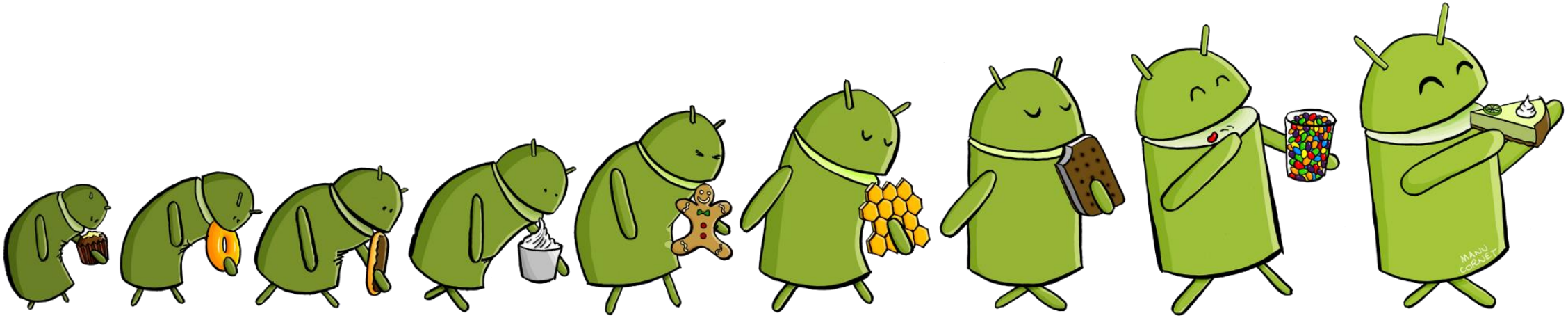


In the name of Allah

Widgets (part one)

Lecture #06

Subject: Mobile App Development
Instructor: Ehsan Hasin {ehsanhasin@gmail.com}
Date: Tuesday, April 16, 2024

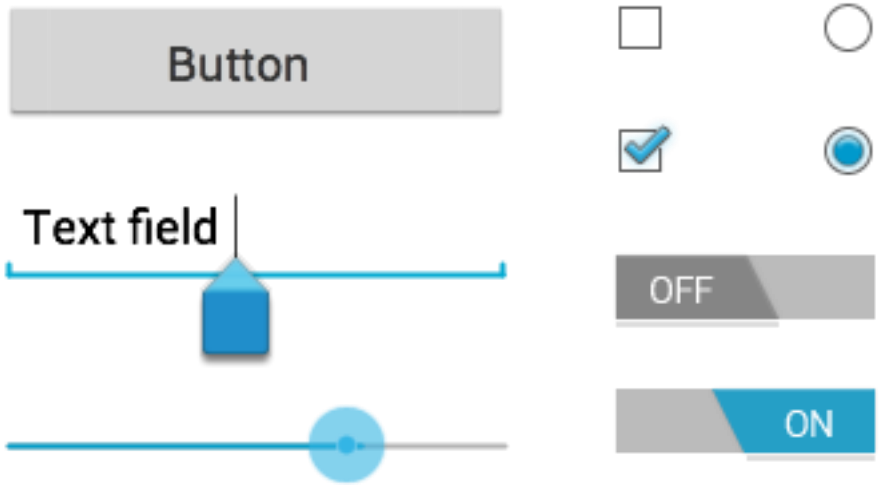
Table of Content

Introduction

- The basic building block for user interface is a **View** object
- which is created from the **View class**
- occupies a **rectangular area** on the screen and is **responsible** for drawing and event handling.
- The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView.
- View is the **base class** for widgets, which are used to create interactive UI components like buttons, text fields, etc.

Android UI Controls/View/Widget

- There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.
  - TextView
    - This control is used to display text to the user.
  - EditText
    - EditText is a predefined subclass of TextView that includes rich editing capabilities.
  - Button
    - A push-button that can be pressed, or clicked, by the user to perform an action.
  - ToggleButton
    - An on/off button with a light indicator
  - DatePicker
    - The DatePicker view enables users to select a date of the day.



TextView

- A user interface element that displays text to the user.

  The following code sample shows a typical use, with an XML layout and code to modify the contents of the text view:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
  <TextView
     android:id="@+id/text_view_id"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/hello" />
</LinearLayout>
```

TextView – Attributes

- These are the important attributes related to TextView control.

No.	Attribute & Description
1	android:**id** This is the ID which uniquely identifies the control.
2	android:**text** Text to display.
3	android:**textColor** Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
4	android**:textSize** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp)

TextView – Common Attributes

- These are the important attributes that are common to all views.

No.	Attribute & Description
1	android:**id** This is the ID which uniquely identifies the view.
2	android:**layout_width** This is the width of the layout.
3	android:**layout_height** This is the height of the layout
4	android:**layout_margin** This is the extra space on the each side of the layout.
5	android:**padding** This is the padding filled for the layout.

TextView

This code sample demonstrates how to modify the contents of the text view (with **Java**) defined in the previous XML layout:

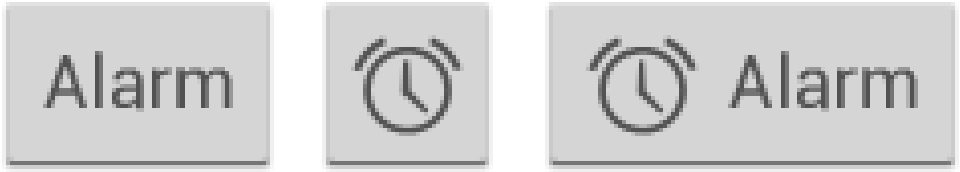
```
public class MainActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      TextView helloTextView = findViewById(R.id.text_view_id);
      helloTextView.setText("This is text of TextView");
   }
}
```

Buttons

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



Buttons - Creating a button in XML

- Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

With text, using the Button class

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

Buttons - Creating a button in XML

With an icon, using the ImageButton class:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    android:contentDescription="@string/button_icon_desc"
    ... />
```

With text and an icon, using the Button class with the android:drawableLeft attribute:

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

Buttons – Attributes

- These are the important attributes related to Button control.
- Inherited from **android.widget.TextView** Class.

No.	Attribute & Description
1	android:**id** This supplies an identifier name for this view.
2	android:**drawableBottom** This is the drawable to be drawn below the text.
3	android:**drawableRight** This is the drawable to be drawn to the right of the text.
4	android:**text** This is the Text to display.

Buttons – Attributes

- These are the important attributes related to Button control.
- Inherited from **android.view.View** Class.

No.	Attribute & Description
1	android:**background** This is a drawable to use as the background.
3	android:**id** This supplies an identifier name for this view.
3	android:**onClick** This is the name of the method in this View's context to invoke when the view is clicked.
4	android:**visibility** This controls the initial visibility of the view.

Buttons - Responding to Click Events

- When the user clicks a button, the Button object receives an on-click event.
- 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.

Buttons - Responding to Click Events

For example, here's a layout with a button using **android:onClick**:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Within the Activity that hosts this layout, the following method handles the click event:

```
/** Called when the user touches the button */
public void sendMessage(View view) {
   // Do something in response to button click
}
```

Buttons - Responding to Click Events

The method you declare in the android:onClick attribute must have a signature exactly as shown in the previous slide. Specifically, the method must:

- Be **public**
- Return **void**
- Define a **View** as its only parameter (this will be the View that was clicked)

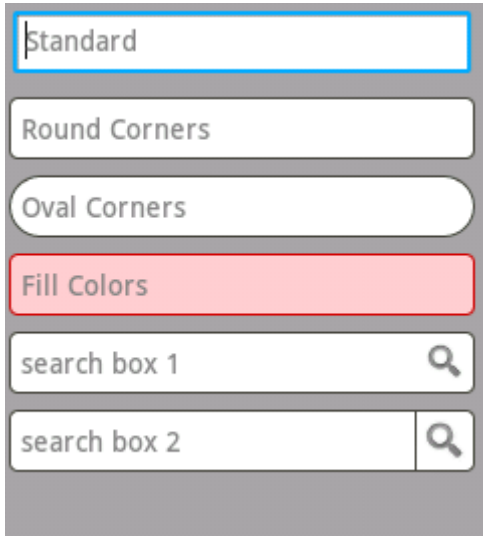
Using an OnClickListener

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 an View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener). For example:

```
Button button = findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

EditText

- A user interface element for entering and modifying text.
- When you define an edit text widget, you must specify the **inputType** attribute. For example, for plain text input set inputType to "**text**":




```
<EditText
   android:id="@+id/plain_text_input"
   android:layout_height="wrap_content"
   android:layout_width="match_parent"
   android:inputType="text"/>
```

EditText – Attributes

- These are the important attributes related to TextView control.

No.	Attribute & Description
1	android:**id** This is the ID which uniquely identifies the control.
2	android:**editable** If set to true, specifies that this TextView has an input method.
3	android:**fontFamily** Font family (named by string) for the text.
4	android:**gravity** Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
5	android:**hint** Hint text to display when the text is empty.

EditText – Attributes

- These are the important attributes related to TextView control.

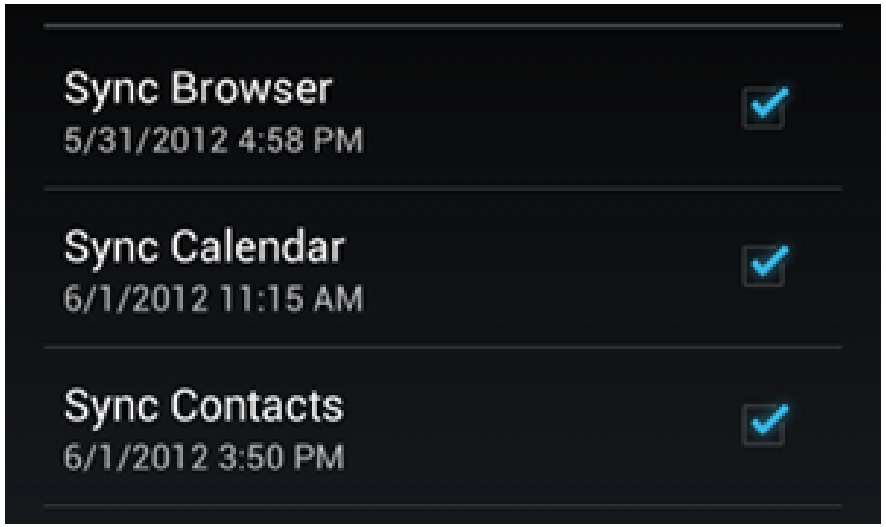
No.	Attribute & Description
6	android:**inputType** The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
7	android:**text** Text to display.
8	android:**textColor** Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
9	android:**textColorHint** Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
10	android**:textSize** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp)

EditText

- Choosing the **input type** configures the **keyboard type** that is shown, acceptable characters, and appearance of the edit text.
- **For example**, if you want to accept a secret number, like a unique pin or serial number, you can set inputType to "**numericPassword**".
- An inputType of "numericPassword" results in an edit text that accepts numbers only, shows a numeric keyboard when focused, and masks the text that is entered for privacy.

Checkboxes

- Checkboxes allow the user to select one or more options from a set.
- Typically, you should present each checkbox option in a vertical list.



Checkboxes

- To **create** each checkbox option, create a CheckBox in your layout.
- Because a set of checkbox options allows the user to select multiple items, each checkbox is **managed separately** and you must **register a click listener** for each one.

```
<?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">
   <CheckBox android:id="@+id/checkbox_meat"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/meat"
      android:onClick="onCheckboxClicked"/>
   <CheckBox android:id="@+id/checkbox_cheese"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cheese"
      android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Checkboxes - Responding to Click Events

- When the user selects a checkbox, the CheckBox object receives an on-click event.
- To define the click event handler for a checkbox
  - add the **android:onClick** attribute to the **<CheckBox>** 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.

Checkboxes - Responding to Click Events

For **example**, here are a couple CheckBox objects in a list:

```
<?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">
   <CheckBox android:id="@+id/checkbox_meat"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/meat"
      android:onClick="onCheckboxClicked"/>
   <CheckBox android:id="@+id/checkbox_cheese"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cheese"
      android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

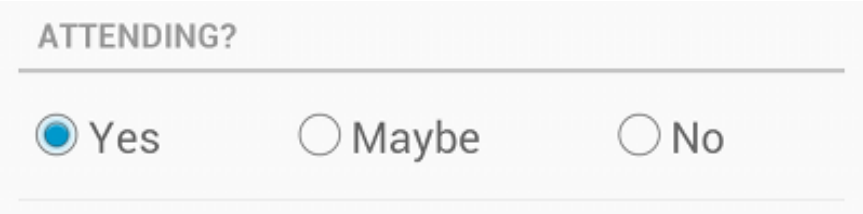
Checkboxes - Responding to Click Events

Within the **Activity** that hosts this layout, the following method handles the click event for both checkboxes:

```
public void onCheckboxClicked(View view) {
    switch(view.getId()) {
      case R.id.checkbox_meat:
            break;
      case R.id.checkbox_cheese:
            break;
 } }
```

Radio Buttons

- Radio buttons allow the user to select one option from a set.



Radio Buttons – Creation

- To create each radio button option, create a **RadioButton** in your layout.
- However, because radio buttons are mutually exclusive, you must group them together inside a **RadioGroup**.
- By grouping them together, the system ensures that only one radio button can be selected at a time.

```
<RadioGroup 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>
```

Radio Buttons – Responding to Click Events

- When the user selects one of the radio buttons, the corresponding RadioButton object receives an on-click event.
  - To define the click event handler for a button, add the **android:onClick** attribute to the **<RadioButton>** 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.

Radio Buttons – Responding to Click Events

For **example**, here are a couple RadioButton objects:

```
<RadioGroup 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>
```

Radio Buttons – Responding to Click Events

Within the **Activity** that hosts this layout, the following method handles the click event for both radio buttons:

```
public void onRadioButtonClicked(View view) {
    switch(view.getId()) {
      case R.id.radio_pirates:
            //the code…
            break;
      case R.id.radio_ninjas:
            //the code…
            break;
    }
}
```

Summary

- The **View** objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView.
- **TextView**
  - A user interface element that displays text to the user.
- **Button**
  - represents a push-button and communicates what action occurs when the user touches it.
- **EditText**
  - A user interface element for entering and modifying text.
- **Checkboxes**
  - Checkboxes allow the user to select one or more options from a set.
- **Radio Buttons**
  - Radio buttons allow the user to select one option from a set.

The End

Thank You