Android - Event Handling

Events are a useful way to collect data about a user's interaction with interactive components of Applications. *Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.*

There are following three concepts related to Android Event Management –

- **Event Listeners** – An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners Registration** – Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** – When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.

onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

There are many more event listeners available as a part of **View** class like OnHoverListener, OnDragListener etc which may be needed for your application. So I recommend to refer official documentation for Android application development in case you are going to develop a sophisticated apps.

Event Listeners Registration

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event. Though there are several tricky ways to register your event listener for any event, but I'm going to list down only top 3 ways, out of which you can use any of them based on the situation.

- Using an Anonymous Inner Class
- Activity class implements the Listener interface.
- Using Layout file activity_main.xml to specify event handler directly.

Below section will provide you detailed examples on all the three scenarios –

Touch Mode

Users can interact with their devices by using hardware keys or buttons or touching the screen.Touching the screen puts the device into touch mode. The user can then interact with it by touching the on-screen virtual buttons, images, etc.You can check if the device is in touch mode by calling the View class's isInTouchMode() method.

Focus

A view or widget is usually highlighted or displays a flashing cursor when it's in focus. This indicates that it's ready to accept input from the user.

- **isFocusable()** – it returns true or false
- **isFocusableInTouchMode()** – checks to see if the view is focusable in touch mode. (A view may be focusable when using a hardware key but not when the device is in touch mode)

```
android:foucsUp="@=id/button_1"
```

onTouchEvent()

```
public boolean onTouchEvent(motionEvent event){
   switch(event.getAction()){
      case TOUCH_DOWN:
      Toast.makeText(this,"you have clicked down Touch button",Toast.LENTH_LONG)
      break();

      case TOUCH_UP:
      Toast.makeText(this,"you have clicked up touch button",Toast.LENTH_LONG).sl
      break;

      case TOUCH_MOVE:
      Toast.makeText(this,"you have clicked move touch button"Toast.LENTH_LONG).s
      break;
   }
   return super.onTouchEvent(event) ;
}
```

Event Handling Examples

Event Listeners Registration Using an Anonymous Inner Class

Here you will create an anonymous implementation of the listener and will be useful if each class is applied to a single control only and you have advantage to pass arguments to event handler. In this approach event handler methods can access private data of Activity. No reference is needed to call to Activity.

But if you applied the handler to more than one control, you would have to cut and paste the code for the handler and if the code for the handler is long, it makes the code harder to maintain.

Following are the simple steps to show how we will make use of separate Listener class to register and capture click event. Similar way you can implement your listener for any other required event type.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as myapplication under a package com.example.myapplication as explained in the Hello World Example chapter.
2	Modify src/MainActivity.java file to add click event listeners and handlers for the two buttons defined.
3	Modify the detault content of res/layout/activity_main.xml file to include Android UI controls.
4	No need to declare default string constants.Android studio takes care default constants.
5	Run the application to launch Android emulator and verify the result of the changes done in the aplication.

Following is the content of the modified main activity file **src/com.example.myapplication/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.myapplication;

import android.app.ProgressDialog;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends ActionBarActivity {
   private ProgressDialog progress;
```

```
   Button b1,b2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      progress = new ProgressDialog(this);

      b1=(Button)findViewById(R.id.button);
      b2=(Button)findViewById(R.id.button2);
      b1.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            TextView txtView = (TextView) findViewById(R.id.textView);
            txtView.setTextSize(25);
         }
      });

      b2.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            TextView txtView = (TextView) findViewById(R.id.textView);
            txtView.setTextSize(55);
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file –

Here abc indicates about tutorialspoint logo

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

<TextView
   android:id="@+id/textView1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Event Handling "
   android:layout_alignParentTop="true"
   android:layout_centerHorizontal="true"
   android:textSize="30dp"/>

<TextView
   android:id="@+id/textView2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Tutorials point "
   android:textColor="#ff87ff09"
   android:textSize="30dp"
   android:layout_above="@+id/imageButton"
   android:layout_centerHorizontal="true"
   android:layout_marginBottom="40dp" />

<ImageButton
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/imageButton"
   android:src="@drawable/abc"
   android:layout_centerVertical="true"
   android:layout_centerHorizontal="true" />

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Small font"
   android:id="@+id/button"
   android:layout_below="@+id/imageButton"
   android:layout_centerHorizontal="true" />

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Large Font"
   android:id="@+id/button2"
```

```
      android:layout_below="@+id/button"
      android:layout_alignRight="@+id/button"
      android:layout_alignEnd="@+id/button" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
      android:id="@+id/textView"
      android:layout_below="@+id/button2"
      android:layout_centerHorizontal="true"
      android:textSize="25dp" />

</RelativeLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">myapplication</string>
</resources>
```

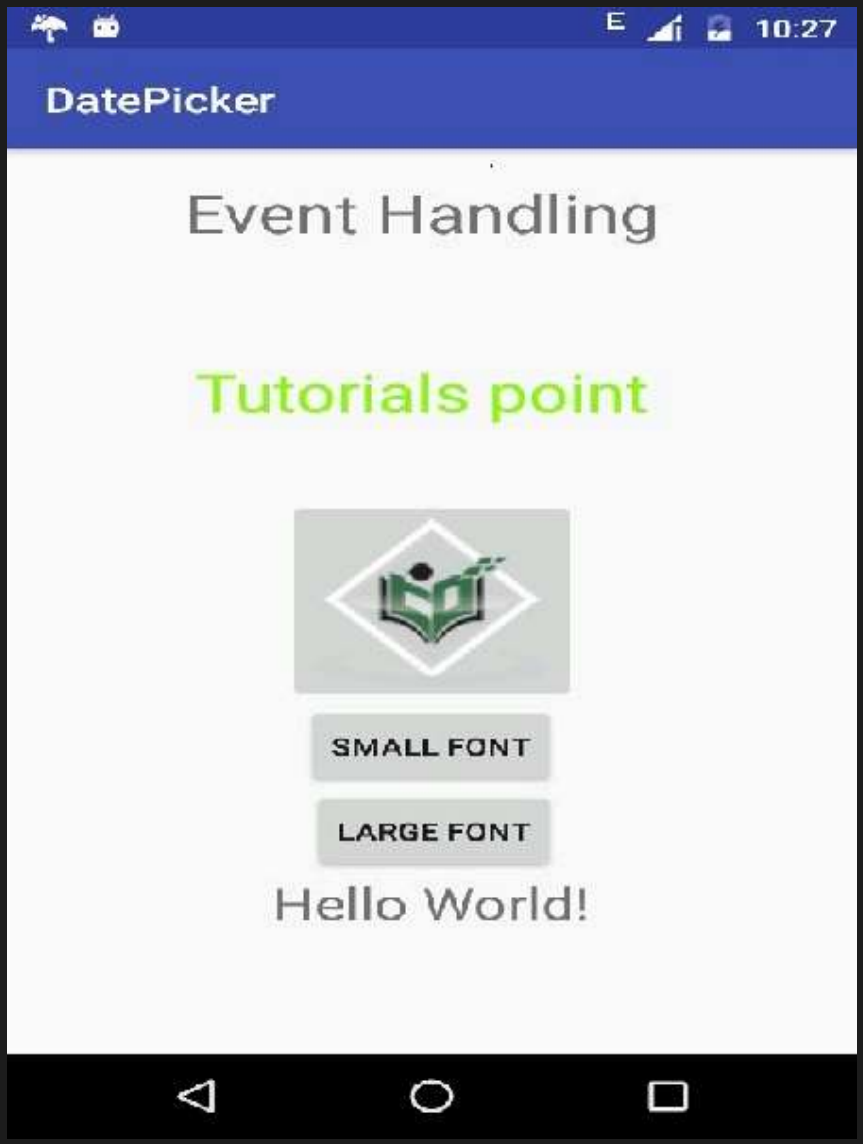
Following is the default content of **AndroidManifest.xml** –

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.myapplication" >

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <activity
         android:name="com.example.myapplication.MainActivity"
         android:label="@string/app_name" >

         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>

      </activity>
```

```
    </application>
</manifest>
```

Let's try to run your **myapplication** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run

icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window –



Now you try to click on two buttons, one by one and you will see that font of the **Hello World** text will change, which happens because registered click event handler method is being called against each click event.

Exercise

I will recommend to try writing different event handlers for different event types and understand exact difference in different event types and their handling. Events related to menu, spinner, pickers widgets are little different but they are also based on the same concepts as explained above.