Android UI Controls

Jigisha Patel

Agenda

- View
- XML Layout file
- findViewById()
- UI Controls
    - TextView
    - EditText
    - Button
    - RadioButton

Android App UI

- The **/res/layout** folder contains the XML files that represents the app screens.
- Each XML layout file is related to respective source file.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity">
```

activity_main.xml

MainActivity.java

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
    }
```

View

- This class represents the basic building block for user interface components.
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.

- View is the base class for widgets (UI controls), which are used to create interactive UI components (buttons, text fields, etc.).

XML Layout Files

- Declaring your UI in XML allows you to separate the presentation of your app from the code that controls its behavior.
- Using XML files also makes it easy to provide different layouts for different screen sizes and orientations.

- XML layout files *usually* consist of **a layout manager** and **UI controls**.
- Using Android's straightforward XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

XML Layout Files cont…

- Each layout file must contain exactly one root element, which must be a View or ViewGroup object.

- Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.

Load the XML Resource

- When you compile your app, each XML layout file is compiled into a View resource.
- You should load the layout resource from your app code, in your Activity.onCreate() callback implementation.
- You can do it by calling setContentView(), passing it the reference to your layout resource in the form of R.layout.layout_file_name.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

Attributes

- Every UI control such as Button and TextView supports its own set of XML attributes in addition to the ones inherited from the root View class.

- The UI Controls also have attributes are considered layout parameters which are attributes that describe certain layout orientations of the View object.

ID

- Any View object may have an integer ID associated with it, to uniquely identify the View within the tree.
- When the app is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute.

```
android:id="@+id/my_button"
```

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources

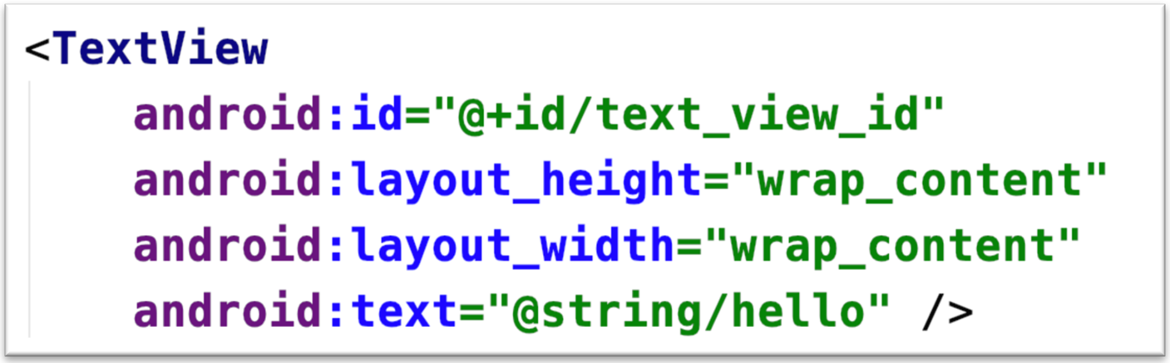
findViewById()

- To retrieve the UI Control and interact it with programmatically to support event handling or changing its attributes dynamically you can use findViewById(int).

- findViewById() method requires an integer id of the UI control to be provided as a parameter to the method which can be accessed using R.
- R represents the resources available in the app such as layout, images, colors, strings and UI controls.

TextView

- It is a user interface element that displays read-only text to the user.

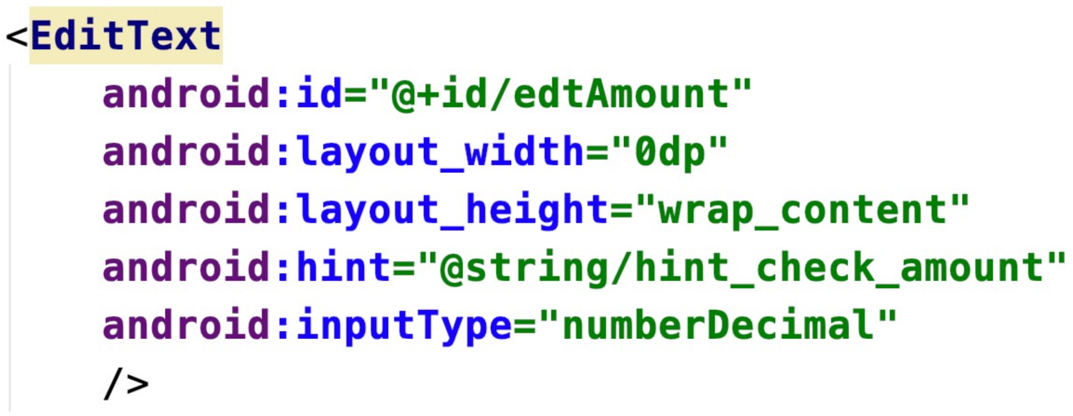


```
<TextView
    android:id="@+id/text_view_id"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/hello" />
```

- You can set or retrieve the value to/from the TextView using the text property

EditText

- It is a user interface element for entering and modifying text.
- When you define an EditText widget, you must specify the inputType attribute.
- Choosing the input type configures the keyboard type that is shown, acceptable characters, and appearance of the edit text.



```xml
<EditText
    android:id="@+id/edtAmount"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="@string/hint_check_amount"
    android:inputType="numberDecimal"
    />
```

Button

- A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it or taps on it.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

- Responding to the button clicks can be managed either by using onClick attribute in XML of the button or by using OnClickListener interface.

Button event handling – onClick attribute

- To define the click event handler for a button, add the android:onClick attribute to the <Button> element in your XML layout.
- The value for this attribute must be the name of the method you want to call in response to a click event.
- The Activity hosting the layout must then implement the corresponding method.

- The method you declare in the android:onClick attribute must be public, return void and define a View as its only parameter.

Button event handling – OnClickListener

- You can also declare the click event handler programmatically rather than in an XML layout.

- This might be necessary if you instantiate the Button at runtime or you need to declare the click behavior in a Fragment subclass.

- To declare the event handler programmatically, create a View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener)

Radio Button

- Radio buttons allow the user to select one option from a set.
- You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side.

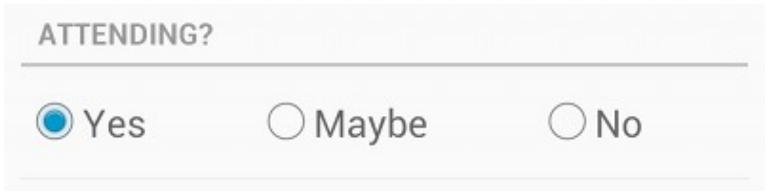


Image Source: https://developer.android.com/guide/topics/ui/controls/radiobutton

- Because radio buttons are mutually exclusive, you must group them together inside a RadioGroup.
- To create each radio button option, create a RadioButton within RadioGroup.

RadioButton example

```
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

Responding to RadioButton selection

- A radio button is a two-states button that can be either checked or unchecked.

- Similar to Button, to define the click event handler for a button, you can add the android:onClick attribute to the <RadioButton> element in your XML layout.

- Optionally, you can also use checkedRadioButtonId() method to retrieve an identifier of the selected radio button in this group.

References

- https://developer.android.com/guide/topics/ui/controls/radiobutton
- https://developer.android.com/guide/topics/ui/controls/button
- https://developer.android.com/guide/topics/ui
- https://developer.android.com/reference/android/view/View