Android Cookbook

A Crowd-sourced Cookbook on Writing Great Android® Apps

Five Ways to Wire Up an Event Listener

Chapter	7. Graphical User Interface
Contributed by	Daniel Fowler 2011-07-27 07:45:48 (updated 2012-02-09 09:35:29)
In Published Edition?	Yes
Minimum Version	1.5

0 Votes

Problem

Developers need to be familiar with the different ways to code event handlers, they they will come across different methods in tutorials, samples and online snippets.

Solution

When writing software very rarely is there only one way to do things, this is true when wiring up View events, five methods are shown here.

Discussion

When a `View` fires an event an Application will not respond to it unless it is listening for it. To detect the event a class that implements a listener is instantiated and assigned to the View. Take for example the `onClick` event, the most widely used event in Android Apps. Nearly every View that can be added to an App screen will fire the event when the user stabs it with their finger (on touch screens) or presses the trackpad/trackball when the View has focus. This event is listened to by a class implementing the `OnClickListener` interface. The class instance is then assigned to the required View using the View's `setOnClickListener` method. In the following code an `Activity` sets the text of a `TextView` (textview1) when a `Button` (button1) is pressed.

1. Member Class

A class called *HandleClick* implementing `OnClickListener` is declared as a member of the `Activity` (*main*). This is useful when several listeners require similar processing than can be handled by a single class.

```
public class main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //attach an instance of HandleClick to the Button
        findViewById(R.id.button1).setOnClickListener(new HandleClick());
    }
    private class HandleClick implements OnClickListener{
        public void onClick(View arg0) {
            Button btn = (Button)arg0;  //cast view to a button
            // get a reference to the TextView
            TextView tv = (TextView) findViewById(R.id.textview1);
            // update the TextView text
            tv.setText("You pressed " + btn.getText());
        }
```

```
    }
}
```

2. Interface Type

In Java an Interface can be used as a type, a variable is declared as an `OnClickListener` and assigned using `new OnClickListener(){...}`, behind the scenes Java is creating an object (an Anonymous Class) that implements `OnClickListener`. This has similar benefits to the first method.

```
public class main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //use the handleClick variable to attach the event listener
        findViewById(R.id.button1).setOnClickListener(handleClick);
    }
    private OnClickListener handleClick = new OnClickListener(){
        public void onClick(View arg0) {
            Button btn = (Button)arg0;
            TextView tv = (TextView) findViewById(R.id.textview1);
            tv.setText("You pressed " + btn.getText());
        }
    };
}
```

3. Anonymous Inner Class

Declaring the `OnClickListener` within the call to the `setOnClickListener` method is common. This method is useful when each listener does not have functionality that could be shared with other listeners. Some novice developers find this type of code difficult to understand. Again behind the scenes for `new OnClickListener(){...}` Java is creating an object that implements the interface.

```
public class main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        findViewById(R.id.button1).setOnClickListener(new OnClickListener(){
            public void onClick(View arg0) {
                Button btn = (Button)arg0;
                TextView tv = (TextView) findViewById(R.id.textview1);
                tv.setText("You pressed " + btn.getText());
            }
        });
    }
}
```

4. Implementation in Activity

The Activity itself can implement the `OnClickListener`. Since the Activity object (*main*) already exists this saves a small amount of memory by not requiring another object to host the `onClick` method. It does make public a method that is unlikely to be used elsewhere. Implementing multiple events will make the declaration of *main* long.

```
public class main extends Activity implements OnClickListener{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        findViewById(R.id.button1).setOnClickListener(this);
    }
    public void onClick(View arg0) {
        Button btn = (Button)arg0;
        TextView tv = (TextView) findViewById(R.id.textview1);
        tv.setText("You pressed " + btn.getText());
```

```
    }
}
```

5. Attribute in View Layout for OnClick Events

In Android 1.6 and later (API level 4 and upwards) the name of a method defined in the Activity can be assigned to the `android:onClick` attribute in a layout file. This can save writing a lot of boilerplate code.

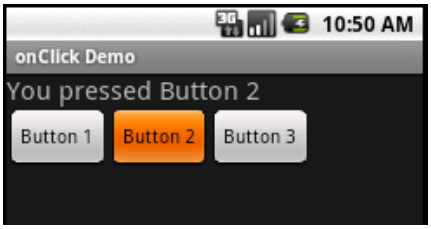
```
public class main extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void HandleClick(View arg0) {
        Button btn = (Button)arg0;
        TextView tv = (TextView) findViewById(R.id.textview1);
        tv.setText("You pressed " + btn.getText());
    }
}
```

In the layout file the Button would be declared with the `android:onClick` attribute.

```
<Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:onClick="HandleClick"/>
```

The first four methods of handling events can be used with other event types (`onLongClick`, `onKey`, `onTouch`, `onCreateContextMenu`, `onFocusChange`). The fifth method only applies to the `onClick` event. The layout file below declares an additonal two buttons and using the `android:onClick` attribute no additional code is required than that defined above, i.e. no additional `findViewById` and `setOnClickListener` for each button is required.

```
<?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:id="@+id/textview1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Click a button."
            android:textSize="20dp"/>
  <LinearLayout android:orientation="horizontal"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content">
    <Button android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 1"
            android:onClick="HandleClick"/>
    <Button android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 2"
            android:onClick="HandleClick"/>
    <Button android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 3"
            android:onClick="HandleClick"/>
  </LinearLayout>
</LinearLayout>
```



Deciding which technique to use to wire up a listener will depend on the functionality required, how much code is reusable across Views and how easy the code would be to understand by future maintainers. Ideally the code should be succint and easy to view.

One method not shown here is similar to the first method. In the first method it would be possible to save the listener class in a different class file as a public class. Then instances of that public class could be used by other Activities, passing the Activity's context in via the constructor. However, Activities should try and stay self contained in case they are killed by Android. Sharing listeners across Activities is against the ideals of the Android platform and could lead to unnecessary complexity passing references between the public classes.

Source Download URL http://tekeye.biz/download/codinglisteners.zip

View WikiText | Edit | Done

Comments (0) | Leave a comment | Edit History (11)

There are no (moderator-approved) comments on this recipe yet.

Privacy Policy Terms and Conditions Powered by Seam 2.1.2 and RichFaces. Originally created by seam-gen.