Unit 3: Event Handling

Prepared by: Aakash Raj Shakya

What is an Event?

Change in the state of an object is known as Event, i.e., event describes the change in the state of the source.

Events are generated as a result of user interaction with the graphical user interface components.

For example, click on button, dragging mouse etc.

In the event model, there are three participants:

1. Event Source
2. Event Object
3. Event Listener

Event Handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The **Delegation Event Model** has the following key participants.

1. **Source –** The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide us with classes for the source object.
2. **Listener –** It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

Advantage of this Delegate Event Model are:

1. This approach is that the user interface logic is completely separated from the logic that generates the event.
2. The user interface element is able to delegate the processing of an event to a separate piece of code.

Listener Interfaces

Event listeners represent the interfaces responsible to handle events.

EventListner interface is a marker interface which every listener interface has to extend. This class is defined in **java.util** package.

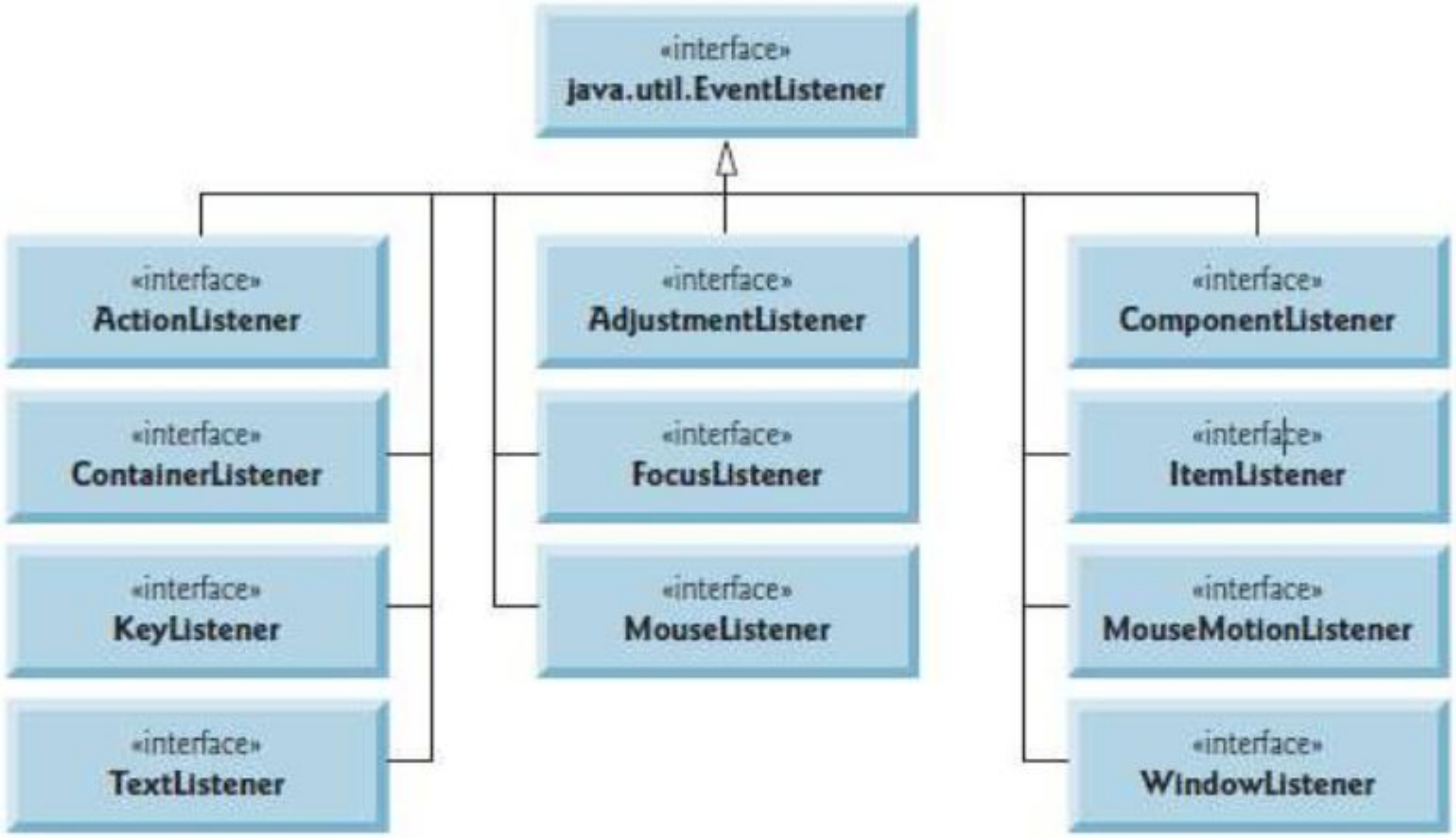
Each event-listener interface specifies one or more event-handling methods that must be declared in the class that implements the interface.

Any class which implements an interface must declare all the abstract methods of that interface; otherwise, the class is an abstract class and cannot be used to create objects.

Listener interface contd..

When an event occurs, the GUI component with which the user interacted **notifies its registered listeners by calling each listener’s appropriate event-handling method**. For example, when the user presses the Enter key in a JTextField, the registered listener’s actionPerformed method is called.

The inherited listener interface resides in **java.awt.event.*** package.


«interface»
java.util.EventListener
«interface»
ActionListener
«interface»
AdjustmentListener
«interface»
ComponentListener
«interface»
ContainerListener
«interface»
FocusListener
«interface»
ItemListener
«interface»
KeyListener
«interface»
MouseListener
«interface»
MouseMotionListener
«interface»
TextListener
«interface»
WindowListener

Interface	**Description**
ActionListener	This interface is used for receiving the action events.
ComponentListener	This interface is used for receiving the component events.
ItemListener	This interface is used for receiving the item events.
KeyListener	This interface is used for receiving the key events.
MouseListener	This interface is used for receiving the mouse events.

TextListener	This interface is used for receiving the text events.
WindowListener	This interface is used for receiving the window events.
AdjustmentListener	This interface is used for receiving the adjustment events.
ContainerListener	This interface is used for receiving the container events.
MouseMotionListener	This interface is used for receiving the mouse motion events.
FocusListener	This interface is used for receiving the focus events.

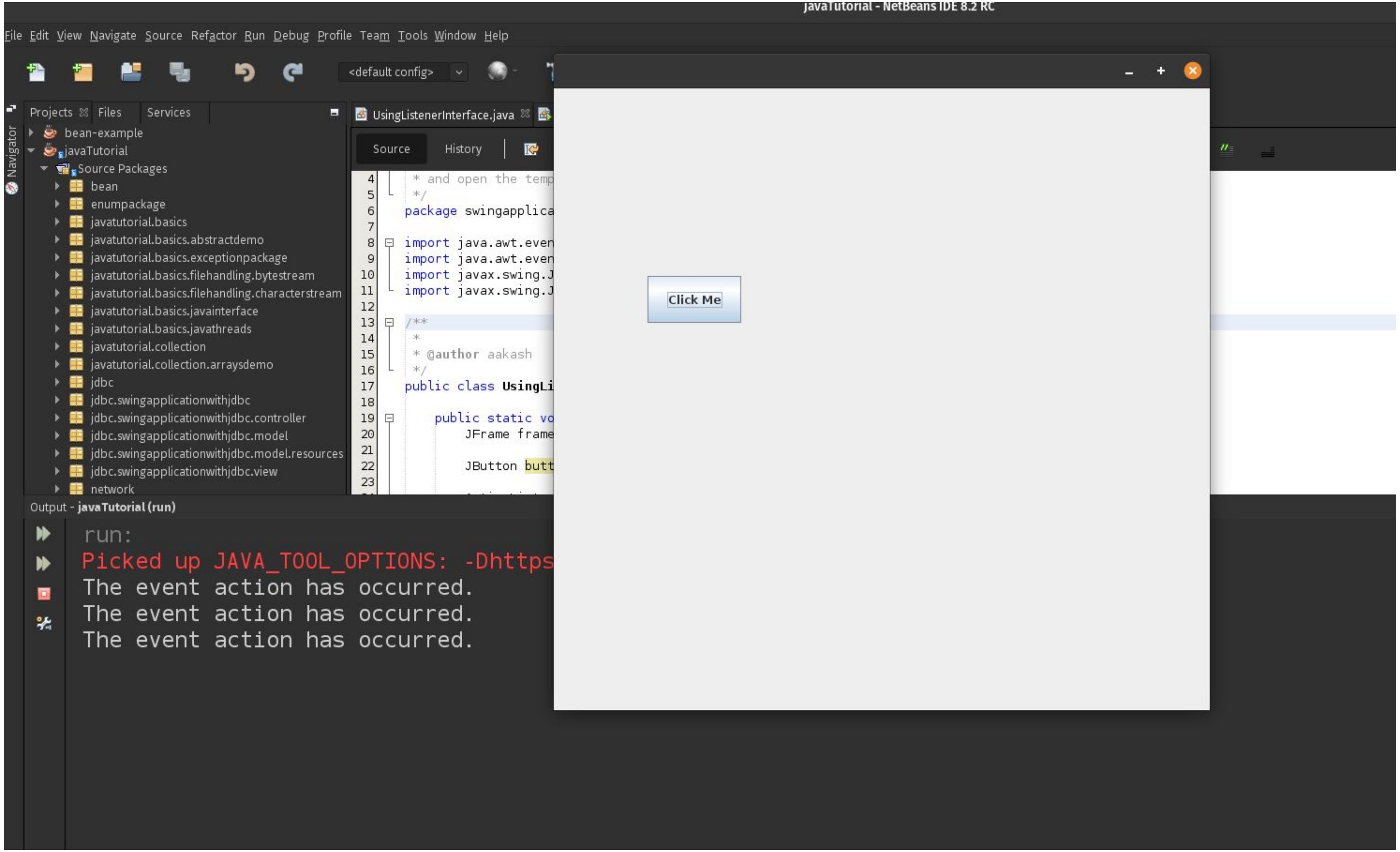
```java
import javax.swing.*;
import java.awt.event.*;

public class ListenerDemo {
   public static void main(String[] args) {
       JFrame frame = new JFrame();
       JButton button = new JButton("Click Me");

       ActionListenerImpl action = new ActionListenerImpl();
       button.addActionListener(action);
       button.setBounds(100, 200, 100, 50);

       frame.add(button);
       frame.setSize(700, 700);
       frame.setLayout(null);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
   }
}

class ActionListenerImpl implements ActionListener {
   @Override
   public void actionPerformed(ActionEvent e) {
       System.out.println("The event action has occurred.");
   }
}
```


javaTutorial - NetBeans IDE 8.2 RC
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects Files Services
bean-example
javaTutorial
Source Packages
bean
enumpackage
javatutorial.basics
javatutorial.basics.abstractdemo
javatutorial.basics.exceptionpackage
javatutorial.basics.filehandling.bytestream
javatutorial.basics.filehandling.characterstream
javatutorial.basics.javainterface
javatutorial.basics.javathreads
javatutorial.collection
javatutorial.collection.arraysdemo
jdbc
jdbc.swingapplicationwithjdbc
jdbc.swingapplicationwithjdbc.controller
jdbc.swingapplicationwithjdbc.model
jdbc.swingapplicationwithjdbc.model.resources
jdbc.swingapplicationwithjdbc.view
network
UsingListenerInterface.java
Source History
Click Me
Output - javaTutorial (run)

```
 * and open the temp
 */
package swingapplica

import java.awt.even
import java.awt.even
import javax.swing.J
import javax.swing.J

/**
 *
 * @author aakash
 */
public class UsingLi

    public static vo
        JFrame frame

        JButton butt
```

```
run:
Picked up JAVA_TOOL_OPTIONS: -Dhttps
The event action has occurred.
The event action has occurred.
The event action has occurred.
```

Adapter Classes

Java adapter classes provide the default implementation of listener interfaces.

If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it prevents us from writing unwanted code.

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exist as convenience for creating listener objects.

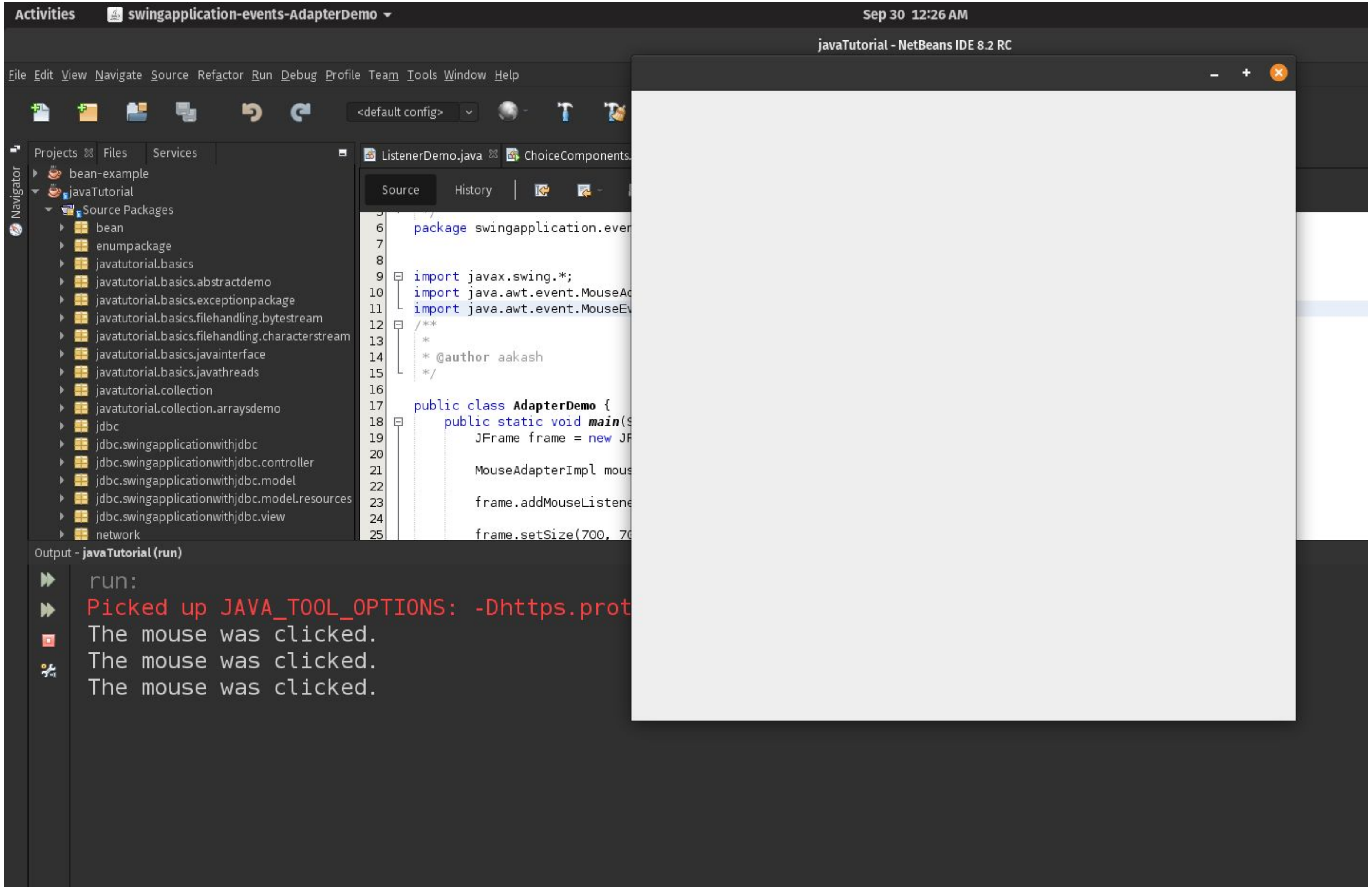
Adapter	Description
FocusAdapter	An abstract adapter class for receiving focus events.
KeyAdapter	An abstract adapter class for receiving key events.
MouseAdapter	An abstract adapter class for receiving mouse events.
MouseMotionAdapter	An abstract adapter class for receiving mouse motion events.
WindowAdapter	An abstract adapter class for receiving window events.

```
import javax.swing.*;
import java.awt.event.*;

public class AdapterDemo {
   public static void main(String[] args) {
       JFrame frame = new JFrame();

       MouseAdapterImpl mouseAdapter = new MouseAdapterImpl();
       frame.addMouseListener(mouseAdapter);

       frame.setSize(700, 700);
       frame.setLayout(null);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
   }
}
class MouseAdapterImpl extends MouseAdapter {
   @Override
   public void mouseClicked(MouseEvent e) {
       System.out.println("The mouse was clicked.");
   }
}
```


Activities
swingapplication-events-AdapterDemo
Sep 30 12:26 AM
javaTutorial - NetBeans IDE 8.2 RC
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects Files Services
bean-example
javaTutorial
Source Packages
bean
enumpackage
javatutorial.basics
javatutorial.basics.abstractdemo
javatutorial.basics.exceptionpackage
javatutorial.basics.filehandling.bytestream
javatutorial.basics.filehandling.characterstream
javatutorial.basics.javainterface
javatutorial.basics.javathreads
javatutorial.collection
javatutorial.collection.arraysdemo
jdbc
jdbc.swingapplicationwithjdbc
jdbc.swingapplicationwithjdbc.controller
jdbc.swingapplicationwithjdbc.model
jdbc.swingapplicationwithjdbc.model.resources
jdbc.swingapplicationwithjdbc.view
network
ListenerDemo.java
ChoiceComponents.
Source History
Navigator
Output - javaTutorial (run)

```
package swingapplication.ever


import javax.swing.*;
import java.awt.event.MouseAc
import java.awt.event.MouseEv
/**
 *
 * @author aakash
 */

public class AdapterDemo {
    public static void main(S
        JFrame frame = new JF

        MouseAdapterImpl mous

        frame.addMouseListene

        frame.setSize(700, 70
```

```
run:
Picked up JAVA_TOOL_OPTIONS: -Dhttps.prot
The mouse was clicked.
The mouse was clicked.
The mouse was clicked.
```

What is the difference between Listeners Interfaces and Adapter classes?

Handling Events

Event handling has three main components:

1. **Events :** An event is a change in state of an object.
2. **Events Source :** Event source is an object that generates an event.
3. **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

How Events are handled?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

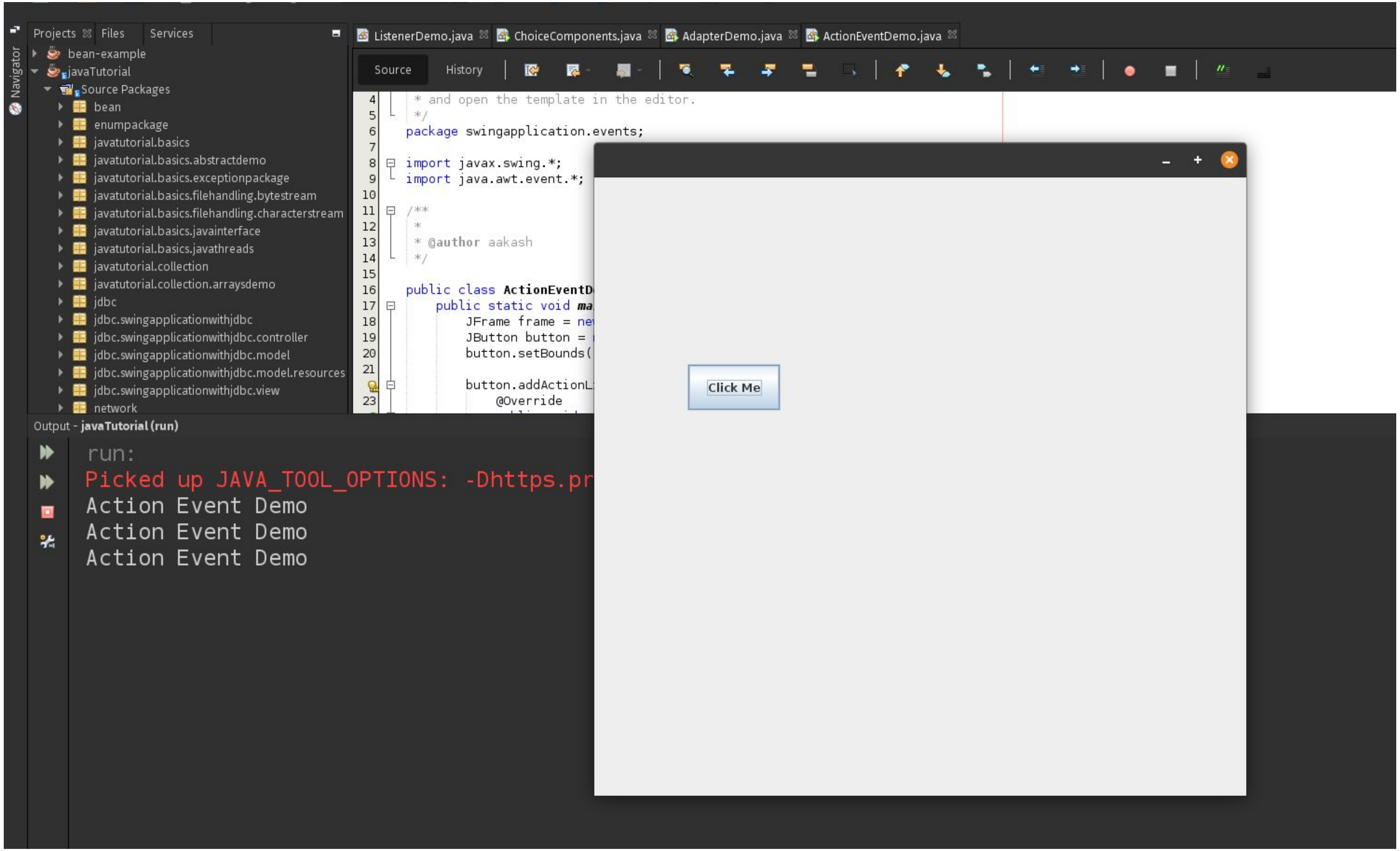
Events	**Description**
ActionEvent	The ActionEvent is generated when the button is clicked or the item of a list is double-clicked.
KeyEvent	On entering the character the Key event is generated.
MouseEvent	This event indicates a mouse action occurred in a component.
WindowEvent	The object of this class represents the change in the state of a window.
FocusEvent	Invoked when a component gains/loses the keyboard focus.
ItemEvent	Generated when checkbox or list item is clicked

```java
import javax.swing.*;
import java.awt.event.*;

public class ActionEventDemo {
   public static void main(String[] args) {
       JFrame frame = new JFrame();
       JButton button = new JButton("Click Me");
       button.setBounds(100, 200, 100, 50);

       button.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               System.out.println("Action Event Demo");
           }
       });

       frame.add(button);
       frame.setSize(700, 700);
       frame.setLayout(null);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
   }
}
```


Projects
Files
Services
Navigator
bean-example
javaTutorial
Source Packages
bean
enumpackage
javatutorial.basics
javatutorial.basics.abstractdemo
javatutorial.basics.exceptionpackage
javatutorial.basics.filehandling.bytestream
javatutorial.basics.filehandling.characterstream
javatutorial.basics.javainterface
javatutorial.basics.javathreads
javatutorial.collection
javatutorial.collection.arraysdemo
jdbc
jdbc.swingapplicationwithjdbc
jdbc.swingapplicationwithjdbc.controller
jdbc.swingapplicationwithjdbc.model
jdbc.swingapplicationwithjdbc.model.resources
jdbc.swingapplicationwithjdbc.view
network
ListenerDemo.java
ChoiceComponents.java
AdapterDemo.java
ActionEventDemo.java
Source
History
Click Me
Output - javaTutorial (run)

```
 * and open the template in the editor.
 */
package swingapplication.events;

import javax.swing.*;
import java.awt.event.*;

/**
 *
 * @author aakash
 */

public class ActionEventD
    public static void ma
        JFrame frame = ne
        JButton button = 
        button.setBounds(

        button.addActionL
            @Override
```

```
run:
Picked up JAVA_TOOL_OPTIONS: -Dhttps.pr
Action Event Demo
Action Event Demo
Action Event Demo
```