2.b. Event handling

Mario Mata
mario.mata@gcu.ac.uk
M215a

GCU
Glasgow Caledonian University
University for the Common Good

Events

- Views trigger **events** when the user interacts with them
- Our app can then trap (handle) those events, and act accordingly
  - There are several ways to do this
- If you didn't do it yet, please review:
  Android Development - Guide 2 - Listeners, Event Handlers and Debugging.pdf
  which is available on GCULearn.

Event handling

- We need to "connect" (**attach**) a *callback* method (the event handler) to the event object triggered by the View

For instance, consider the `OnClick` event from a Button (the approach is the same for any other event)

1. The class containing the event handler method has to:
   - **implement** the event's **interface**, in this case `OnClickListener`
   - Have a method named like the event to handle, i.e. `onClick()`
2. And we need to register (attach) the class as the handler for the View's with `setOnClickListener()`

Let's quickly summarize 3 ways of doing this

A - Event with Activity class

1. The class containing the event handler method has to:
   - **implement** the event's **interface**, in this case `OnClickListener`
   - Have a method named like the event to handle, i.e. `onClick()`
2. Register (attach) the class as a handler for the View's

```
public class MainActivity extends AppCompatActivity
                          implements OnClickListener {
   private Button exitBtn; //handler for the Button
   @Override
     public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
           ……
        exitBtn = (Button)findViewById(R.id.exitButton);
        exitBtn.setOnClickListener(this);
     }
   @Override
     public void onClick (View v) {
       if (v == exitBtn)
            doSomething();
     }
```

B - Event with *anonymous* class/object

1. The class containing the event handler method has to:
   - **implement** the event's **interface**, in this case `OnClickListener`
   - Have a method named like the event to handle, i.e. `onClick()`
2. Register (attach) the class as a handler for the View's

```
public class MainActivity extends AppCompatActivity {
private Button exitBtn; //handler for the Button
  @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        exitBtn = (Button)findViewById(R.id.exitButton);
        exitBtn.setOnClickListener(eBtnListener);
    }

   private OnClickListener eBtnListener;
   eBtnListener = new OnClickListener(){
     @Override
     public void onClick (View v) {
          doSomething();
       }
    };
```

This creates a new class and object (with no name) on-the-fly, with an `onClick()` handler method

C - Simple event handler

For simple cases, we can specify a method name as the event handler using a property in the *xml*:

```
<Button
   android:id = "@+id/exitButton"
   ……
   android:onClick = "exitBtnClickHandler" />
```

And we implement that method in the Activity – it must have *the same parameters* as the interface for `onClick()`

```
public class MainActivity extends AppCompatActivity{
   private Button exitBtn; //handler for the Button
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
           ……
        exitBtn = (Button)findViewById(R.id.exitButton);
   }
   public void exitBtnClickHandler (View v) {
       if (v == exitBtn)
            doSomething();
   }
```

Which event handling to use?

- Approach A (Activity as the listener) handles the same event type (i.e. OnClick) from several widgets in the same place
  - Best to handle *related* widgets together
- Approach B (anonymous in-line handler) sets a dedicated handler for a single widget
  - Clean and convenient for widgets unrelated to others
- Approach C (simple event handler) seems convenient, but it presents issues:
  - The name of the handler method needs to match in layout xml and Java code files –inconvenient to reuse
  - Need to know the *signature* of the method (whilst the IDE can auto-complete this for us when using approach A or B)
- You can **combine them as needed** (typically A and B)

Other events

- `OnLongClick`, from `View.OnLongClickListener`
  - Triggered when the user presses the widget for over 1 second
  - Interface definition: `public boolean onLongClick (View v)`
    - We must return `true` if we used the event, or `false` if we ignored it

- `OnTouch`, from `View.OnTouchListener`
  - Similar to `OnClick`, but richer information from Motion events (ACTION_x):

```
public boolean onTouch (View v, MotionEvent ev) {
 boolean status = false;   //mark as unused
 if (v == button3){   //this is the button we are handling
    status = true;  //mark as used
    if (ev.getAction() == MotionEvent.ACTION_DOWN){
                // do something on press
    }
    else if (ev.getAction() == MotionEvent.ACTION_UP){
             //do something on release
    }
  }
  return status;
}
```