Event Handling – Listening for Events

So far, we have dealt with default consequences of User interaction with the swing components we have created, Frames can be minimised or closed with clicks on the top right hand corner of the Frame, Dialogs have buttons built in with predefined actions etc ….

This section introduces the concept of Events and Actions, this will be developed on as each new component is introduced, because each has a different response to an event occurring.

This section is really to introduce the concept which is standard to all components.

Event Handling – Listening for Events

One of the most basic concepts behind Events is that something must be created to 'listen' for the Event's occurrence. If a JButton is created on a JFrame it will have the ability already defined within its classification to indent and return when the User clicks on it, however there is no other processing behind it, just as an electrician can screw light switches on the wall so a JButton can be created without the wiring etc behind it therefore not serving any purpose.

The wiring in the case of an Event is called a 'listener' and the Developer will need to create a listener defined for a Component if that event is to be 'actioned'.

Event Handling – Listening for Events

There are effectively three stages to developing event handling, not necessarily in this order …

The first stage is to create a component which will have an event which needs handling.

Second, a listener is created to watch for the event taking place, this is then associated to the Component.

And lastly, the listener will need a method which will be run when the listener detects an event has taken place.

Event Handling – Listening for Events

There are three main methods of defining listeners in Java

A **public** Listener, shareable by all components within a class and may need to be tested when an event occurs to ascertain which component 'fired' the event. Components needing to use this Listener must be registered to it.

A **private** Listener, created strictly for the use of one instance of a component, useful if only one component needs a listener but rather cumbersome if applied to a process which has ten buttons for example.

An **implemented** Listener, defined within the instance class and its events are tested within it.