Basic User Interface Screen elements

Learning Objective

After studying this module, you will be able to learn

- The user interface elements available within the Android Software Development Kit (SDK).
- Uses of various user interface elements
- How to use a variety of different components and controls to build a screen
- How your application can listen for various actions performed by the user.

Introduction

Most Android applications inevitably need some form of user interface. In this module, we will discuss the user interface elements available within the Android Software Development Kit (SDK). Some of these elements display information to the user, whereas others gather information from the user.

You learn how to use a variety of different components and controls to build a screen and how your application can listen for various actions performed by the user. Finally, you learn how to style controls and apply themes to entire screens.

Introduction to Views, Controls and Layout

Before we go any further, we need to define a few terms. This gives you a better understanding of certain capabilities provided by the Android SDK before they are fully introduced. First, let’s talk about the View class.

Introduction to Android Views

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, which are used to create interactive UI components (buttons, text fields, etc.). The ViewGroup subclass is the base class for layouts, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

All of the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files. There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

Once you have created a tree of views, there are typically a few types of common operations you may wish to perform:

Set properties: for example, setting the text of a TextView. The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.
Set focus: The framework will handle moving focus in response to user input. To force focus to a specific view, call requestFocus().

Set up listeners: Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using setOnFocusChangeListener(android.view.View.OnFocusChangeListener).
Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.

Set visibility: You can hide or show views using setVisibility(int).

Introduction to Android Controls

The Android SDK contains a Java package named android.widget. When we refer to controls, we are typically referring to a class within this package.The Android SDK includes classes to draw most common objects, including ImageView, FrameLayout, EditText, and Button classes. All controls are typically derived from the View class. We cover many of these basic controls in detail.

Introduction to Android Layout

One special type of control found within the android.widget package is called a layout. A layout control is still a View object, but it doesn't actually draw anything specific on the screen. Instead, it is a parent container for organizing other controls (children). Layout controls determine how and where on the screen child controls are drawn. Each type of layout control draws its children using particular rules. For instance, the LinearLayout control draws its child controls in a single horizontal row or a single vertical column. Similarly, a TableLayout control displays each child control in tabular format (in cells within specific rows and columns).

By necessity, we use some of the layout View objects within this module to illustrate how to use the controls previously mentioned. However, we don't go into the details of the various layout types available as part of the Android SDK until the next module. We will lean in more details about layout in next module.

TextView

TextView is a user interface element that displays text to the user. Following table shows important XML Attributes of TextView control.

Attribute	**Description**
id	id is an attribute used to uniquely identify a text view
gravity	The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.
text	text attribute is used to set the text in a text view.
textColor	textColor attribute is used to set the text color of a text view. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".
textSize	textSize attribute is used to set the size of text of a text view. We can

	set the text size in sp(scale independent pixel) or dp(density pixel).
textStyle	textStyle attribute is used to set the text style of a text view. The possible text styles are bold, italic and normal.
background	background attribute is used to set the background of a text view. We can set a color or a drawable in the background of a text view
padding	padding attribute is used to set the padding from left, right, top or bottom.

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_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="This is TextView"
         android:layout_centerInParent="true"
         android:textSize="35sp"
         android:padding="15dp"
         android:textColor="#aaa"
         android:background="#fff"/>
</LinearLayout>
```

This code sample demonstrates how to modify the contents of the text view defined in the previous XML layout:

```
 public class MainActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView helloTextView = (TextView) findViewById(R.id.text_view_id);
        helloTextView.setText(R.string.user_greeting);
    }
}
```

To display this TextView on the screen, all your Activity needs to do is call the setContentView() method with the layout resource identifier in which you defined in the preceding XML shown.

You can change the text displayed programmatically by calling the setText() method on the TextView object. Retrieving the text is done with the getText() method. To customize the appearance of TextView we can use Styles and Themes.

EditText

EditText is a user interface element for entering and modifying text. Following table shows important XML Attributes of EditText control.

Attribute	**Description**
id	This is an attribute used to uniquely identify an edit text
gravity	The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.
text	This attribute is used to set the text in a text view.
hint	It is an attribute used to set the hint i.e. what you want user to enter in this edit text. Whenever user start to type in edit text the hint will automatically disappear.
lines	Defines how many lines tall the input box is. If this is not set, the entry field grows as the user enters text.
textColorHint	It is an attribute used to set the color of displayed hint.
textColor	This attribute is used to set the text color of a edit text. Color value is

	in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.
textSize	This attribute is used to set the size of text of a edit text. We can set the text size in sp(scale independent pixel) or dp(density pixel).
textStyle	This attribute is used to set the text style of a edit text. The possible text styles are bold, italic and normal.
background	This attribute is used to set the background of a edit text. We can set a color or a drawable in the background of a edit text
padding	Padding attribute is used to set the padding from left, right, top or bottom.

Following layout code shows a basic EditText element.

```
<EditText
android:id=”@+id/txtName”
android:layout_height=”wrap_content”
android:hint=”Full Name”
android:lines=”4”
android:layout_width=”fill_parent” />
```

The EditText object is essentially an editable TextView. You can read text from it in by using the getText() method. You can also set initial text to draw in the text entry area using the setText() method. You can also highlight a portion of the text from code by call to setSelection() method and a call to selectAll() method highlights the entire text entry field.

By default, the user can perform a long press to bring up a context menu. This provides to the user some basic copy, cut, and paste operations as well as the ability to change the input method and add a word to the user’s dictionary of frequently used words. You can set the editable attribute to false, so the user cannot edit the text in the field but can still copy text out of it using a long press.

AutoCompleteTextView

In Android, AutoCompleteTextView is a view i.e. similar to EditText, except that it displays a list of completion suggestions automatically while the user is typing. A list of suggestions is displayed in drop down menu from which user can choose an item which actually replace the content of EditBox with that.
It is a subclass of EditText class so we can inherit all the properties of EditText in a AutoCompleteTextView.

Following layout code shows a basic AutoCompleteTextView element.

```
<AutoCompleteTextView
android:id="@+id/ac"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text=" Auto Suggestions EditText"/>
```

To display the Array content in an AutoCompleteTextView we need to implement Adapter. In AutoCompleteTextView we mainly display text values so we use Array Adapter for that. ArrayAdapter is used when we need list of single type of items which is backed by an Array. For example, list of phone contacts, countries or names.

```
ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects)
AutoCompleteTextView ac = (AutoCompleteTextView) findViewById(R.id.ac);
```

Following code retrieve the value from a AutoCompleteTextView in Java class.

```
String v = ac.getText().toString();
```

Button

A user interface element the user can tap or click to perform an action. To display a button in an activity, add a button to the activity's layout XML file:

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

To specify an action when the button is pressed, set a click listener on the button object in the corresponding activity code:

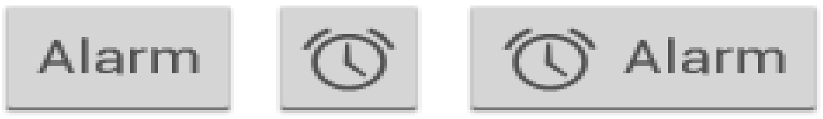

Figure-61

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.content_layout_id);

        final Button button = findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Code here executes on main thread after user presses button
            }
        });
    }
}
```

The above snippet creates an instance of View.OnClickListener and wires the listener to the button using setOnClickListener(View.OnClickListener). As a result, the system executes the code you write in onClick(View) after the user presses the button.

Every button is styled using the system's default button background, which is often different from one version of the platform to another. If you are not satisfied with the default button style, you can customize it.

Checkbox

A checkbox is a specific type of two-states button that can be either checked or unchecked.

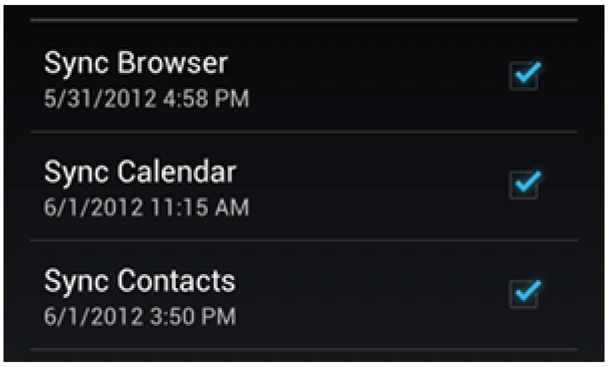

Figure-62

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.

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.

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>
```

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

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

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. If it's not necessary to show all options side-by-side, use a spinner instead.

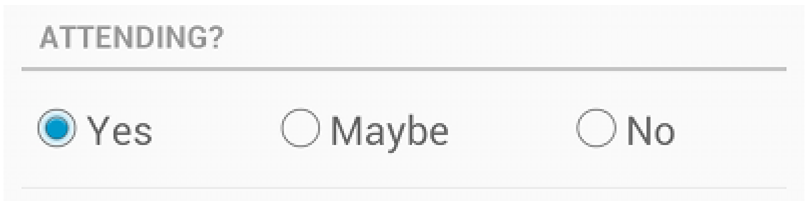

Figure-63

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.

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.

For example, here are a couple RadioButton objects:

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

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

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
            break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
            break;
    }
}
```

Let us sum up

In this module you have learned about user interface elements available within the Android Software Development Kit (SDK). We have discussed use of various user interface elements and use of different components and controls to build a screen; this module also explains about how application can listen for various actions performed by the user.

Further Reading

- https://developer.android.com/reference/android/widget/TextView
- https://developer.android.com/reference/android/widget/EditText
- https://developer.android.com/reference/android/widget/Button
- https://developer.android.com/reference/android/widget/CheckBox

Activity

- Write an application to demonstrate use of user interface element you have learned in this module.

Acknowledgement: “The content in this module is modifications based on work created and shared by the Android Open-Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.”