" button
        if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
            (keyCode == KeyEvent.KEYCODE_ENTER)) {
          // Perform action on key press
          Toast.makeText(HelloFormStuff.this, edittext.getText(), Toast.LENGTH_SHORT).show();
          return true;
        }
        return false;
    }
});
```

3. Execute the application.

WIDGETS EXAMPLE

CHECK BOX

Create a checkbox for selecting items, using the CheckBox widget.

When the checkbox is pressed, a toast message will indicate the current state of the checkbox.

1. Add the CheckBox element to the main.xml layout:

```
<CheckBox android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="check it out" />
```

2. Add the OnClick listener to the checkbox at the end of the onCreate() method, to show a toast message when the checkbox state changes:

```
final CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox);
checkbox.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks, depending on whether it's now checked
        if (((CheckBox) v).isChecked()) {
            Toast.makeText(HelloFormStuff.this, "Selected", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(HelloFormStuff.this, "Not selected", Toast.LENGTH_SHORT).show();
        }
    }
});
```

WIDGETS EXAMPLE

RADIO BUTTON

Now, we will create two mutually-exclusive radio buttons (enabling one disables the other), using the RadioGroup and RadioButton widgets.

When either radio button is pressed, a toast message will be displayed.

1. Add two RadioButtons, nested in a RadioGroup the main.xml layout:

```
<RadioGroup
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical">
  <RadioButton android:id="@+id/radio_red"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Red" />
  <RadioButton android:id="@+id/radio_blue"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Blue" />
</RadioGroup>
```

It's important that the RadioButtons are grouped together by the RadioGroup element so that **no** more than one can be selected at a time.

- This logic is automatically handled by the Android system. When one RadioButton within a group is selected, all others are automatically deselected.

WIDGETS EXAMPLE

RADIO BUTTON

1. To do something when each RadioButton is selected, you need an View.OnClickListener for this element.

   In this case, we want that the listener to be re-usable, so we add the following code to create a new member in the HelloFormStuff Activity:

```
private OnClickListener radio_listener = new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        RadioButton rb = (RadioButton) v;
        Toast.makeText(HelloFormStuff.this, rb.getText(), Toast.LENGTH_SHORT).show();
    }
};
```

2. Now, at the bottom of the onCreate() method, add the following code to add the newly created View.OnClickListener listener to each one:

```
final RadioButton radio_red = (RadioButton) findViewById(R.id.radio_red);
final RadioButton radio_blue = (RadioButton) findViewById(R.id.radio_blue);
radio_red.setOnClickListener(radio_listener);
radio_blue.setOnClickListener(radio_listener);
```

3. Run the application

WIDGETS EXAMPLE

TOGGLE BUTTON

Create a button used specifically for toggling between two states, using the ToggleButton widget.

This widget is an excellent alternative to radio buttons if you have two simple states that are mutually exclusive ("on" and "off", for example).

1. Add the ToggleButton element to the main.xml layout:

```
    <ToggleButton android:id="@+id/togglebutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="Vibrate on"
        android:textOff="Vibrate off"/>
```

2. Add the OnClick listener to the end of the onCreate() method:

```
final ToggleButton togglebutton = (ToggleButton) findViewById(R.id.togglebutton);
togglebutton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        if (togglebutton.isChecked()) {
            Toast.makeText(HelloFormStuff.this, "Checked", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(HelloFormStuff.this, "Not checked", Toast.LENGTH_SHORT).show();
        }
    }
});
```

3. Execute the application.

GUI EXAMPLE

- ✧ Create UI for the conversion of Celsius temperature to Fahrenheit Temperature.
- ✧ The UI requires:
  - Three linear layouts.
  - A TextView.
  - An EditText view.
  - A group of two radio buttons.
  - A button.

GUI EXAMPLE

SOLUTION

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/an
   droid"
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:background="@color/myColor">
7     <TextView android:layout_height="wrap_content"
8        android:text="Temperature:"
9        android:id="@+id/textView1"
10       android:textSize="9pt"
11              android:textColor="#ff0000ff"
12       android:layout_width="match_parent">
13    </TextView>
14    <EditText android:layout_height="wrap_content"
15          android:id="@+id/editText1"
16       android:layout_width="match_parent"
17
         android:inputType="numberDecimal|numberSigned">
18    </EditText>
19    <LinearLayout android:orientation="horizontal"
20    android:layout_width="fill_parent"
21    android:layout_height="wrap_content" >
22          <LinearLayout android:orientation="vertical"
23          android:layout_width="wrap_content"
24          android:layout_height="wrap_content" >
25          <RadioGroup
            android:layout_height="wrap_content"
26                android:id="@+id/radioGroup1"
27             android:layout_width="match_parent">
28             <RadioButton
               android:layout_width="wrap_content"
29                android:id="@+id/radio0"
30
                  android:layout_height="wrap_content"
31                android:text="@string/celsius"
32                android:checked="true">
33             </RadioButton>
34             <RadioButton
               android:layout_width="wrap_content"
35                android:id="@+id/radio1"
36
                  android:layout_height="wrap_content"
37                android:text="@string/fahrenheit">
38             </RadioButton>
39               </RadioGroup>
40          </LinearLayout>
41       <Button android:id="@+id/button1"
42          android:text="@string/calc"
43          android:onClick="myClickHandler"
44          android:layout_width="250px"
45                android:layout_height="60px"
46                android:layout_marginLeft="20px"
47                android:layout_gravity="center_vertical">
48       </Button>
49       </LinearLayout>
50    <Button android:id="@+id/buttonexit"
56    </Button>
57 </LinearLayout>
```

EVENTS EXAMPLE

Capture the user interaction events for:

- Button Calculate: onClick event to perform the calculation, using an event Handler
- Button Exit: onClick event to exit activity, using an anonymous event listener.
- EditText: on ENTER Key press event to perform also the temperature conversion, using an event listener.

SOLUTION

EVENTS EXAMPLE

```
  private void doConversion()
  {
        RadioButton celsiusButton = (RadioButton) findViewById(R.id.radio0);
        RadioButton fahrenheitButton = (RadioButton) findViewById(R.id.radio1);
        if (text.getText().length() == 0) {
            Toast.makeText(this, "Please enter a valid number",
                    Toast.LENGTH_LONG).show();
            return;
        }

        float inputValue = Float.parseFloat(text.getText().toString());
        if (celsiusButton.isChecked()) {
            text.setText(String
                    .valueOf(convertCelsiusToFahrenheit(inputValue)));
            celsiusButton.setChecked(false);
            fahrenheitButton.setChecked(true);
        } else {
            text.setText(String
                    .valueOf(convertFahrenheitToCelsius(inputValue)));
            fahrenheitButton.setChecked(false);
            celsiusButton.setChecked(true);
        }
  }


  // Converts to celsius
  private float convertFahrenheitToCelsius(float fahrenheit) {
      return ((fahrenheit - 32) * 5 / 9);
  }

  // Converts to fahrenheit
  private float convertCelsiusToFahrenheit(float celsius) {
      return ((celsius * 9) / 5) + 32;
  }
}
```

SOLUTION

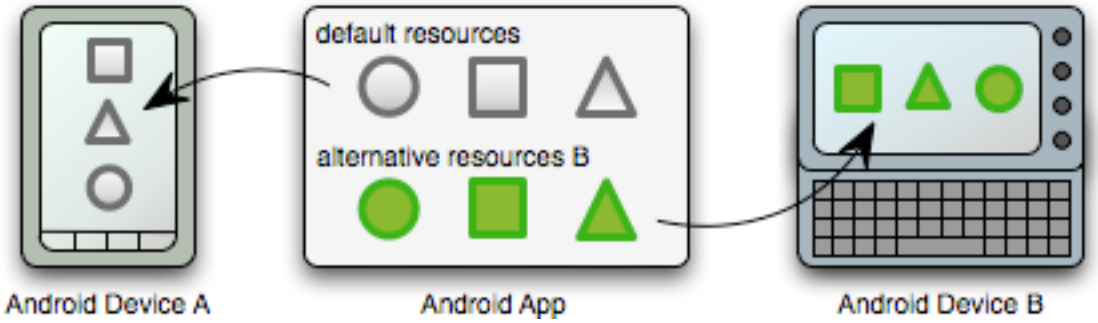
EVENTS EXAMPLE

```
public class TemperatureConverterActivity extends Activity implements OnKeyListener {

  static final int DIALOG_EXIT_ID = 0;
  static final int DIALOG_WAITING_ID = 1;

  private EditText text;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        text = (EditText) findViewById(R.id.editText1);
        text.setOnKeyListener(this);

        final Button buttonExit = (Button) findViewById(R.id.buttonexit);
        buttonExit.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // Perform action on clicks
              showDialog(DIALOG_EXIT_ID);
            }
        });

    }

    // Implement the OnClickListener callback
    public boolean onKey(View v, int keycode, KeyEvent keyevent) {
      // We only want to handle ACTION_UP events, when user releases a key
      if(keyevent.getAction()==KeyEvent.ACTION_DOWN)
        return false;

      if (keycode == KeyEvent.KEYCODE_ENTER)
      {
            doConversion();
            return true;
      }
      else return false;
    }
```

```
// This method is called at button click because we assigned the name to
// "On Click property" of the button
public void myClickHandler(View view) {
    switch (view.getId()) {
    case R.id.button1:
        doConversion();
        break;
    }
}
```

RESOURCES

- You should always externalize resources, so that you can maintain them independently
  - Support specific devices configurations
  - Support multiple languages
- You can specify *default* and multiple *alternative* resources for your application:
  - **Default resources**: Independent of device or there are no alternative.
  - **Alternative resources:** To be used with a specific configuration



RESOURCES

- The resources are provided by grouping them in specially-named resource directories.

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            main.xml
            info.xml
        values/
            strings.xml
```

- At runtime, Android uses the appropriate resource based on the current configuration.
- Resources can be accessed using resource IDs that are generated in your project's R class

RESOURCES

DIRECTORY	RESOURCE TYPE	CLASS
anim/	XML files that define tween animations	R.anim
color/	XML files that define a state list of colors.	R.color
drawable/	Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into drawable resources.	R.drawable
layout/	XML files that define a user interface layout.	R.layout
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu.	R.menu

RESOURCES

DIRECTORY	RESOURCE TYPE	CLASS
raw/	Arbitrary files to save in their raw form.	R.raw
values/	XML files that contain simple values, such as strings, integers, and colors. Files in the values/ directory describe multiple resources: • arrays.xml for resource arrays (typed arrays). • colors.xml for color values • dimens.xml for dimension values. • strings.xml for string values. • styles.xml for styles.	R.array R.color R.dimen R.string
xml/	Arbitrary XML files that can be read at runtime by calling Resources.getXML().	
mipmap/	Drawable files for different launcher icon densities.	R.mipmap

RESOURCES

ALTERNATIVE RESOURCES

- Almost every application should provide alternative resources to support specific device configurations.
- To specify configuration-specific alternatives for a set of resources:
  - Create a new directory in res/ named in the form *<resources_name>-<config_qualifier>*.
    - *<resources_name>* is the directory name of the corresponding default resources (previous table).
    - *<qualifier>* is a name that specifies an individual configuration for which these resources are to be used: screen pixel density (ldpi, mdpi, hdpi,…), language (en, es, ca,..), screen size (small, normal, large,..), screen orientation (port, land), night mode (night, notnight), ect.
  - You can append more than one *<qualifier>*. Separate each one with a dash:

    drawable-en-rUS-land


```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

More Information: *http://developer.android.com/guide/topics/resources/providing-resources.html*

RESOURCES

ACCESSING RESOURCES

For each resource of some type, there is a static integer, ***resource ID***, that can be used to reference/retrieve the resource:

- Resources can be referenced from code using integers from R.java file:

  R.drawable.myimage

  R.string.hello

- Resources can be referenced from resources using a special XML syntax:

  @drawable/myimage

  @string/hello

- You can also access your app resources with methods in Resources class.

  getResources(). getDrawable(R.drawable.myimage);

  getResources().getText(R.string.hello);

ACCESSING RESOURCES

- You can use a resource in code by passing the resource ID as a method parameter.

  [<package_name>.]R.<resource_type>.<resource_name>

- For example:

```
//  Set an imageview to use res/drawable/myimage.png resource
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
// Load a background for the current screen from a drawable resource
getWindow().setBackgroundDrawableResource(R.drawable.my_back_image);
// Set the Activity title by getting a string from the Resources object
getWindow().setTitle(getResources().getText(R.string.main_title));
// Load a custom layout for the current screen
setContentView(R.layout.main_screen);
// Set the text on a TextView object using a resource ID
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

ACCESSING RESOURCES

- You can define values for some XML attributes and elements using a reference to an existing resource:

  @[<package_name>:]<resource_type>/<resource_name> <package_name>

- For example:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <string name="hello">Hello!</string>
</resources>
```

Resource name

Resource type

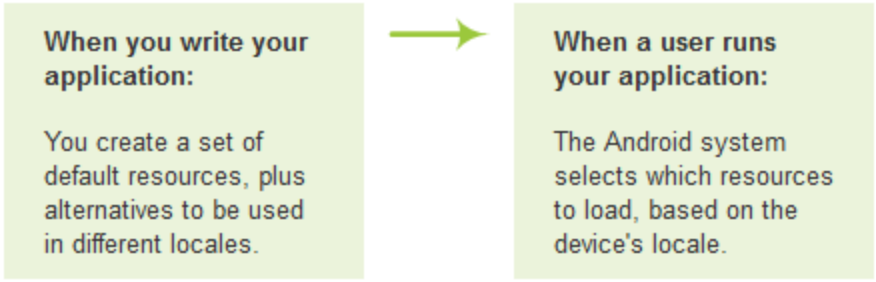
```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

External package resource

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@android:color/secondary_text_dark"
    android:text="@string/hello" />
```

LOCALIZATION

- Android applications will run on many devices in many regions.
  - To reach the most users, your application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your application will be used.
- Android lets you create different resource sets for different locales.
  - When your application runs, Android will load the resource set that match the device's locale.
  - If locale-specific resources are not available, Android falls back to defaults.
- Use the resource framework to separate the localized aspects of your application as much as possible from the core Java functionality



LOCALIZATION

- Design your application to **work in any locale**.
  - You cannot assume anything about the device on which a user will run your application.
- Design a **flexible layout**
  - It is better to create a single layout that is more flexible and can be adapted to different and longer languages.
- Avoid creating more resource files and text strings than you need
- Use the Android Context object for manual locale lookup:

```
// Returns this locale's language name, country name, and variant
String locale =
context.getResources().getConfiguration().locale.getDisplayName();
```

  - For example:

    Locale("en", "US", "POSIX") -> English (United States,Computer)

LOCALIZATION

- Create **default** resources:
  - Put the application's default text in a file with the following location and name: res/values/strings.xml
  - The text strings should use the default language.
- Create **alternative** resources:
  - You need to provide alternative text for different languages.
  - In some cases you will also provide alternative graphics, sounds, layouts, and other locale-specific resources.
- Example:
  - res/values/strings.xml
    Contains English text for all the strings that the application uses, including title.
  - res/values-fr/strings.xml
    Contain French text for all the strings, including title.
  - At rutime, If your Java code refers to **R.string.title**:
    - If the device is set to any language other than French, Android will load title from the res/values/strings.xml file.
    - If the device is set to French, Android will load title from the res/values-fr/strings.xml file.
  - ISO 639-1 language definition: *http://www.loc.gov/standards/iso639-2/php/code_list.php*
  - ISO 3166-1-alpha-2 region code: *http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html*

HELLO LOCALIZATION

- Do the localization tutorial from Android portal:
  http://developer.android.com/resources/tutorials/localization/index.html
  http://tool.oschina.net/uploads/apidocs/android/resources/tutorials/localization/index.html
- In this tutorial, we will create a Hello, L10N application that uses the Android framework to selectively load resources. Then we will localize the application by adding resources to the res/ directory.
  - Create an unlocalized Application
    - Create the Project and Layout
    - Create default Resources
    - Run the Unlocalized application
  - Modify the application to be used with different locales (german, french, english-usa, spanish, catalonian, etc.)
    - Localize the strings
    - Localize the images
    - Run and Test the Localized Application

HELLO LOCALIZATION

CREATE THE PROJECT AND LAYOUT

- Start a new project and Activity called "HelloLocalization“
  - *Project name:* HelloLocalization
  - *Application name:* Hello Localization
  - *Package name:* com.ejemplos.hellolocaliztion
  - *Create Activity:* HelloLocalizationActivity
  - *Min SDK Version: 7*
- Open the *res/layout/ main.xml* file and replace it with the right code:
  - Two textviews
  - One button

```
<?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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:text="@string/text_a"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:text="@string/text_b"
    />
<Button
    android:id="@+id/flag_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    />
</LinearLayout>
```

HELLO LOCALIZATION

CREATE DEFAULT RESOURCES (I)

- ✧ Create default text strings.
  - Open the res/values/strings.xml file and replace it with the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hello, L10N</string>
    <string name="text_a">Shall I compare thee to a summer"'"s day?</string>
    <string name="text_b">Thou art more lovely and more temperate.</string>
    <string name="dialog_title">No Localisation</string>
    <string name="dialog_text">This dialog box"'"s strings are not localised. For every locale, the text here will come from values/strings.xml.</string>
</resources>
```

- ✧ Add a default flag graphic to the res/drawable folder by saving flag.png as res/drawable/flag.png.

  flag: *http://developer.android.com/images/hello_l10n/flag.png*

  http://jayxie.com/mirrors/android-sdk/images/hello_l10n/flag.png

- ✧ F5 for refresh the project and show it the new res folder

HELLO LOCALIZATION

CREATE DEFAULT RESOURCES (II)

Open HelloLocalization.java (in the src/ directory) and add the following code inside the onCreate() method (after setContentView).

```
// assign flag.png to the button, loading correct flag image for current locale
Button b;
(b = (Button)findViewById(R.id.flag_button)).setBackgroundDrawable(this.getResources().getDrawable(R.drawable.flag));

// build dialog box to display when user clicks the flag
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage(R.string.dialog_text)
    .setCancelable(false)
    .setTitle(R.string.dialog_title)
    .setPositiveButton("Done", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
        dialog.dismiss();
        }
    });
final AlertDialog alert = builder.create();

// set click listener on the flag to show the dialog box
b.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        alert.show();
    }
    });
```

Set the flag icon to the button

Create new alert dialog box

Setting up the Alert Dialog properties: NotCanceable, title and the on-click listener.

Set a click listener to display the Alert Dialog when the user clicks the button

HELLO LOCALIZATION

RUN THE UNLOCALIZED APPLICATION

Execute the un-localized application (you can change your locale using the Custom Locale tool in the android virtual device).

HELLO LOCALIZATION

PLAN THE LOCALIZATION

The first step in localizing an application is to plan how the application will render differently in different locales.

- Define default (English) and alternative (English-USA, German, French, French-Canada, English-Canada, Japanese, Spanish) locales.

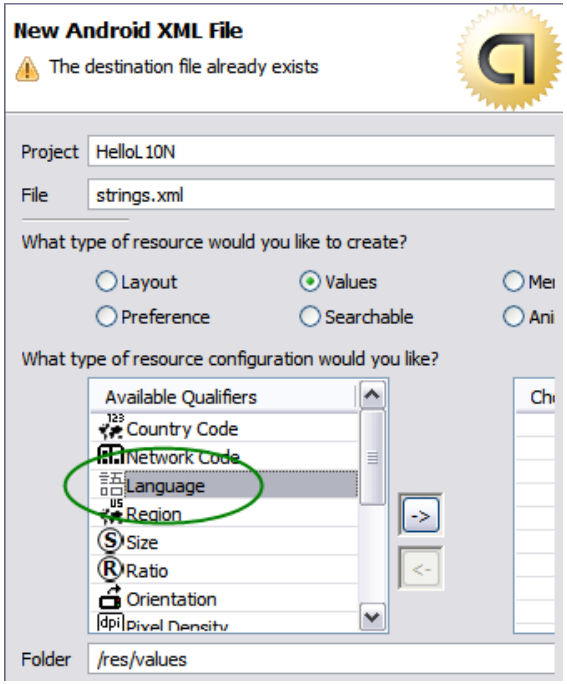
Locale Code	Language / Country	Location of strings.xml	Location of flag.png
Default	English / United Kingdom	res/values/	res/drawable/
de-rDE	German / Germany	res/values-de/	res/drawable-de-rDE/
fr-rFR	French / France	res/values-fr/	res/drawable-fr-rFR/
fr-rCA	French / Canada	res/values-fr/	res/drawable-fr-rCA/
en-rCA	English / Canada	*(res/values/)*	res/drawable-en-rCA/
ja-rJP	Japanese / Japan	res/values-ja/	res/drawable-ja-rJP/
en-rUS	English / United States	*(res/values/)*	res/drawable-en-rUS/
es-rES	Spanish / Spain	res/values-es/	res/drawable-es-rES/

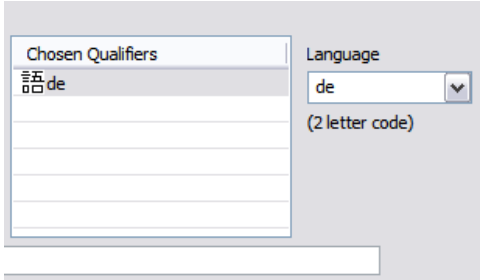
HELLO LOCALIZATION

LOCALIZE THE STRINGS

The application requires four more strings.xml files, one each for German, French, Japanese and Spanish.

- Select **File** > **New** > **Android XML File** to open the **New Android XML File wizard** (you can open the wizard by the icon in the toolbar).
- In the **New Android XML File** wizard:
  - Select *HelloLocalization* for the Project field, and type *strings.xml* into the File field.
  - In the left-hand list, select Language, then click the right arrow.
  - Type the language cod (“de”) in the Language box and click Finish.





HELLO LOCALIZATION

LOCALIZE THE IMAGES

The application requires seven more drawable folders, each containing a flag.png icon.

We have to add the needed icons and folders to your project.

For example:

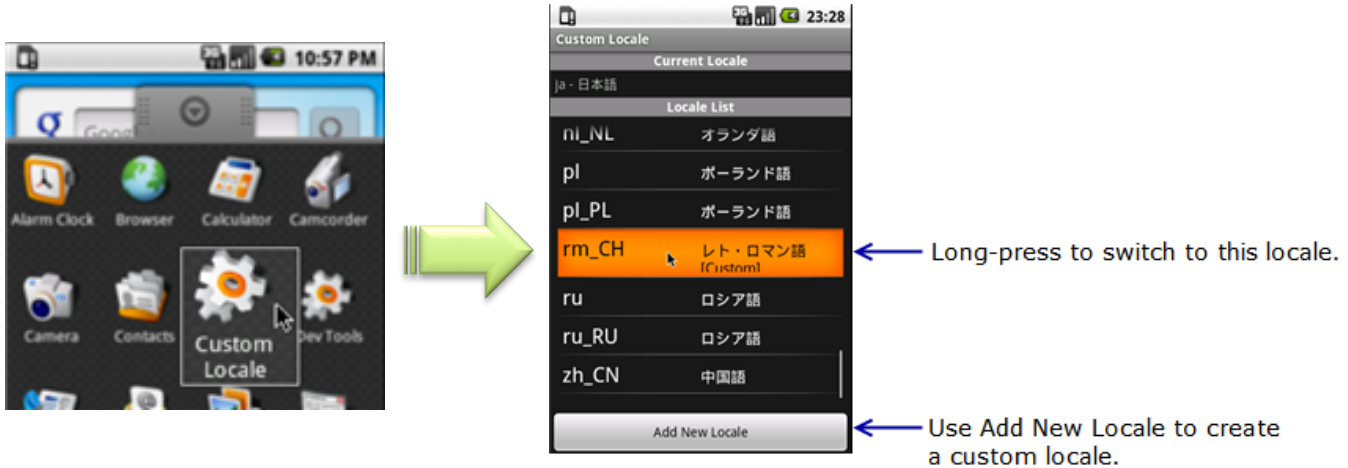
- ✧ Save this *German flag icon* as res/drawable-de-rDE/flag.png in the application's project workspace:
  1. Click the link to open the flag image.
  2. Save the image in *your-workspace*/HelloL10N/res/drawable-de-rDE/
- ✧ Repeat for all the flag icons (french, canada, Japanese, USA and Spain flags)

HELLO LOCALIZATION

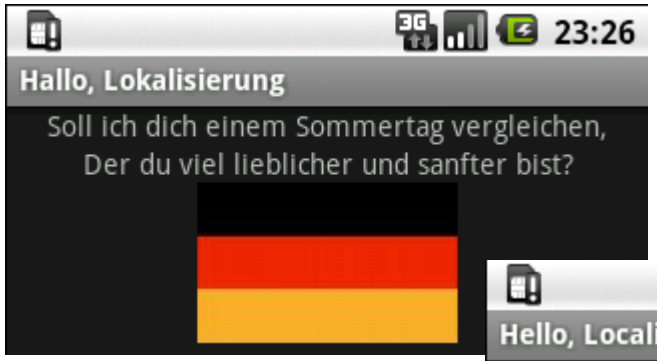
Run and Test the localization Application

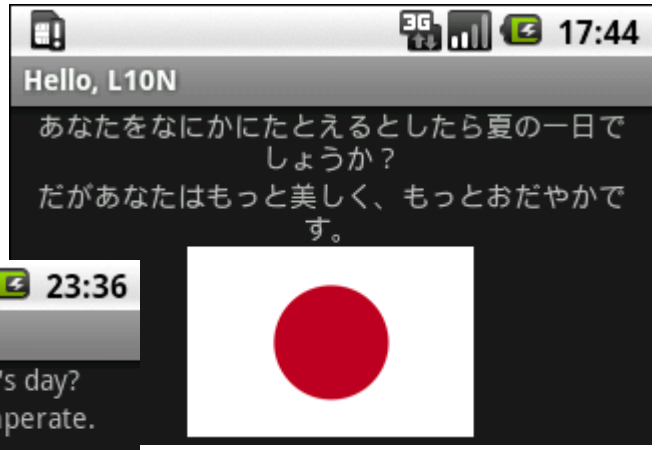
Execute the application using different locales and test its handling of the added localized strings and images resources.

- To change the locale on a device or in the emulator, use the Settings application (**Home** > **Menu** > Settings > **Language & keyboard** > **Select locale**)
- To set the emulator to a locale that is not available in the system image, use the Custom Locale application (available in the Application tab):



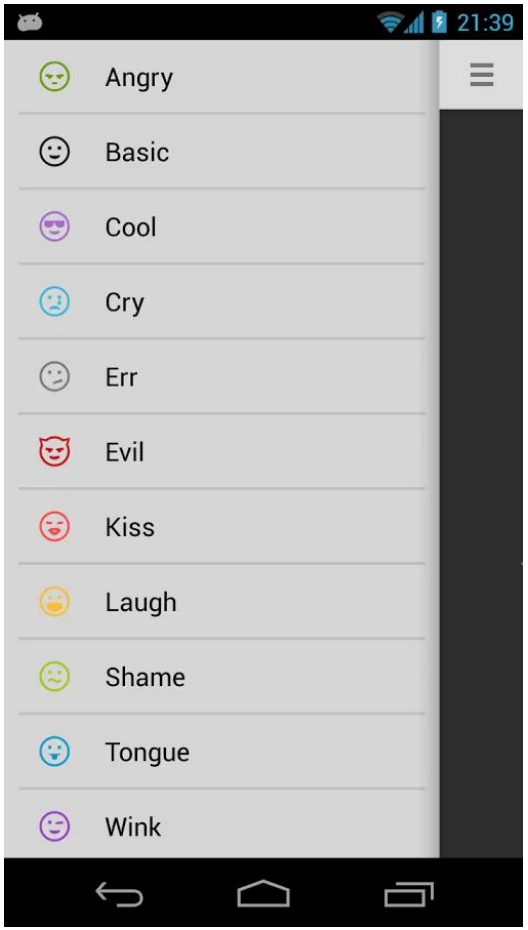
HELLO LOCALIZATION





MENUS

- Menus are a common user interface component in many types of applications. They provide users a familiar way to perform actions.
- There are three types of application menus:
  - **Options menu:** the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search" or "Settings".
  - **Contextual menu:** is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.
  - **Popup menu:** A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu (submenu).



MENUS

- For creating a menu, you should define a menu and all its items in an XML menu resource, then inflate the menu resource (load it as a programmable object) in your application code.
- To create a menu resource, create an XML file inside your project's res/menu/ directory and build the menu with the following elements:
  - **<menu>** defines a Menu, which is a **container** for menu items. A menu can hold one or more <item> and <group> elements.
  - **<item>** creates a MenuItem, which represents a **single ite**m in a menu. This element may contain a nested <menu> element in order to create a submenu.
  - **<group>** an optional, **invisible container** for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game" />
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

OPTIONS MENU

- The Options Menu is where you should include basic activity actions and necessary navigation items.
- When the Android creates the options menu for first time, it calls your activity's onCreateOptionsMenu() method.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

- You can also populate the menu in code, using add() to add items to the Menu:

  menu.add(0, EDIT, NONE, R.string.*menu_edit*);

- To change the options menu after it's first created, you must override the onPrepareOptionsMenu() method.

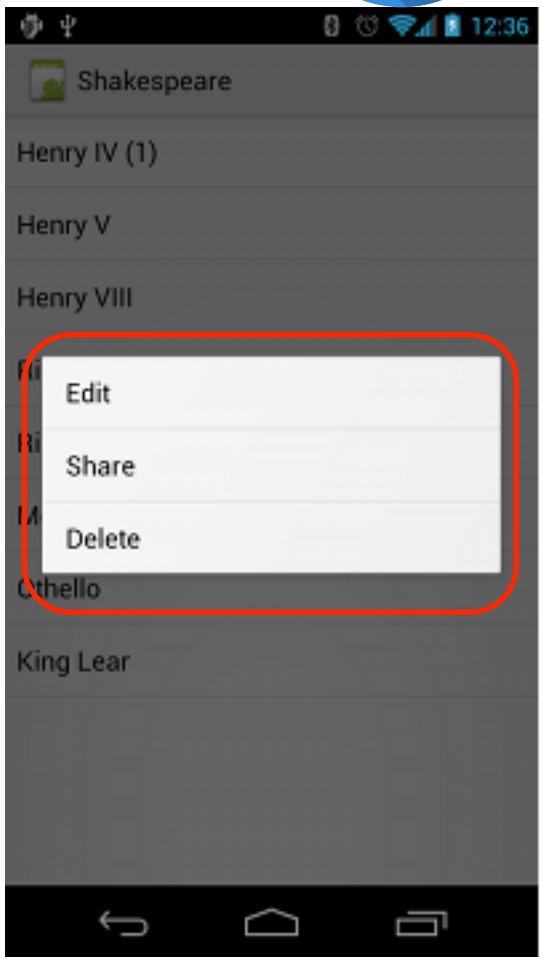
OPTIONS MENU

- When the user selects a menu item from the Options Menu, the system calls your activity's onOptionsItemSelected() method.
- This method passes the MenuItem that the user selected.
  - You can identify the menu item by calling getItemId(), which returns the unique ID for the menu item.
  - You can match this ID against known menu items and perform the appropriate action.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.new_game:
        newGame();
        return true;
    case R.id.help:
        showHelp();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

CONTEXT MENU

- A context menu is conceptually similar to the menu displayed when the user performs a "right-click" on a PC.
- You should use a context menu to provide the user access to actions that belongs to a specific item in the user interface.
- To provide a context menu in a View, you must "register" the view for a context menu.
  - Call registerForContextMenu() and pass it the View you want to give a context menu.
  - To define the context menu's appearance and behavior, override your activity's context menu callback methods, onCreateContextMenu() and onContextItemSelected().




```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.context_menu, menu);
}
```

CONTEXT MENU

- When the user selects an item from the context menu, the system calls onContextItemSelected().
- This method passes the MenuItem that the user selected.
  - You can identify the menu item by calling getItemId(), which returns the unique ID for the menu item.
  - You can match this ID against known menu items and perform the appropriate action.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
  switch (item.getItemId()) {
  case R.id.edit:
    editNote(info.id);
    return true;
  case R.id.delete:
    deleteNote(info.id);
    return true;
  default:
    return super.onContextItemSelected(item);
  }
}
```