COMP-304
Fall 2018

Review of Lecture 3

- **Declaration of Application resources**:
  - Declare resources, such as Strings, Integers, Booleans, Colors, Drawables, String Arrays, etc., in XML files
  - Common files to use:
    - **strings.xml**
    - **color.xml**
    - **dimens.xml**
    - **drawbles.xml**
  - Also, you can declare resources in Java code
- Introduction to **Android User Interface** elements
  - **View**, superclass of UI classes
  - **ViewGroup**, inherits from View, represents a container which holds views
  - Android Layout classes:
    - **LinearLayut**
    - **FrameLayout**
    - **RelativeLayout**
    - **ConstraintLayout**
    - **TableLayout**
    - **ScrollView**

Review of Lecture 3

- **Simple UI controls and event handling**:
  - Declare Android controls in XML files, in **res\layout** folder
  - Use Layout editor to create the UI
  - Use the toolbars in Layout editor to arrange UI controls in the screen
  - Drag UI controls from the palette to the layout
  - Use Attributes window to set the values for attributes of UI controls

- **Simple UI controls and event handling**:
  - TextView
  - EditText
  - Button
- Use findViewById method to instantiate UI objects in Java code
  - android.widget package
  - Use onClick attribute to handle Button click events
  - Use Attributes window to **associate an event handler** method in the activity class with onClick attribute

Designing UI with Views

Objectives:

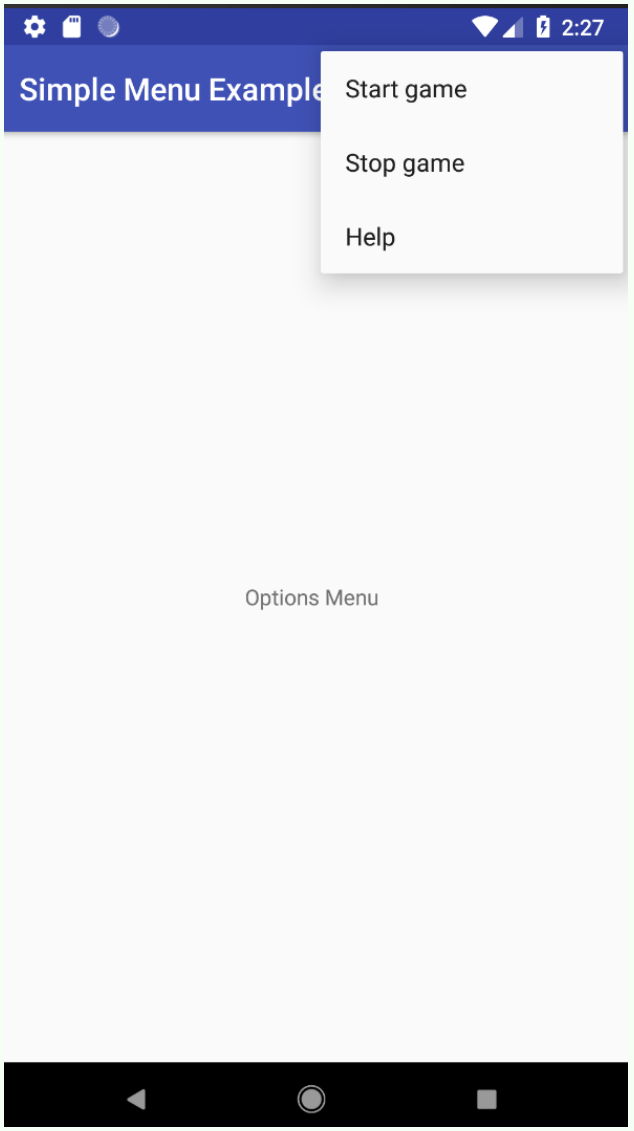
- Use Menus in Android Apps
- Use Check boxes, RadioGroup & RadioButton, ImageButton, ToggleButton, ImageButton, Spinner, Progress Indicators
  - Defining UI elements
  - Event Handling
  - Handling threads

Using Menus in Android Apps

- There are three fundamental types of menus or action presentations on all versions of Android:
  - The **options menu** is 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," "Compose email," and "Settings."
  - A **context menu** is a floating menu that appears when the user performs a long-click on an element.
  - A **popup menu** displays a list of items in a vertical list that's anchored to the view that invoked the menu
- Each menu resource is stored as a specially formatted XML files in the /res/menu directory
- Here's an example of a simple menu resource file /res/menu/game_menu.xml that defines a short menu with three items in a specific order:

Options Menu

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

    </item>

    <item
        android:id="@+id/stop"
        android:title="@string/stop">

    </item>

    <item
        android:id="@+id/help"
        android:title="@string/help">

    </item>

</menu>
```



Options Menu

- To access the preceding menu resource called /res/menu/game_menu.xml, simply override the method onCreateOptionsMenu() in your application:

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

Options Menu

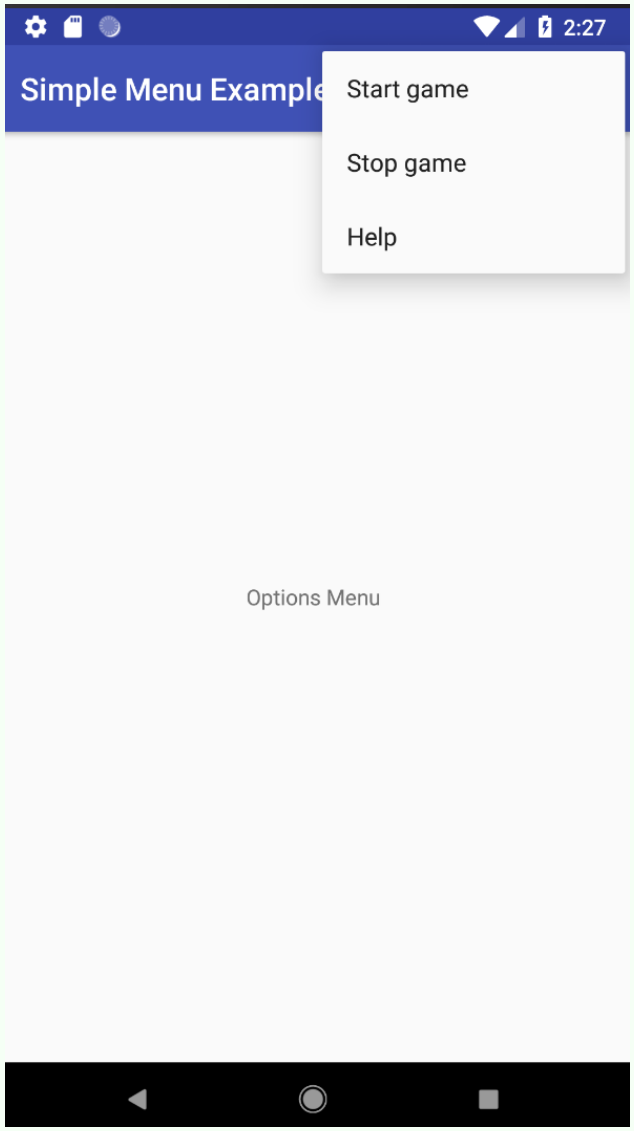
- **Handling the event** when a menu option item is selected:

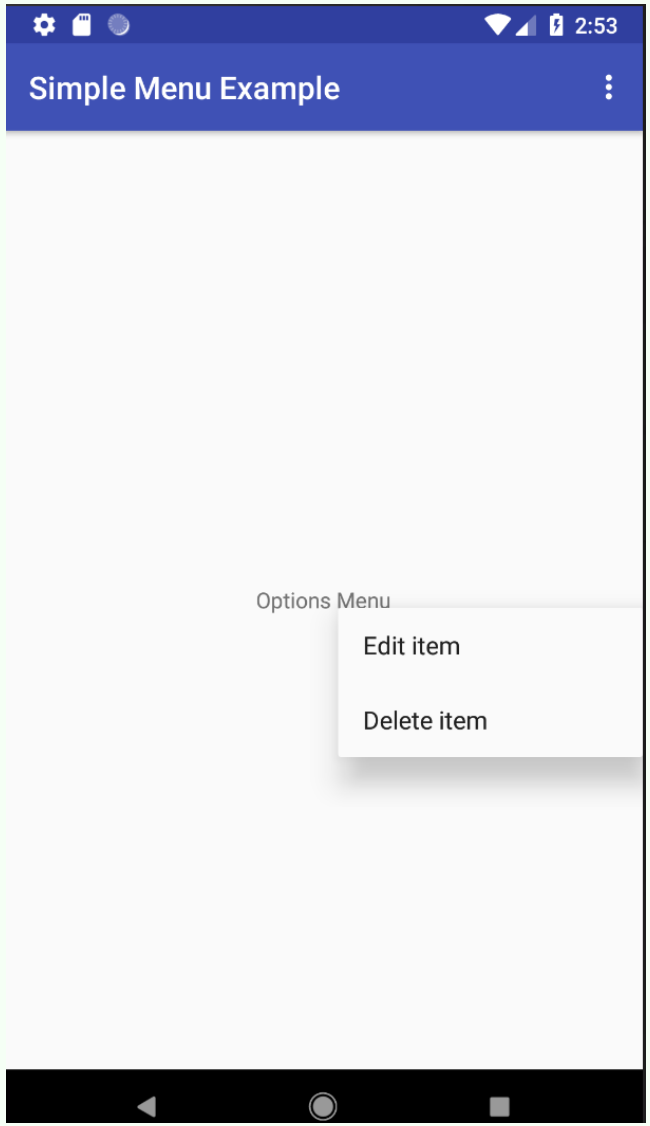
```
@Override
   public boolean onOptionsItemSelected(MenuItem item) {
      // Handle item selection
      switch (item.getItemId()) {
         case R.id.start:
            Toast.makeText(this, "You selected start!", Toast.LENGTH_LONG).show();
            break;
         case R.id.stop:
            Toast.makeText(this, "You selected stop!", Toast.LENGTH_LONG).show();
            break;
         case R.id.help:
            Toast.makeText(this, "You selected help!", Toast.LENGTH_LONG).show();
            break;
         default:
            return super.onOptionsItemSelected(item);
      }
      return true;
   }
```

Context Menu

- Register the View to which the context menu should be associated by calling registerForContextMenu() and pass it the View.
- Implement the onCreateContextMenu() method in your Activity or Fragment.
- Implement onContextItemSelected() in your activity

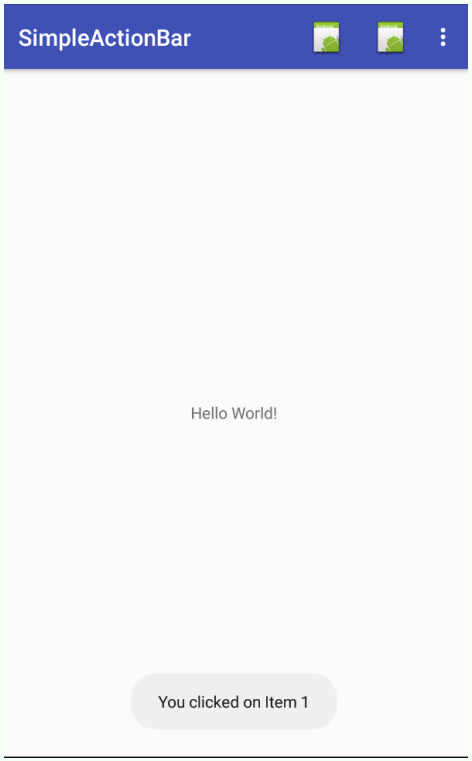
SimpleMenuExample app





Action Bar

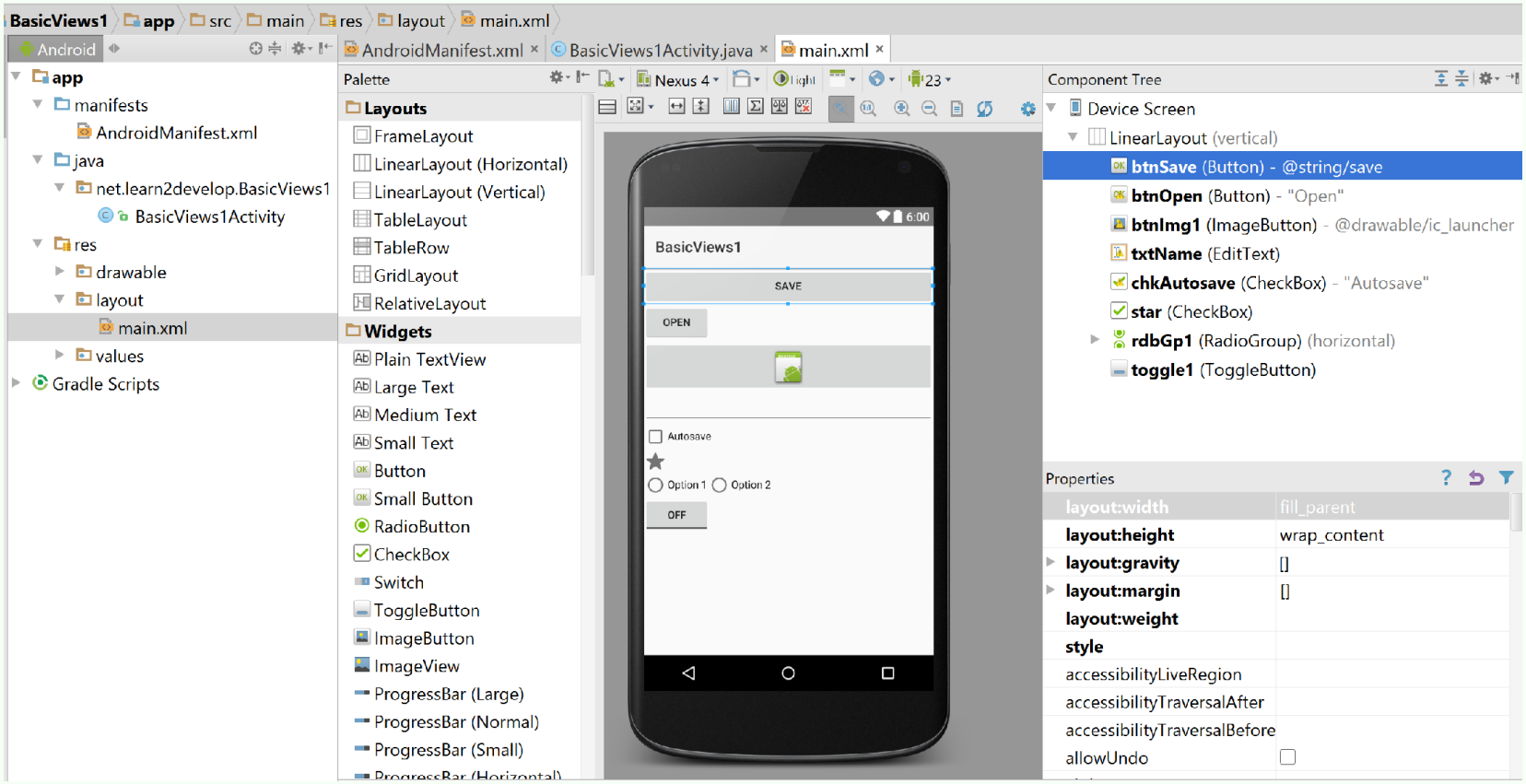
- Another newer feature introduced in Android 3 and 4 is the Action Bar.
- Located at the top of the device's screen, the Action Bar displays the application icon together with the activity title.
  - Optionally, on the right side of the Action Bar are *action items.*



- *SimpleActionBar example*

Designing UI

- Use design editor (drag and drop)
- XML definitions will be stored in layout files



Event Handling of Basic Views

1. **Create a reference to the control** using **findViewById** method:

```
Button btnOpen = (Button) findViewById(R.id.btnOpen);
```

2. **Register the control** with a proper listener:

```
btnOpen.setOnClickListener(new View.OnClickListener() {
3.  //Implement the event handler method
    public void onClick(View v) {
            DisplayToast("You have clicked the Open button");
            }
});
```

- The code above does all it's needed for a button control click event.

Using Check Boxes

- The Android check box contains a **text** attribute that appears to the side of the check box.
- This is used in a similar way to the label of a basic button.
  - In fact, it's basically a TextView next to the button.
- Here's an XML layout resource definition for a CheckBox control:

```
<CheckBox
android:id="@+id/checkbox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Check me?" />
```

Event handling of Check Boxes

```
//1- create the check box reference
CheckBox checkBox = (CheckBox) findViewById(R.id.chkAutosave);
//2- register the checkbox reference with a click listener
        checkBox.setOnClickListener(new View.OnClickListener()
        {
//3- implement the event handler method
                public void onClick(View v) {
                        if (((CheckBox)v).isChecked())
                            DisplayToast("CheckBox is checked");
                        else
                            DisplayToast("CheckBox is unchecked");
                }
        });
```

Using Check Boxes

- The following example checks the state of the button programmatically and changes the text label to reflect the change:

```
final CheckBox check_button = (CheckBox)
    findViewById(R.id.checkbox);
check_button.setOnClickListener(new View.OnClickListener()
{
public void onClick (View v) {
 TextView tv = (TextView)findViewById(R.id.checkbox);
 tv.setText(check_button.isChecked() ? “This option is
    checked” : “This option is not checked”);
 }
});
```

Using **RadioGroups and RadioButtons**

- The RadioButton controls are similar to CheckBox controls.
- They have a **text label** next to them, set via the text attribute, and they have a **state** (checked or unchecked)
- However, you should group **RadioButton** objects inside a **RadioGroup** that handles enforcing their combined states so that **only one RadioButton can be checked at a time**.

Using **RadioGroups and RadioButtons**

- The XML layout resource definition below shows a RadioGroup containing four RadioButton objects

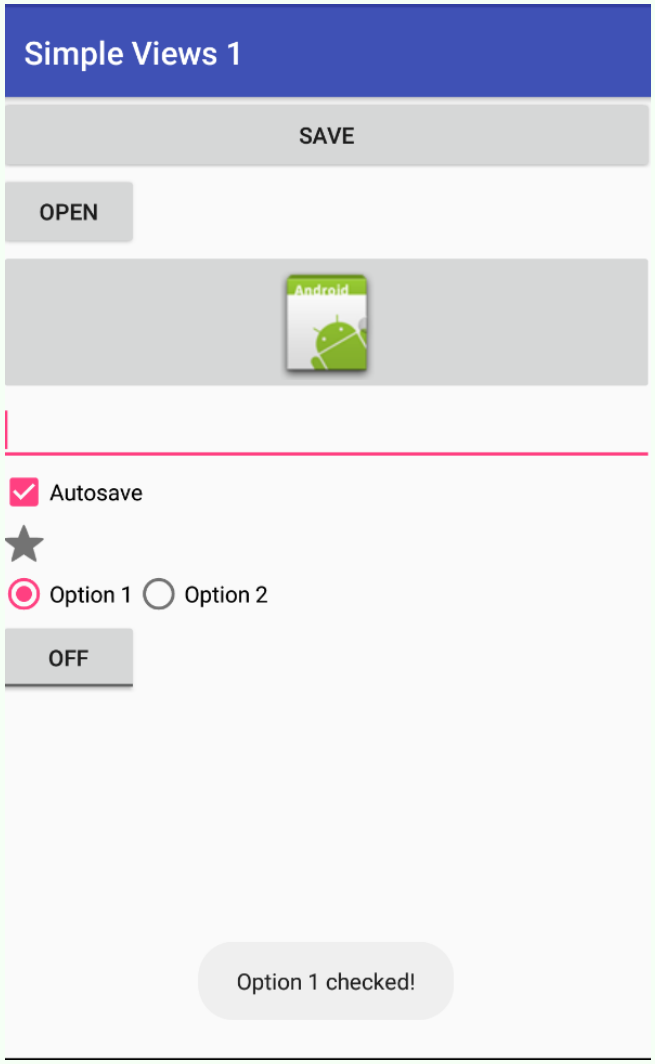
```
<RadioGroup android:id="@+id/RadioGroup01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <RadioButton
    android:id="@+id/RadioButton01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 1">
    </RadioButton>
```

Using **RadioGroups and RadioButtons**

```
    <RadioButton android:id="@+id/RadioButton02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 2"></RadioButton>
    <RadioButton android:id="@+id/RadioButton03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 3"></RadioButton>
    <RadioButton android:id="@+id/RadioButton04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 4"></RadioButton>
</RadioGroup>
```

Using **RadioGroups and RadioButtons**

- You handle actions on these RadioButton objects through the RadioGroup object.
- The following example shows event handling of RadioButton objects for a click event.



Event handling of Radio buttons

```
//1- create the radio group reference
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.rdbGp1);


//2- register the radio group reference with a click listener
radioGroup.setOnCheckedChangeListener(new OnCheckedChangeListener()
{
//3- implement the event handler method
        public void onCheckedChanged(RadioGroup group, int checkedId)
        {
                RadioButton rb1 = (RadioButton) findViewById(R.id.rdb1);
                        if (rb1.isChecked()) {
                                DisplayToast("Option 1 checked!");
                        } else {
                                DisplayToast("Option 2 checked!");
                        }
        }
});
```

Using **RadioGroups and RadioButtons**

- The following example checks for the state of the radio button and changes the text label to reflect the change:

```
final RadioGroup group =
    (RadioGroup)findViewById(R.id.RadioGroup01);
final TextView tv = (TextView) findViewById(R.id.TextView01);
group.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
public void onCheckedChanged( RadioGroup group, int checkedId) {
if (checkedId != -1) {
RadioButton rb = (RadioButton) findViewById(checkedId);
if (rb != null) { tv.setText("You chose: " + rb.getText());}
}
else { tv.setText("Choose 1");}
}
});
```

Using **RadioGroups and RadioButtons**

- The entire RadioGroup can be cleared so that none of the RadioButton objects are selected.
- The following example demonstrates how to do this in response to a button click outside of the RadioGroup:

```
final Button clear_choice = (Button)
    findViewById(R.id.Button01);
clear_choice.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) { RadioGroup group =
        (RadioGroup) findViewById(R.id.RadioGroup01);
          if (group != null) {
          group.clearCheck();
          }
      }
}
```

Toggle buttons

- A **Toggle** Button is similar to a check box in behavior but is usually used to show or alter the **on** or **off state** of something.
- Like the CheckBox, it **has a state** (checked or not).
- Unlike the CheckBox, it does not show text next to it. Instead, it **has two text fields**.
  - The first attribute is **textOn**, which is the text that displays on the button when its checked state is on.
  - The second attribute is **textOff**, which is the text that displays on the button when its checked state is off.

Toggle buttons

- The following layout code shows a definition for a toggle button that shows “Enabled” or “Disabled” based on the state of the button:

```
<ToggleButton
android:id=”@+id/toggle_button”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:text=”Toggle”
android:textOff=”Disabled”
android:textOn=”Enabled” />
```

Event handling of Toggle buttons

```
//1- create the toggle button reference
ToggleButton toggleButton = (ToggleButton) findViewById(R.id.toggle1);

//2- register the radio group reference with a click listener
        toggleButton.setOnClickListener(new View.OnClickListener()
        {
//3- implement the event handler method
                public void onClick(View v) {
                        if (((ToggleButton)v).isChecked())
                                DisplayToast("Toggle button is On");
                        else
                                DisplayToast("Toggle button is Off");
                }
        });
```

Validating Input

- Using attributes of UI controls:
  - **maxLength**

```
<EditText
  android:id="@+id/editText1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:maxLength="5“ />
```

- Using **Input Filters** programmatically:
  - Here is an example of an EditText control with two **built-in filters** that might be appropriate for a two-letter state abbreviation:

```
final EditText text_filtered = (EditText)
   findViewById(R.id.input_filtered);
text_filtered.setFilters(new InputFilter[] {
new InputFilter.AllCaps(), new InputFilter.LengthFilter(2)
});
```

Using **Spinner Controls**

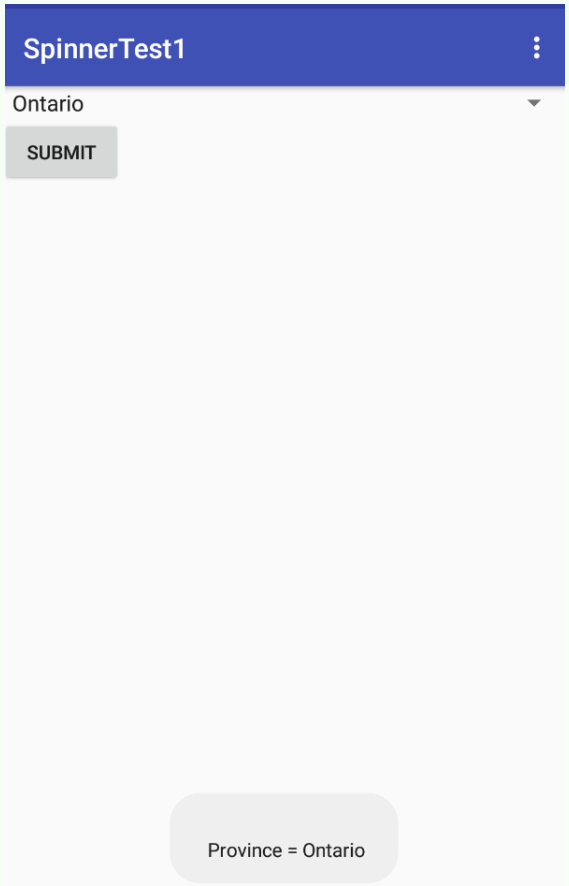
- Limit the choices available for users to type:
  - set the available choices in the layout definition by using **the entries attribute** with an array resource (specifically a string-array that is referenced as something such as @array/state/province-list):
- Here is an example of the XML layout definition for a Spinner control for choosing a color:

```
<Spinner
android:id="@+id/Spinner01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:entries="@array/colors"
android:prompt="@string/spin_prompt" />
```

Spinner Control example

- Retrieve the selected View and extract the text directly:

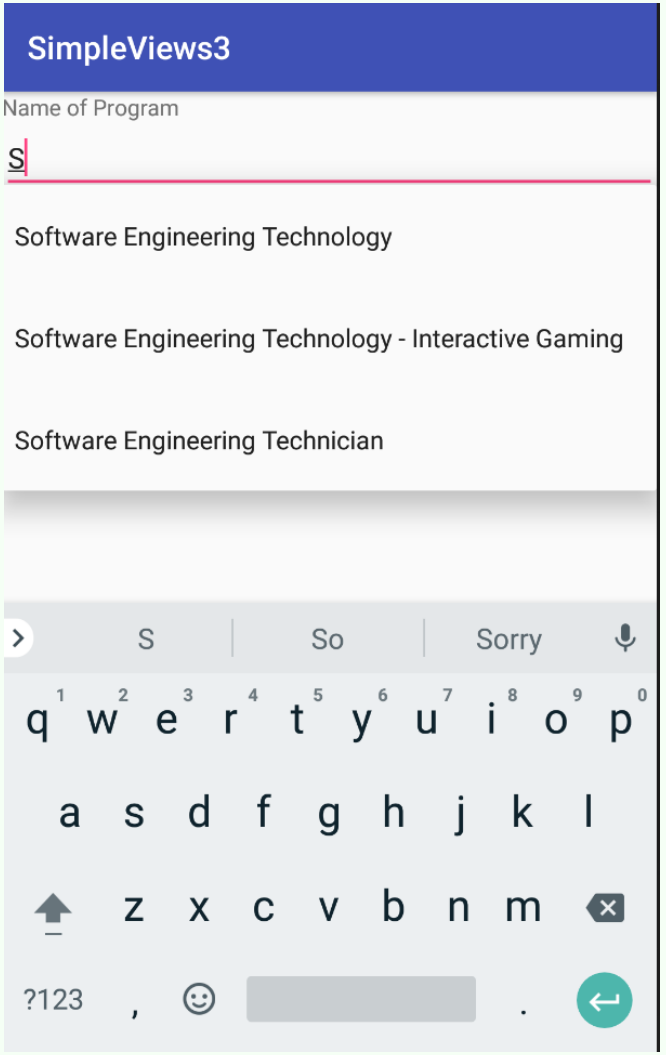
```
final Spinner spin = (Spinner)
    findViewById(R.id.provinces_spinner);
final Button submit =
    (Button)findViewById(R.id.submit);
    submit.setOnClickListener(new
    OnClickListener() {
    public void onClick(View v) {
            TextView text_sel =
    (TextView)spin.getSelectedView();
            Toast.makeText(MainActivity.this, "\n
    spinner = "+text_sel.getText(),
    Toast.LENGTH_SHORT).show();
                    }
            });
```



Filtering choices with a **spinner** control

AutoCompleteTextView View

- The **AutoCompleteTextView** is a view that is similar to EditText (in fact it is a subclass of EditText), except that it **shows a list of completion suggestions automatically while the user is typing**.



AutoCompleteTextView View

- An **ArrayAdapter** object manages the array of strings that will be displayed by the **AutoCompleteTextView**.

String[] **programs** = { "Software Engineering Technology",
"Interactive Gaming", "Health Informatics Technology",
"Mobile Apps Development", "Software Engineering Technician“ };

- You set the AutoCompleteTextView to display in the ***simple_dropdown_item_1line*** mode:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line, programs);
AutoCompleteTextView textView = (AutoCompleteTextView)
        findViewById(R.id.txtPrograms);
textView.setThreshold(2);
textView.setAdapter(adapter);
```

Using ImageButton

- An **ImageButton** is, for most purposes, almost exactly like a basic button.
  - Click actions are handled in the same way.
- The primary difference is that you can set its **src attribute to be an image**.
- Here is an example of an ImageButton definition in an XML layout resource file:

```
<ImageButton
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:id=”@+id/image_button”
android:src=”@drawable/droid” />
```

Progress Indicators

- The Android SDK provides several types of progress bars.
- The **standard progress** bar is a **circular** indicator that only animates.
  - It only shows that something is taking place
  - There **are three sizes** of this type of progress indicator
- A basic **indeterminate** progress bar:

```
<ProgressBar
android:id="@+id/progress_bar"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

- The **default style is for a medium-size circular progress indicator**

Progress Indicators

- Two other styles for indeterminate progress bar are **progressBarStyleLarge** and **progressBarStyleSmall**.
  - This style animates automatically
  - When the value reaches the maximum value, the indicators fade away so that they aren't visible.
- The following code demonstrates how to place this type of indeterminate progress indicator on your Activity screen:

```
requestWindowFeature(Window.FEATURE_INDETERMINATE
    _PROGRESS);
requestWindowFeature(Window.FEATURE_PROGRESS);
setContentView(R.layout.indicators);
setProgressBarIndeterminateVisibility(true);
setProgressBarVisibility(true);
setProgress(5000);
```

Progress Indicators

- We can set the indicator progress status programmatically as follows:

```
Progress = (ProgressBar) findViewById(R.id.progress_bar);
Progress.setProgress(75);
```

  - Setting the progress to 75 shows the indicator at 75 percent complete

Progress Indicators

- The second type is a **horizontal progress** bar that **shows the completeness** of an action.
  - For example, you can see how much of a file is downloading.
  - This horizontal progress bar can also have a secondary progress indicator on it
- This example shows the layout definition for a horizontal progress indicator:

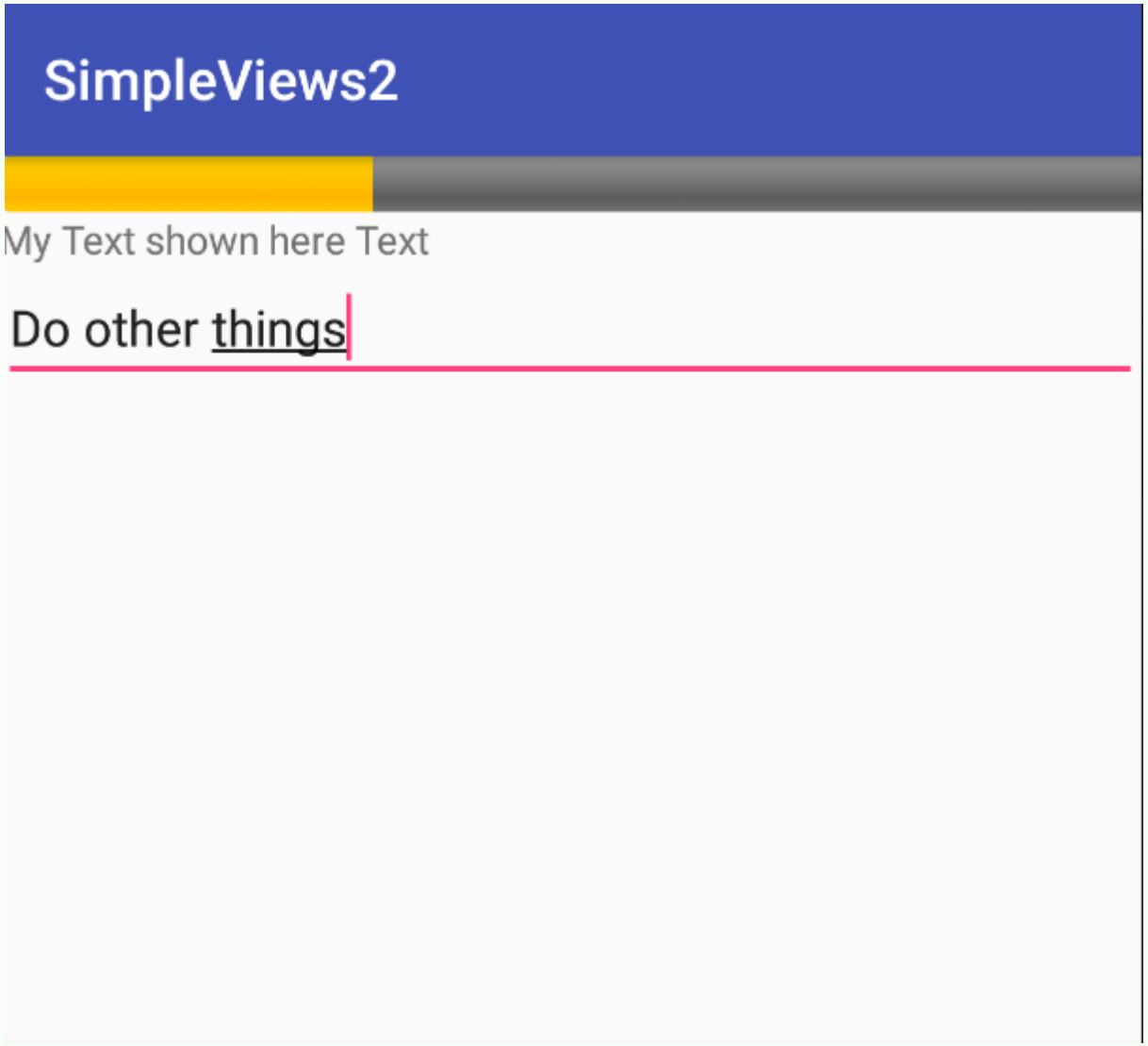
```
<ProgressBar android:id="@+id/progressbar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Horizontal" />
```

Android UI thread

- Android's main thread launched by the system is called UI thread. Android requires:
  - **Do not block the UI thread** – create other threads to do some time consuming work
  - **Do not access the Android UI toolkit from outside the UI thread** – to communicate with the UI thread from your new thread, just post a message to the Handler created on the UI thread:

```
while (progressStatus < 500)
{
        progressStatus = doSomeWork();
        //Update the progress bar
        handler.post(new Runnable()
        {
            public void run() {
                progressBar.setProgress(progressStatus);
            }
        });
}
```

SimpleVews2 example



Adjusting Progress with **SeekBar**

- **SeekBar looks** like the regular horizontal progress bar, but **includes a thumb**, or selector, that can be dragged by the user.
  - A default thumb selector is provided, but you can **use any drawable item as a thumb**.

```
<SeekBar
android:id="@+id/seekbar1"
android:layout_height="wrap_content"
android:layout_width="240px"
android:max="500" />
```

Adjusting Progress with **SeekBar**

- To show the user what exact value the user is selecting
  - Just provide an implementation of the onProgressChanged() method:

```
SeekBar seek = (SeekBar) findViewById(R.id.seekbar1);
seek.setOnSeekBarChangeListener(
new SeekBar.OnSeekBarChangeListener()
{ //start anonymous class
      public void onProgressChanged(
      SeekBar seekBar, int progress, boolean fromTouch)
      {
       ((TextView)findViewById(R.id.seek_text)).setText("Value:
         "+progress);

         seekBar.setSecondaryProgress((progress+seekBar.getMax())
         /2);
      }
});
```

Displaying Rating Data with **RatingBar**

- **RatingBar** has a more specific purpose: showing ratings or getting a rating from a user:
- Here's an example of an XML layout resource definition for a RatingBar with four stars:

```
<RatingBar android:id="@+id/ratebar1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:numStars="4"
android:stepSize="0.25" />
```

- Here, users can choose any rating value between 0 and 4.0, but only in increments of 0.25, the **stepSize** value

Displaying Rating Data with **RatingBar**

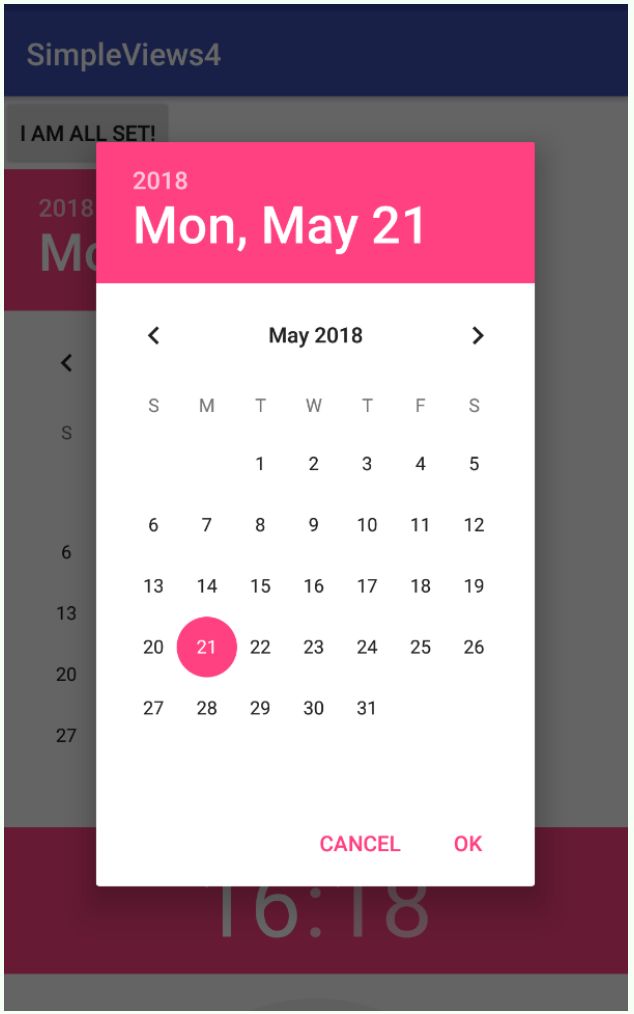
- To show a numeric representation of this value to the user implement onRatingChanged() method of the RatingBar.OnRatingBarChangeListener class.

```
RatingBar rate = (RatingBar) findViewById(R.id.ratebar1);
rate.setOnRatingBarChangeListener(new
RatingBar.OnRatingBarChangeListener()
{
      public void onRatingChanged(RatingBar ratingBar,
      float rating, boolean fromTouch)
      {
      ((TextView)findViewById(R.id.rating_text)).setText("Rating: "+
         rating);
      }
});
```

Getting Dates and Times from Users

- DatePicker control can be used to get a month, day, and year from the user
- The basic XML layout resource definition for a DatePicker follows:

```
<DatePicker
android:id="@+id/DatePicker01"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```



Getting Dates and Times from Users

- Your code can register to receive a method call when the date changes.
- You do this by implementing the onDateChanged() method.

```
final DatePicker date = (DatePicker)findViewById(R.id.DatePicker01);
date.init(date.getYear(), date.getMonth(), date.getDayOfMonth(),
new DatePicker.OnDateChangedListener() {
public void onDateChanged(DatePicker view, int year,
int monthOfYear, int dayOfMonth) { Date dt = new Date(year-1900,
    monthOfYear, dayOfMonth, time.getCurrentHour(),
time.getCurrentMinute());
text.setText(dt.toString());
}
});
```

Getting Dates and Times from Users

- The preceding code sets the DatePicker.OnDateChangedListener by a call to the DatePicker.init() method.
- A DatePicker control is initialized with the current date.
- A TextView is set with the date value that the user entered into the DatePicker control.
- The value of 1900 is subtracted from the year parameter **to make it compatible with the java.util.Date** class.

Getting Dates and Times from Users

- A **TimePicker** control is similar to the DatePicker control.
  - It also doesn't have any unique attributes.
  - To register for a method call when the values change, you call setOnTimeChangedListener() method:

```
time.setOnTimeChangedListener(new
    TimePicker.OnTimeChangedListener() {
public void onTimeChanged(TimePicker view,
int hourOfDay, int minute) {
Date dt = new Date(date.getYear()-1900, date.getMonth(),
date.getDayOfMonth(), hourOfDay, minute);
text.setText(dt.toString());
}
});
```

- Sets a TextView to a string displaying the time value that the user entered

Working with Styles

- Android user interface designers can **group layout element attributes together** in styles
- Styles are tagged with the <style> tag and should be stored in the /res/values/ directory.
- Style resources are defined in XML and compiled into the application binary at build time
- Styles in Android share a similar philosophy to cascading stylesheets in web design—they allow you to **separate the design from the content**.

Working with Styles

- For example, by using a style, you can take this layout XML:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

And turn it into this:

```
<TextView
    style = ”@style/mandatory_text_field_style”
    android:text="@string/hello" />
```

Working with Styles

- Here's an example of a simple style resource file /res/values/styles.xml containing two styles:
  - one for mandatory form fields, and one for optional form fields on TextView and EditText objects:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="mandatory_text_field_style">
    <item name="android:textColor">#000000</item>
    <item name="android:textSize">14pt</item>
    <item name="android:textStyle">bold</item>
    </style>
    <style name="optional_text_field_style">
    <item name="android:textColor">#0F0F0F</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textStyle">italic</item>
    </style>
</resources>
```

Working with Styles

- Here's the **styles.xml** file again; this time, the color and text size fields are available in the other resource files: colors.xml and dimens.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="mandatory_text_field_style">
    <item name="android:textColor">@color/mand_text_color</item>
    <item name="android:textSize">@dimen/important_text</item>
    <item name="android:textStyle">bold</item>
    </style>
    <style name="optional_text_field_style">
    <item name="android:textColor">@color/opt_text_color</item>
    <item name="android:textSize">@dimen/unimportant_text</item>
    <item name="android:textStyle">italic</item>
    </style>
</resources>
```

Working with Styles

- You can set each control's style attribute by referencing it as:

  style="@style/name_of_style"

  For example:

```
<TextView
android:id="@+id/TextView01"
style="@style/mandatory_text_field_style"
android:layout_height="wrap_content"
android:text="@string/mand_label"
android:layout_width="wrap_content" />
```

Working with Themes

- **Themes** are like styles, but instead of being applied to one layout element at a time, they **are applied to all elements of a given activity**.
- Themes are defined in exactly the same way as styles.
  - Themes use the <style> tag and should be stored in the /res/values directory.
- The only difference is that instead of applying that named style to a layout element, you define it as the **theme attribute of an activity in the AndroidManifest.xml** file

Working with Themes

- To set a theme for all the activities of your application, open the AndroidManifest.xml file and edit the <application> tag to include the android:theme attribute with the style name. For example:

```
<application android:theme="@style/CustomTheme">
```

- To apply a theme to just one Activity in your application, add the android:theme attribute to the <activity> tag instead:

```
<activity android:theme="@android:style/CustomTheme">
```

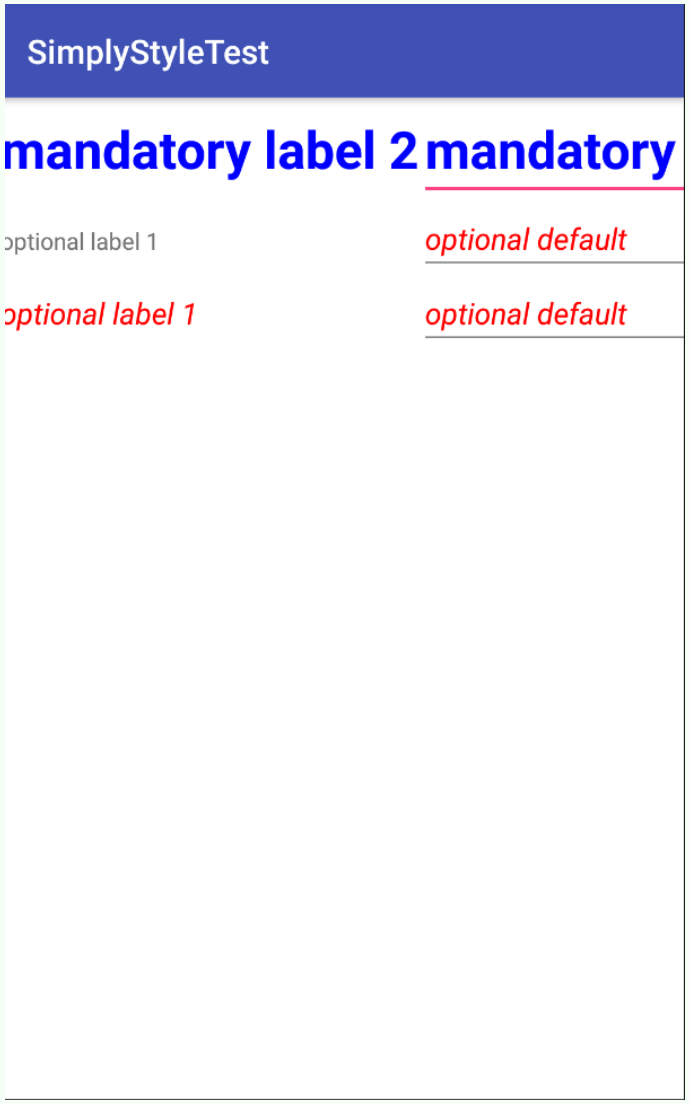
- You can inherit built-in themes. For example:

```
<activity android:theme="@android:style/Theme.Dialog">
```

will make your activity to look like a dialog box

Styles Example

- SimplyStylesTest app



References

- Textbook
- Android Documentation
- Lauren Darcey, Shane Conder: Introduction to Android Application Development: Android Essentials (5th Edition)