GWT - EVENT HANDLING

GWT provides a event handler model similar to Java AWT or SWING User Interface frameworks.

- A listener interface defines one or more methods that the widget calls to announce an event. GWT provides a list of interfaces corresponding to various possible events.
- A class wishing to receive events of a particular type implements the associated handler interface and then passes a reference to itself to the widget to subscribe to a set of events.

For example, the **Button** class publishes **click** events so you will have to write a class to implement *ClickHandler* to handle **click** event.

Event Handler Interfaces

All GWT event handlers have been extended from *EventHandler* interface and each handler has only a single method with a single argument. This argument is always an object of associated event type. Each **event** object have a number of methods to manipulate the passed event object. For example for click event you will have to write your handler as follows:

```
/**
 * create a custom click handler which will call
 * onClick method when button is clicked.
 */
public class MyClickHandler implements ClickHandler {
   @Override
   public void onClick(ClickEvent event) {
      Window.alert("Hello World!");
   }
}
```

Now any class wishing to receive click events will call **addClickHandler** to register an event handler as follows:

```
/**
 * create button and attach click handler
 */
Button button = new Button("Click Me!");
button.addClickHandler(new MyClickHandler());
```

Each widget supporting an event type will have a method of the form HandlerRegistration add**Foo**Handler(**Foo**Event) where **Foo** is the actual event like Click, Error, KeyPress etc.

Following is the list of important GWT event handlers and associated events and handler registration methods:

S.N.	Event Interface	Event Method & Description
1	Before Selection Handler<I>	**void on Before Selection** *BeforeSelectionEvent < I > event*; Called when BeforeSelectionEvent is fired.
2	BlurHandler	**void on Blur***BlurEventevent*; Called when Blur Event is fired.
3	ChangeHandler	**void onChange***ChangeEventevent* ;

		Called when a change event is fired.
4	ClickHandler	**void onClick***ClickEventevent***;** Called when a native click event is fired.
5	CloseHandler<T>	**void onClose***CloseEvent* < *T* > *event* **;** Called when CloseEvent is fired.
6	Context Menu Handler	**void on Context Menu***ContextMenuEventevent***;** Called when a native context menu event is fired.
7	Double Click Handler	**void on Double Click** ***DoubleClickEventevent*;** Called when a Double Click Event is fired.
8	Error Handler	**void on Error***ErrorEventevent***;** Called when Error Event is fired.
9	Focus Handler	**void on Focus** ***FocusEventevent*** **;** Called when Focus Event is fired.
10	Form Panel.Submit Complete Handler	**void on Submit Complete** ***FormPanel. SubmitCompleteEventevent*** **;** Fired when a form has been submitted successfully.
11	FormPanel.SubmitHandler	**void on Submit***FormPanel. SubmitEventevent***;** Fired when the form is submitted.
12	Key Down Handler	**void on Key Down***KeyDownEventevent***;** Called when KeyDownEvent is fired.
13	KeyPressHandler	**void onKeyPress***KeyPressEventevent* **;** Called when KeyPressEvent is fired.
14	KeyUpHandler	**void onKeyUp***KeyUpEventevent* **;** Called when KeyUpEvent is fired.
15	LoadHandler	

		void onLoad*LoadEventevent***;** Called when LoadEvent is fired.
16	MouseDownHandler	**void onMouseDown***MouseDownEventevent* **;** Called when MouseDown is fired.
17	MouseMoveHandler	**void onMouseMove***MouseMoveEventevent***;** Called when MouseMoveEvent is fired.
18	MouseOutHandler	**void onMouseOut***MouseOutEventevent* **;** Called when MouseOutEvent is fired.
19	MouseOverHandler	**void onMouseOver***MouseOverEventevent***;** Called when MouseOverEvent is fired.
20	MouseUpHandler	**void onMouseUp***MouseUpEventevent* **;** Called when MouseUpEvent is fired.
21	MouseWheelHandler	**void onMouseWheel***MouseWheelEventevent* **;** Called when MouseWheelEvent is fired.
22	ResizeHandler	**void onResize***ResizeEventevent* **;** Fired when the widget is resized.
23	ScrollHandler	**void onScroll***ScrollEventevent* **;** Called when ScrollEvent is fired.
24	SelectionHandler<I>	**void onSelection***SelectionEvent < I > event* **;** Called when SelectionEvent is fired.
25	ValueChangeHandler<I>	**void onValueChange***ValueChangeEvent < I > event* **;** Called when ValueChangeEvent is fired.
26	Window.ClosingHandler	**void onWindowClosing***Window. ClosingEventevent* **;** Fired just before the browser window closes or navigates to a different site.

27	Window.ScrollHandler	**void onWindowScroll***Window. ScrollEventevent* **;** Fired when the browser window is scrolled.

Event Methods

As mentioned earlier, each handler has a single method with a single argument which holds the event object, for example *void onClick**ClickEventevent* or *void onKeyDown**KeyDownEventevent*. The event objects like *ClickEvent* and *KeyDownEvent* has few common methods which are listed below:

S.N.	Method & Description
1	**protected void dispatch***ClickHandlerhandler* This method Should only be called by HandlerManager
2	**DomEvent.Type <FooHandler> getAssociatedType** This method returns the type used to register **Foo** event.
3	**static DomEvent.Type<FooHandler> getType** This method gets the event type associated with **Foo** events.
4	**public java.lang.Object getSource** This method returns the source that last fired this event.
5	**protected final boolean isLive** This method returns whether the event is live.
6	**protected void kill** This method kills the event

Example

This example will take you through simple steps to show usage of a **Click** Event and **KeyDown** Event handling in GWT. Follow the following steps to update the GWT application we created in *GWT - Create Application* chapter:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint* as explained in the *GWT - Create Application* chapter.
2	Modify *HelloWorld.gwt.xml*, *HelloWorld.css*, *HelloWorld.html* and *HelloWorld.java* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to verify the result of the implemented logic.

Following is the content of the modified module descriptor **src/com.tutorialspoint/HelloWorld.gwt.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='helloworld'>
  <!-- Inherit the core Web Toolkit stuff.                        -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.                       -->
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>

  <!-- Specify the app entry point class.                         -->
  <entry-point class='com.tutorialspoint.client.HelloWorld'/>

  <!-- Specify the paths for translatable code                    -->
  <source path='client'/>
  <source path='shared'/>

</module>
```

Following is the content of the modified Style Sheet file **war/HelloWorld.css**.

```
body{
   text-align: center;
   font-family: verdana, sans-serif;
}
h1{
   font-size: 2em;
   font-weight: bold;
   color: #777777;
   margin: 40px 0px 70px;
   text-align: center;
}
```

Following is the content of the modified HTML host file **war/HelloWorld.html**.

```
<html>
<head>
<title>Hello World</title>
   <link rel="stylesheet" href="HelloWorld.css"/>
   <script language="javascript" src="helloworld/helloworld.nocache.js">
   </script>
</head>
<body>

<h1>Event Handling Demonstration</h1>
<div ></div>

</body>
</html>
```

Let us have following content of Java file **src/com.tutorialspoint/HelloWorld.java** which will demonstrate use of Event Handling in GWT.

```
package com.tutorialspoint.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.event.dom.client.KeyCodes;
import com.google.gwt.event.dom.client.KeyDownEvent;
import com.google.gwt.event.dom.client.KeyDownHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.DecoratorPanel;
import com.google.gwt.user.client.ui.HasHorizontalAlignment;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
```

```
public class HelloWorld implements EntryPoint {
   public void onModuleLoad() {
      /**
       * create textbox and attach key down handler
       */
      TextBox textBox = new TextBox();
      textBox.addKeyDownHandler(new MyKeyDownHandler());

      /*
       * create button and attach click handler
       */
      Button button = new Button("Click Me!");
      button.addClickHandler(new MyClickHandler());

      VerticalPanel panel = new VerticalPanel();
      panel.setSpacing(10);
      panel.setHorizontalAlignment(HasHorizontalAlignment.ALIGN_CENTER);
      panel.setSize("300", "100");
      panel.add(textBox);
      panel.add(button);

      DecoratorPanel decoratorPanel = new DecoratorPanel();
      decoratorPanel.add(panel);
      RootPanel.get("gwtContainer").add(decoratorPanel);
   }

   /**
    * create a custom click handler which will call
    * onClick method when button is clicked.
    */
   private class MyClickHandler implements ClickHandler {
      @Override
      public void onClick(ClickEvent event) {
         Window.alert("Hello World!");
      }
   }

   /**
    * create a custom key down handler which will call
    * onKeyDown method when a key is down in textbox.
    */
   private class MyKeyDownHandler implements KeyDownHandler {
      @Override
      public void onKeyDown(KeyDownEvent event) {
         if(event.getNativeKeyCode() == KeyCodes.KEY_ENTER){
            Window.alert(((TextBox)event.getSource()).getValue());
         }
      }
   }
}
```

Once you are ready with all the changes done, let us compile and run the application in development mode as we did in GWT - Create Application chapter. If everything is fine with your application, this will produce following result: