POLITEKNIK STATISTIKA STIS

*For Better Official Statistics*

# Pemrograman Berorientasi Objek (PBO) – Pertemuan 9 (Teori)

oleh: Nano Yulian P.

Jakarta, 21-03-2022

## Bahasan materi

*Event Handling* di Java

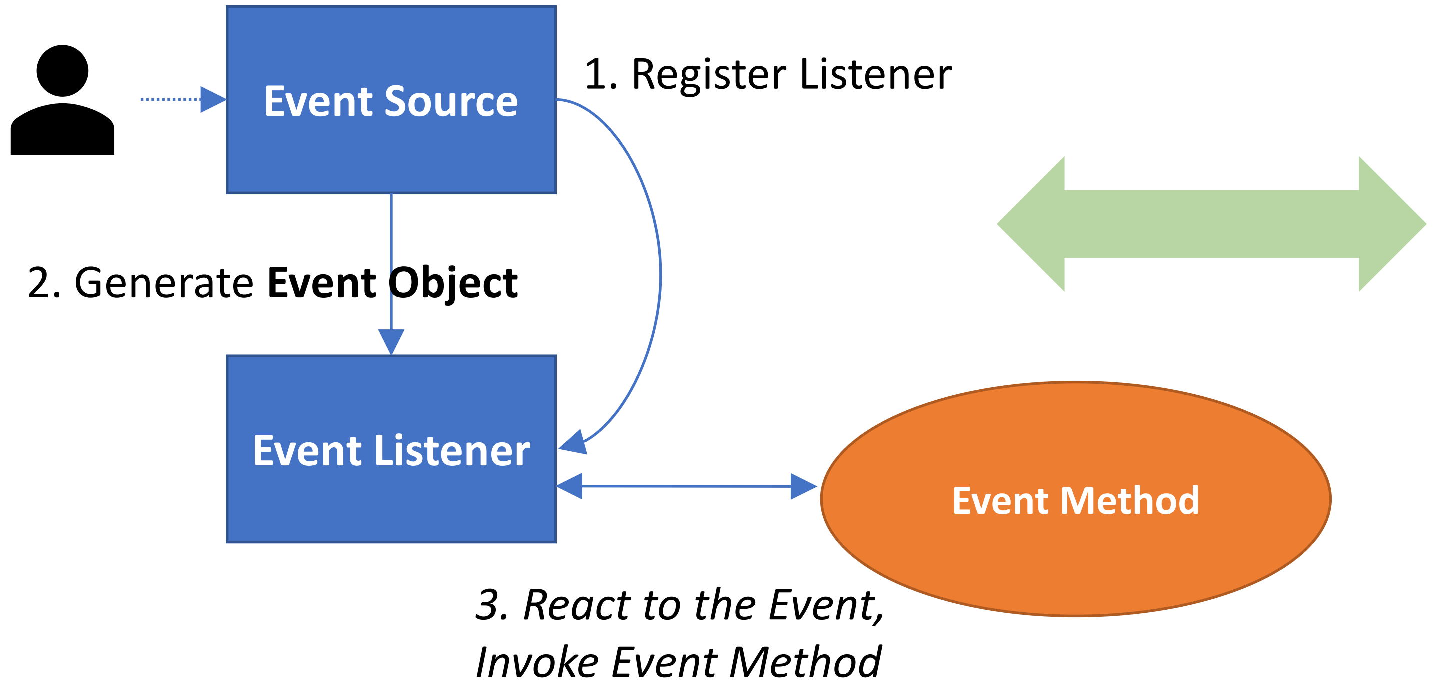
Java *Graphical User Interface (GUI)*

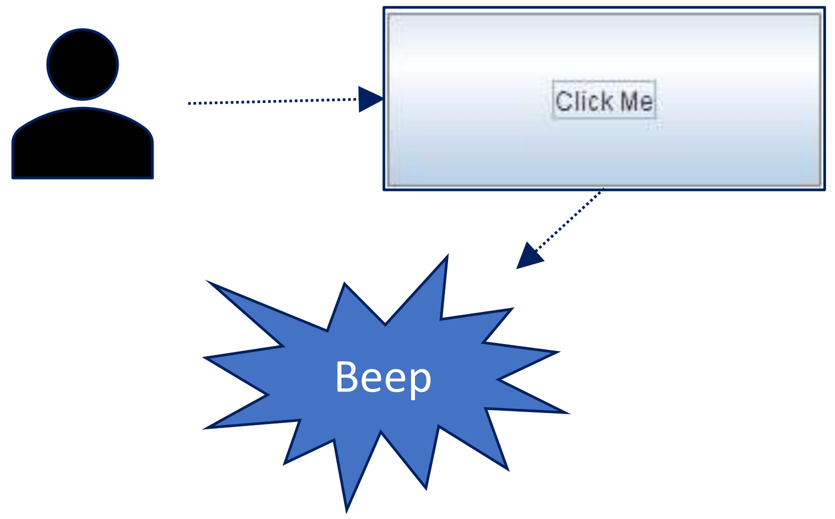
# *Event Handling (Event, Event Source, Event Listener)*

*Event* **adalah Peristiwa** Perubahan keadaan suatu objek, *terjadi krn Interaksi antara user dengan komponen GUI.*
*Button click, moving the mouse, enter character on keyboard, select item from list, scroll page, etc.*

## *Event Handling ?*

Mekanisme yang mengontrol suatu *event* dan menentukan apa yang harus dilakukan jika suatu *event* muncul.
Mekanisme ini digunakan untuk membuat program *event driven* seperti Applet, Aplikasi Java GUI, Aplikasi web.
Java menggunakan ***Delegation Event Model*** (menerapkan *observer pattern*) untuk meng*handle event*.

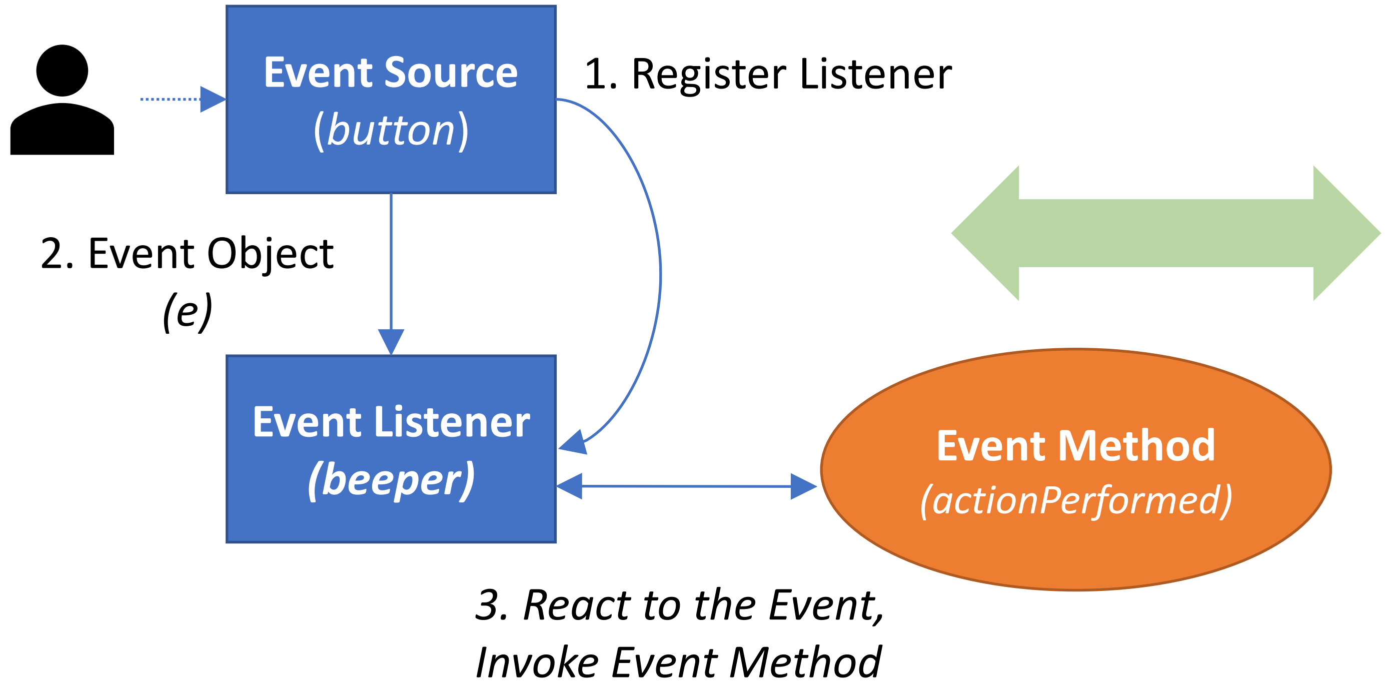

Event Source
1. Register Listener
2. Generate **Event Object**
Event Listener
Event Method
*3. React to the Event, Invoke Event Method*

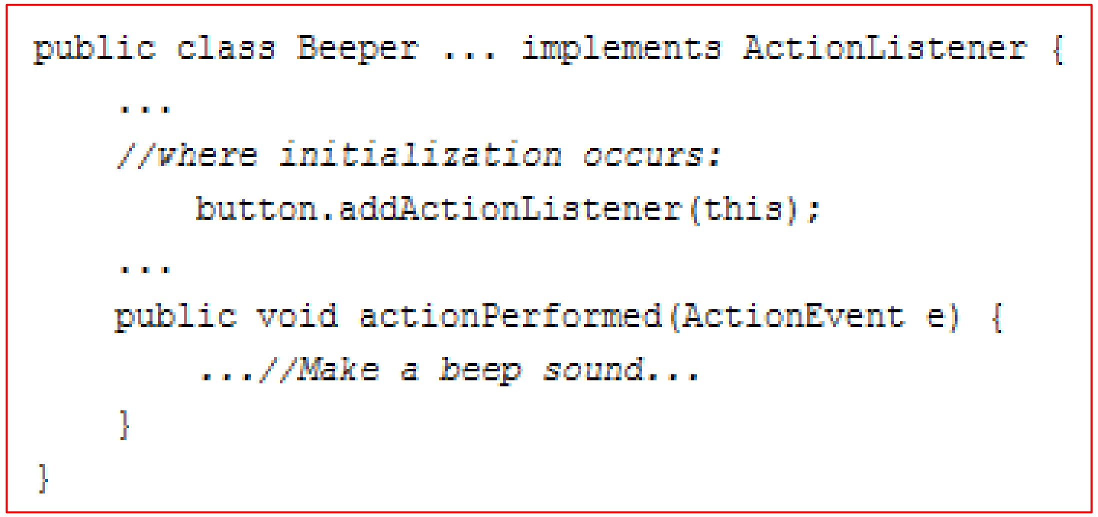


# *Event Handling (Event, Event Source, Event Listener)*

*Event source* **:** objek (*subyek/publisher*) yang statusnya (*state*) berubah. Menghasilkan *event*.
*Event object* **(Event) :** objek yang membungkus perubahan status pada *event source*.
*Event listener:* objek (*observer/subscriber*) yang akan menerima notifikasi.

"**Event source** mendelegasikan tugas untuk meng*handle* suatu ***event*** kepada ***event listener***, selanjutnya ***event listener*** akan meng-invoke *event method*. "

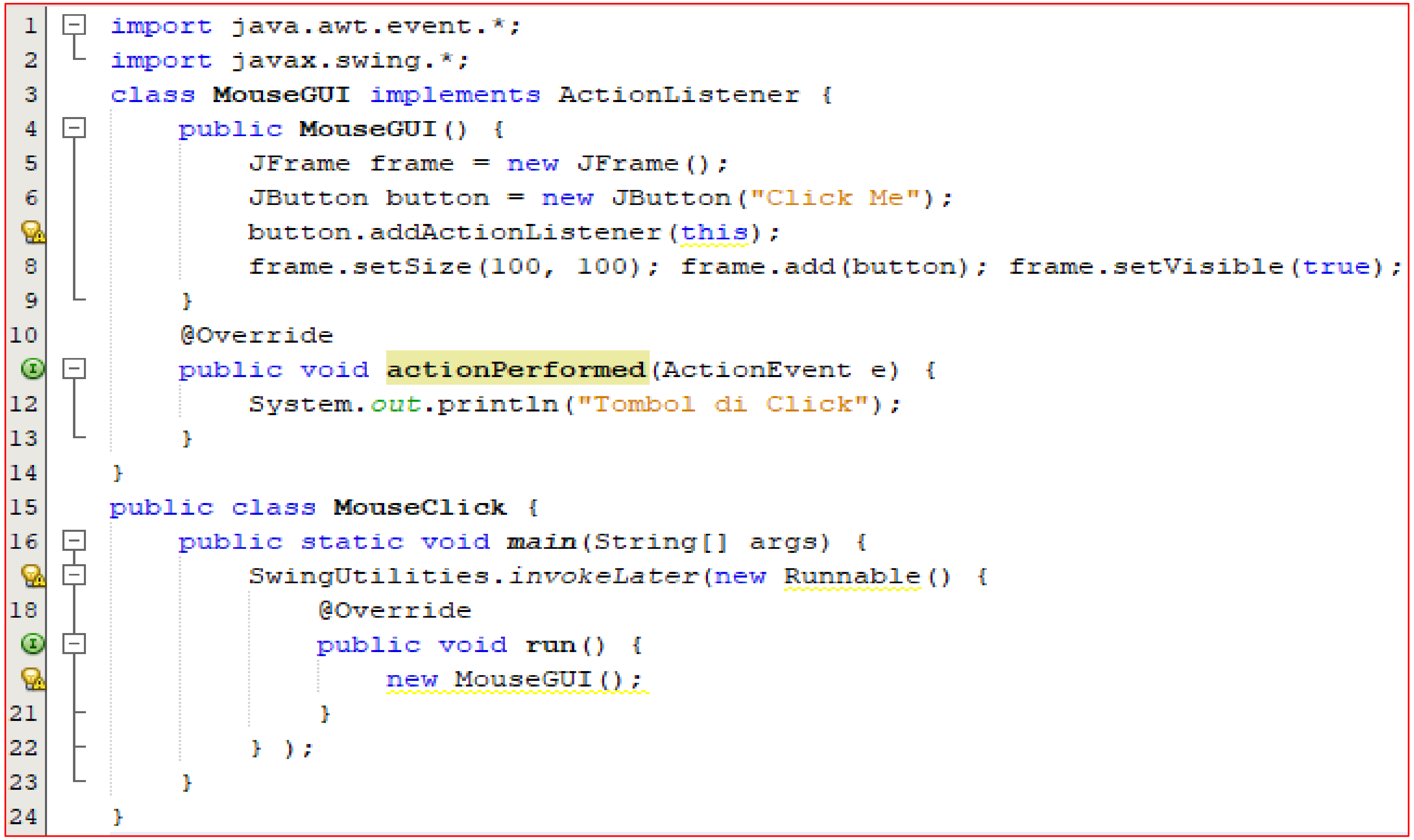




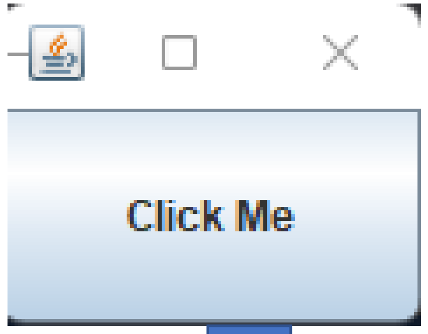
```
public class Beeper ... implements ActionListener {
    ...
    //where initialization occurs:
        button.addActionListener(this);
    ...
    public void actionPerformed(ActionEvent e) {
        ...//Make a beep sound...
    }
}
```

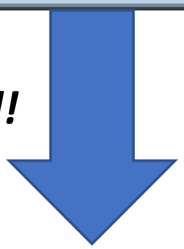
# Menulis program *event driven* di java

1. Menentukan komponen *control* Java sebagai *subyek/publisher*.
2. Menyiapkan objek ***listener*** dengan *meng-implement interface* yang sesuai agar bisa menerima tipe *event* yang diharapkan.
3. *Register* dan *unregister* ***listener*** (jika diperlukan) sebagai penerima notifikasi suatu *event*.
4. Menuliskan tugas yang harus dilakukan pada ***event method (Handler)***.

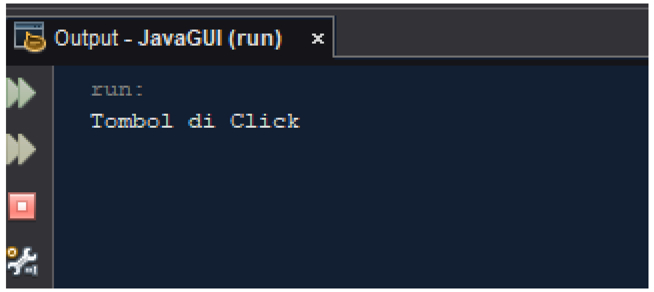


```java
import java.awt.event.*;
import javax.swing.*;
class MouseGUI implements ActionListener {
    public MouseGUI() {
        JFrame frame = new JFrame();
        JButton button = new JButton("Click Me");
        button.addActionListener(this);
        frame.setSize(100, 100); frame.add(button); frame.setVisible(true);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Tombol di Click");
    }
}
public class MouseClick {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new MouseGUI();
            }
        } );
    }
}
```





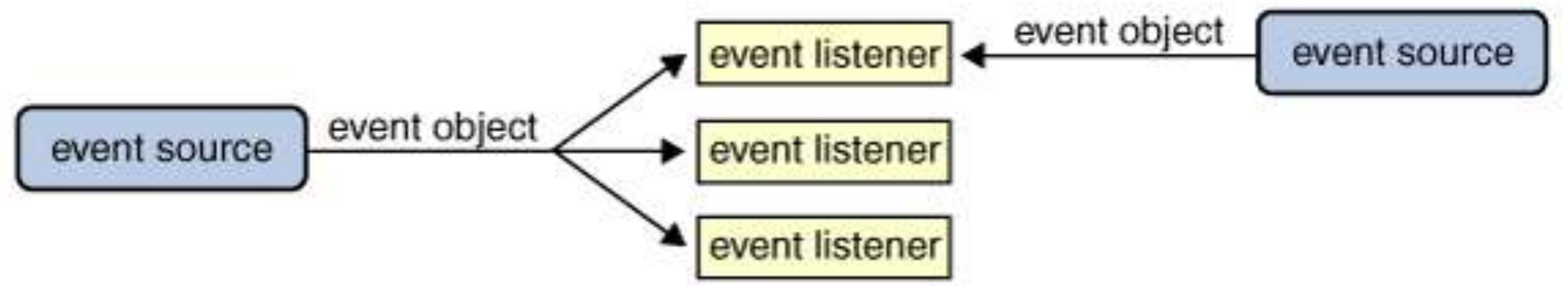
***Clicked!***



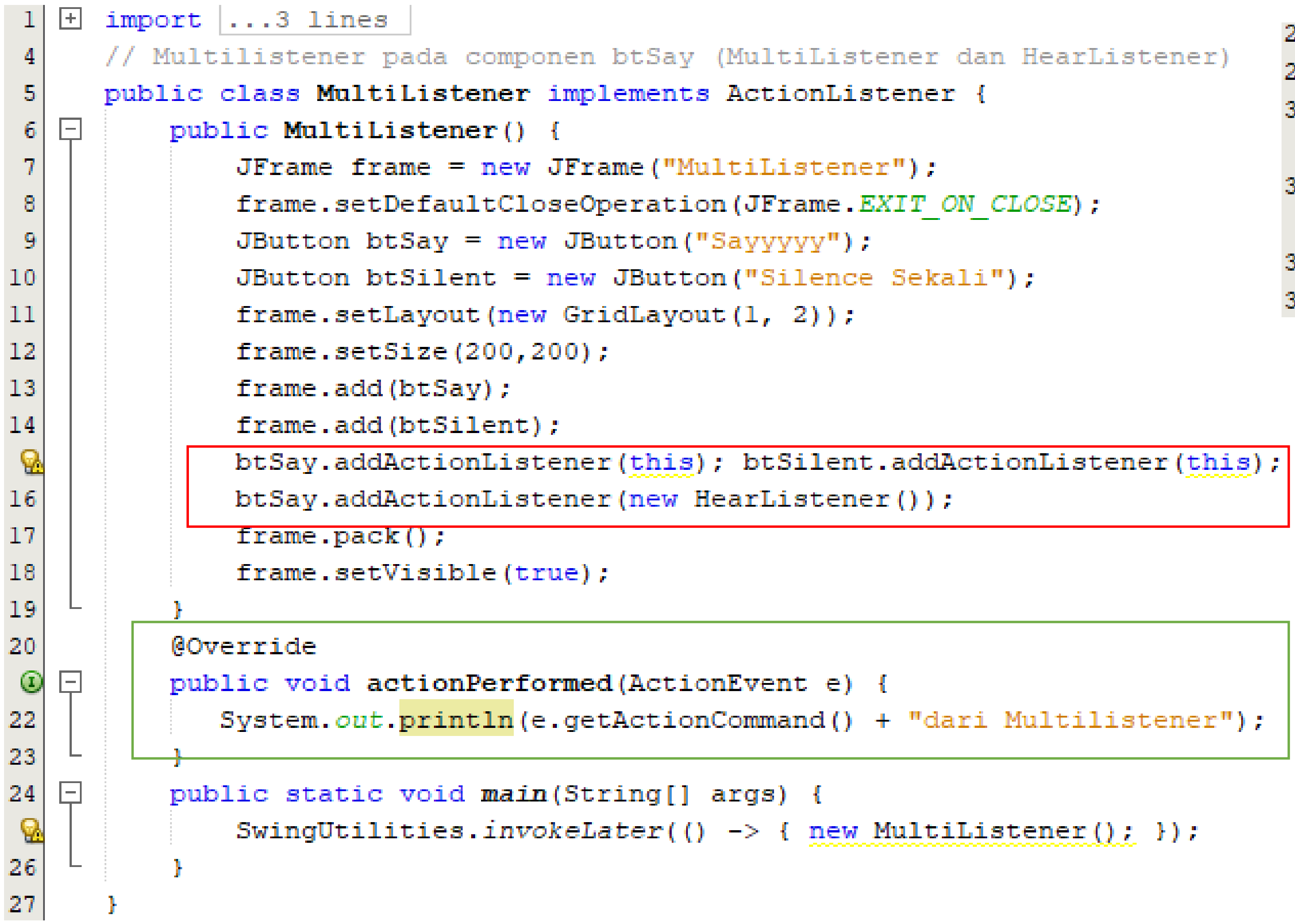
```
run:
Tombol di Click
```

# Menulis program *event driven* di java

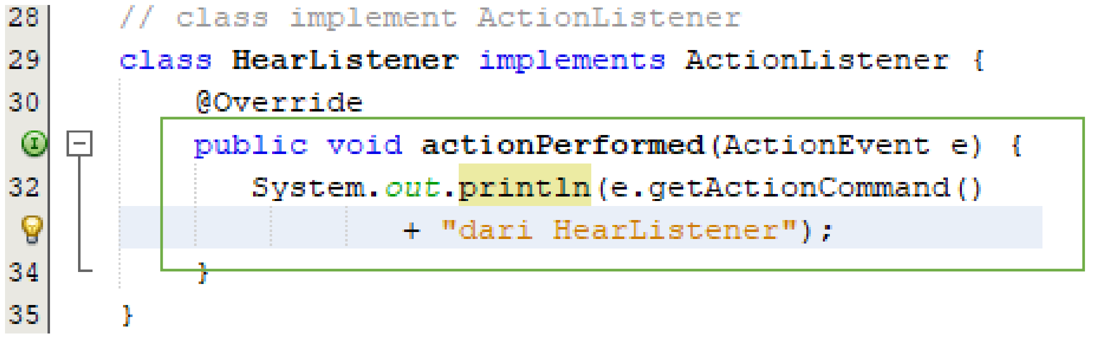
- **Note**: a **source object** can fire one or multiple events, and **one or multiple listeners can be registered by a source object**. Also, **a listener can declare one or multiple handlers**.
- For example, Java has provided standard models:
  - A **JCheckBox** can fire both **ActionEvent** and **ItemEvent** when the user checks or un-checks the checkbox. The corresponding listener interface is **ActionListener** (with the handler method **actionPerformed(ActionEvent e)**), and **ItemListener** (with the handler **itemStateChanged(ItemEvent e)**).
  - A Component **(JTextArea, or JLabel)** can fire **MouseEvent** or **KeyEvent** when the user has pressed, clicked, moved, exited mouse, pressed or typed key on/from that component. Corresponding listener interface is **MouseListener** (which defines several handlers, such as **mousePressed(MouseEvent e)**, and **mouseEntered(MouseEvent e)**), and **KeyListener** (with pre-defined handlers).



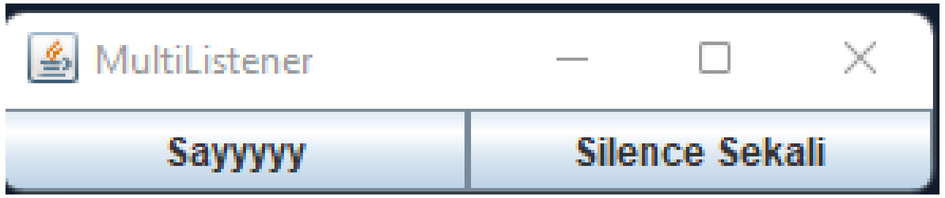
# MultiListener

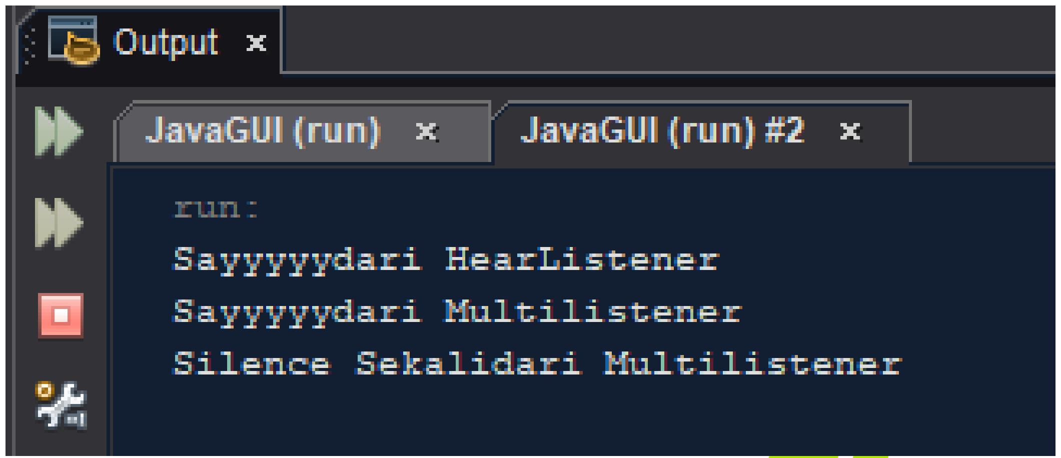


```java
import ...3 lines
// Multilistener pada componen btSay (MultiListener dan HearListener)
public class MultiListener implements ActionListener {
    public MultiListener() {
        JFrame frame = new JFrame("MultiListener");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton btSay = new JButton("Sayyyyy");
        JButton btSilent = new JButton("Silence Sekali");
        frame.setLayout(new GridLayout(1, 2));
        frame.setSize(200,200);
        frame.add(btSay);
        frame.add(btSilent);
        btSay.addActionListener(this); btSilent.addActionListener(this);
        btSay.addActionListener(new HearListener());
        frame.pack();
        frame.setVisible(true);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
       System.out.println(e.getActionCommand() + "dari Multilistener");
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> { new MultiListener(); });
    }
}
```



```java
// class implement ActionListener
class HearListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println(e.getActionCommand()
                + "dari HearListener");
    }
}
```



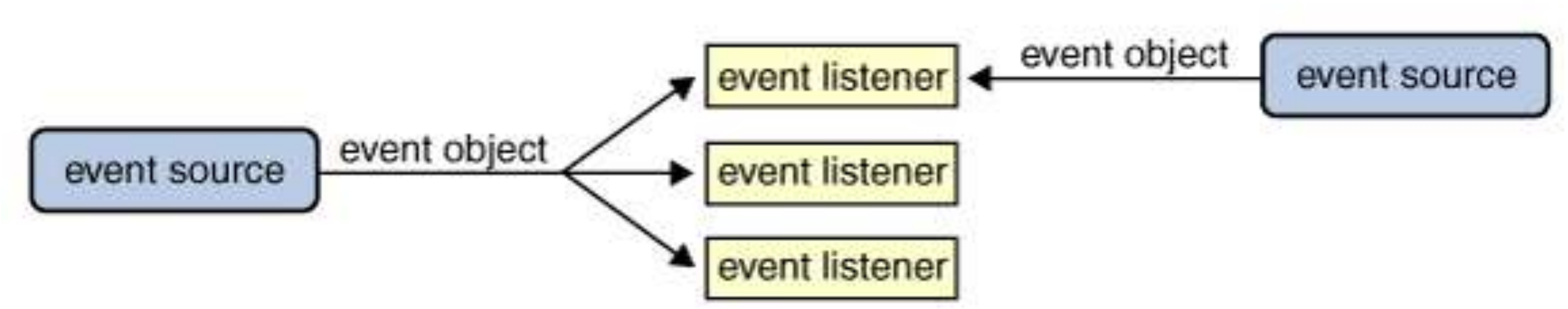


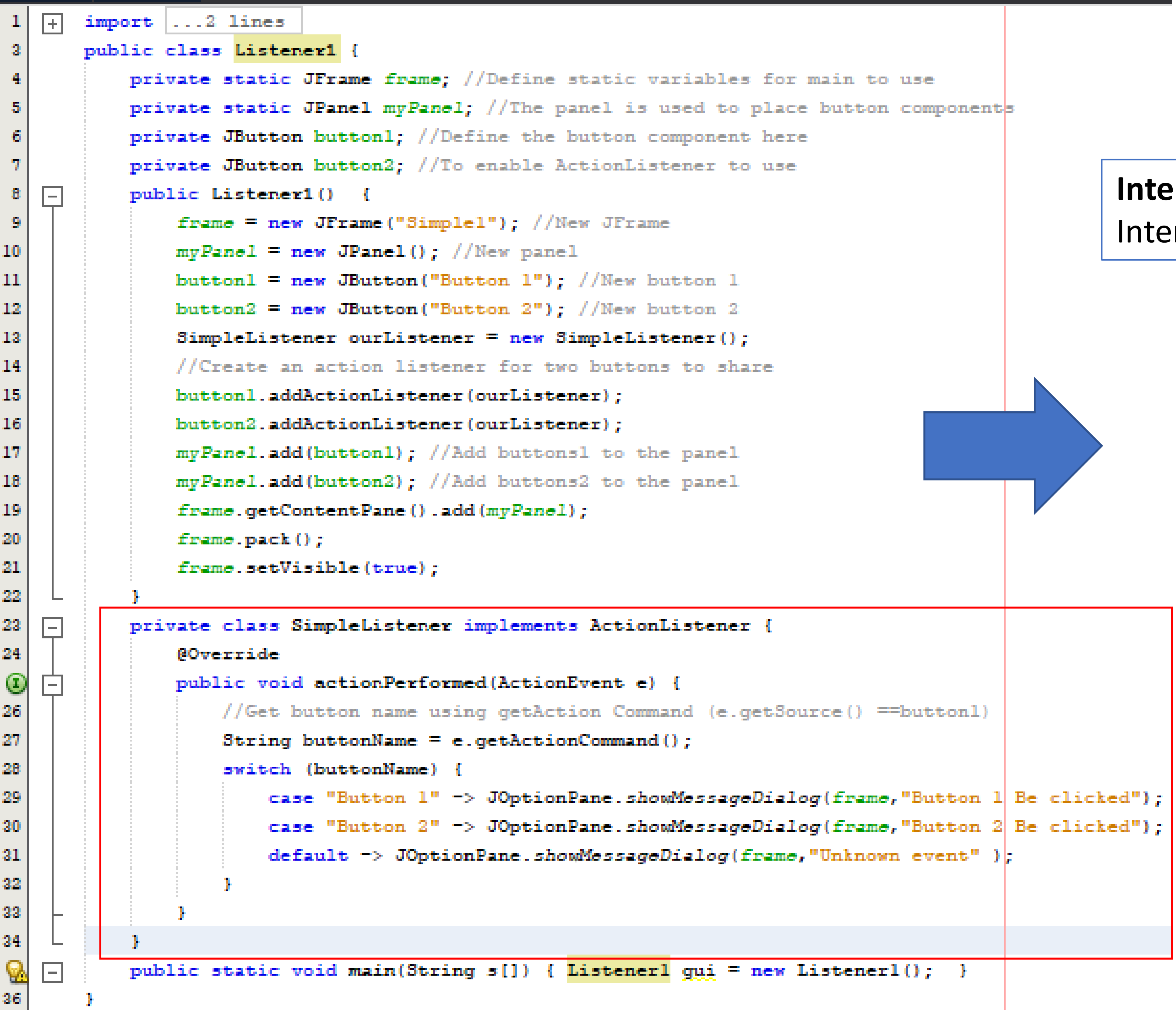
```
run:
Sayyyyydari HearListener
Sayyyyydari Multilistener
Silence Sekalidari Multilistener
```

# Cara lain meng*implements* Listener

Terdapat Beberapa Metode dalam meng-*implements* Listener diantaranya :

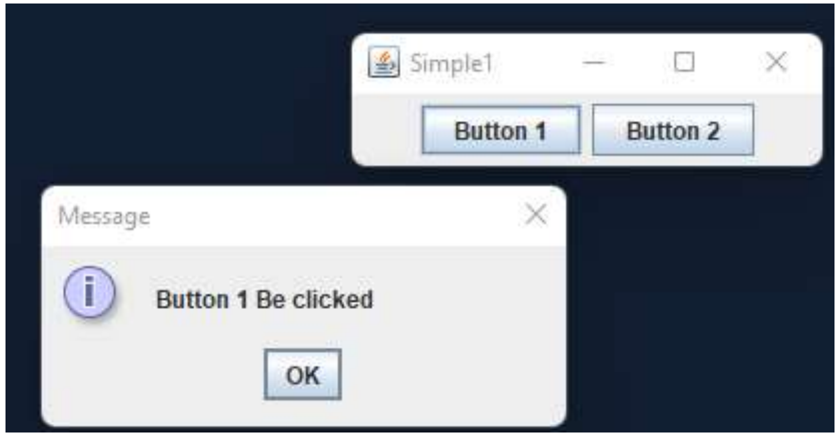
1. ***Internal Class*** yang meng*implements Listener Interface* untuk semua ***event source*** yang ada (ex: ActionListener). => ( ***e.getSource()*** atau ***e.getActionCommand()*** )
2. ***Anonymous inner classes***, digunakan sebagai parameter variable pada saat register Listener (ex: addActionListener)
3. ***Internal classes*** yang meng*implements Listener Interface* untuk masing-masing ***event source*** yang ada (ex: ActionListener).



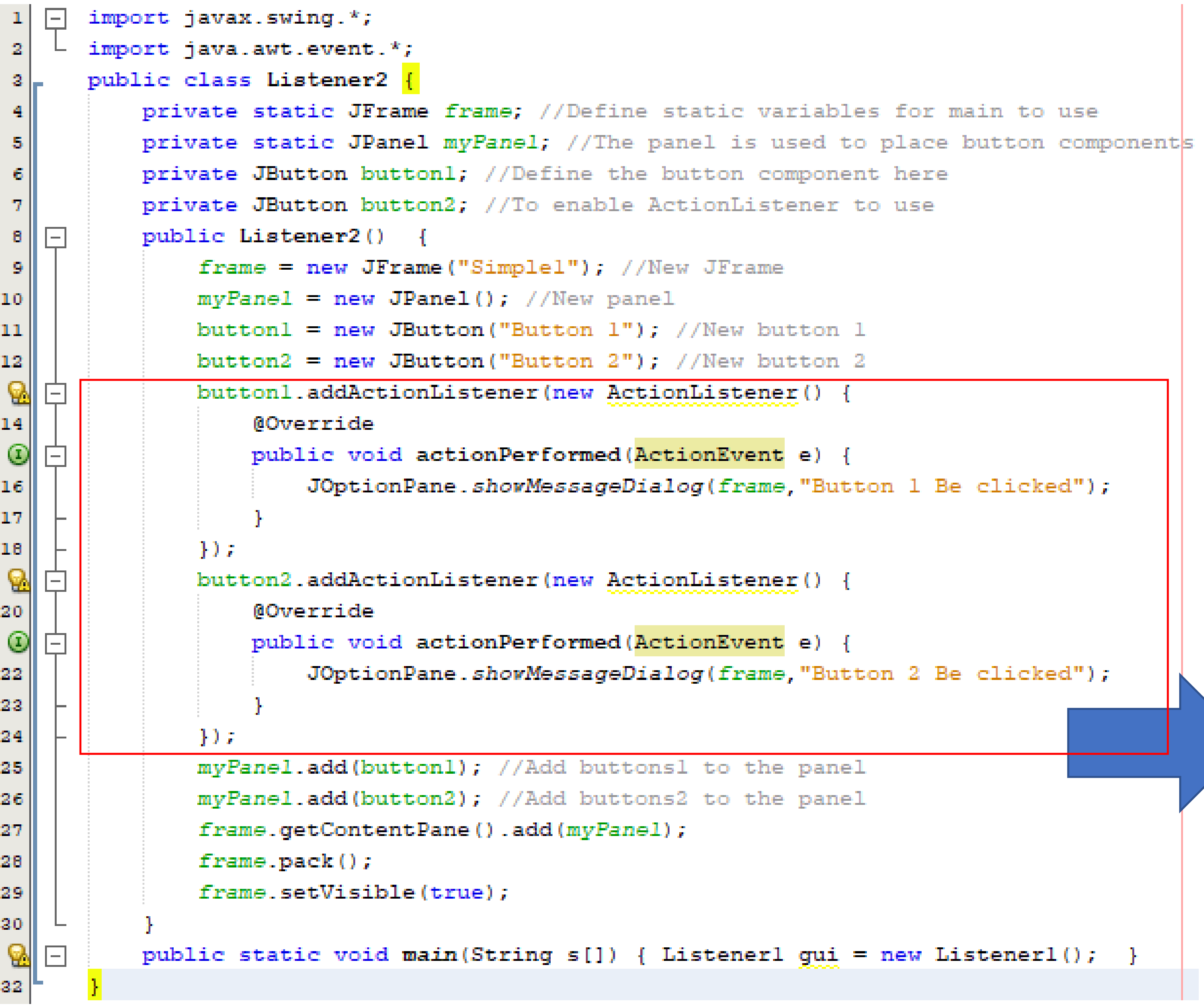


```java
import ...2 lines
public class Listener1 {
    private static JFrame frame; //Define static variables for main to use
    private static JPanel myPanel; //The panel is used to place button components
    private JButton button1; //Define the button component here
    private JButton button2; //To enable ActionListener to use
    public Listener1()  {
        frame = new JFrame("Simple1"); //New JFrame
        myPanel = new JPanel(); //New panel
        button1 = new JButton("Button 1"); //New button 1
        button2 = new JButton("Button 2"); //New button 2
        SimpleListener ourListener = new SimpleListener();
        //Create an action listener for two buttons to share
        button1.addActionListener(ourListener);
        button2.addActionListener(ourListener);
        myPanel.add(button1); //Add buttons1 to the panel
        myPanel.add(button2); //Add buttons2 to the panel
        frame.getContentPane().add(myPanel);
        frame.pack();
        frame.setVisible(true);
    }
    private class SimpleListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            //Get button name using getAction Command (e.getSource() ==button1)
            String buttonName = e.getActionCommand();
            switch (buttonName) {
                case "Button 1" -> JOptionPane.showMessageDialog(frame,"Button 1 Be clicked");
                case "Button 2" -> JOptionPane.showMessageDialog(frame,"Button 2 Be clicked");
                default -> JOptionPane.showMessageDialog(frame,"Unknown event" );
            }
        }
    }
    public static void main(String s[]) { Listener1 gui = new Listener1();  }
}
```

**Internal Class** yang meng*implements* Listener Interface untuk semua event source yang ada

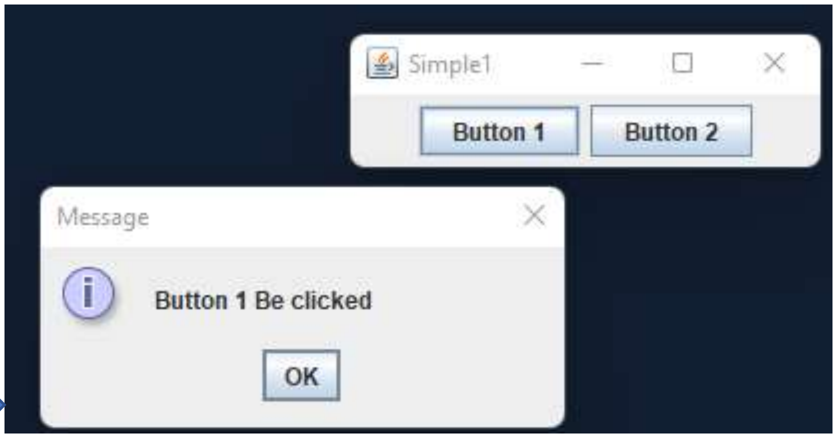


```
Button 1 Be clicked
```



```java
import javax.swing.*;
import java.awt.event.*;
public class Listener2 {
    private static JFrame frame; //Define static variables for main to use
    private static JPanel myPanel; //The panel is used to place button components
    private JButton button1; //Define the button component here
    private JButton button2; //To enable ActionListener to use
    public Listener2()  {
        frame = new JFrame("Simple1"); //New JFrame
        myPanel = new JPanel(); //New panel
        button1 = new JButton("Button 1"); //New button 1
        button2 = new JButton("Button 2"); //New button 2
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame,"Button 1 Be clicked");
            }
        });
        button2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame,"Button 2 Be clicked");
            }
        });
        myPanel.add(button1); //Add buttons1 to the panel
        myPanel.add(button2); //Add buttons2 to the panel
        frame.getContentPane().add(myPanel);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String s[]) { Listener1 gui = new Listener1();  }
}
```

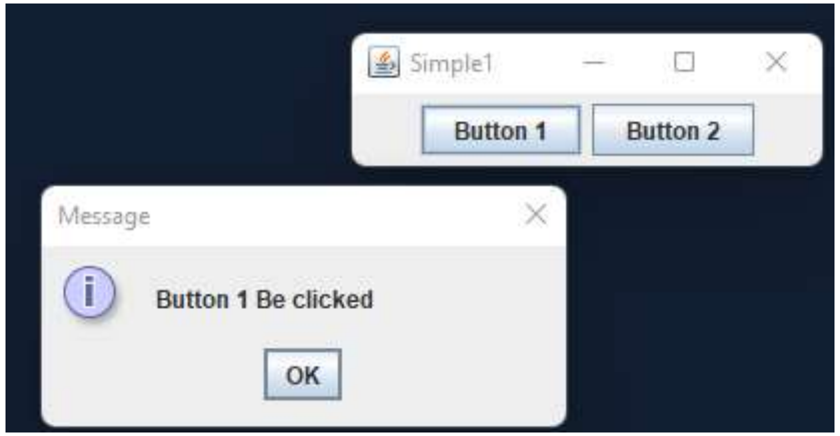
***Anonymous inner classes***, digunakan sebagai parameter variable pada saat menggunakan **method register Listener.**



Button 1 Be clicked

```java
public class Listener3 {
    private static JFrame frame; //Define static variables for main to use
    private static JPanel myPanel; //The panel is used to place button component
    private JButton button1; //Define the button component here
    private JButton button2; //To enable ActionListener to use
    public Listener3()  {
        frame = new JFrame("Simple1"); //New JFrame
        myPanel = new JPanel(); //New panel
        button1 = new JButton("Button 1"); //New button 1
        button2 = new JButton("Button 2"); //New button 2
        button1.addActionListener(new Button1Listener());
        button2.addActionListener(new Button2Listener());
        myPanel.add(button1); //Add buttons1 to the panel
        myPanel.add(button2); //Add buttons2 to the panel
        frame.getContentPane().add(myPanel);
        frame.pack();
        frame.setVisible(true);
    }
    private class Button1Listener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame,"Button 1 Be clicked");
        }
    }
    private class Button2Listener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame,"Button 2 Be clicked");
        }
    }
    public static void main(String s[]) { Listener1 gui = new Listener1();  }
}
```

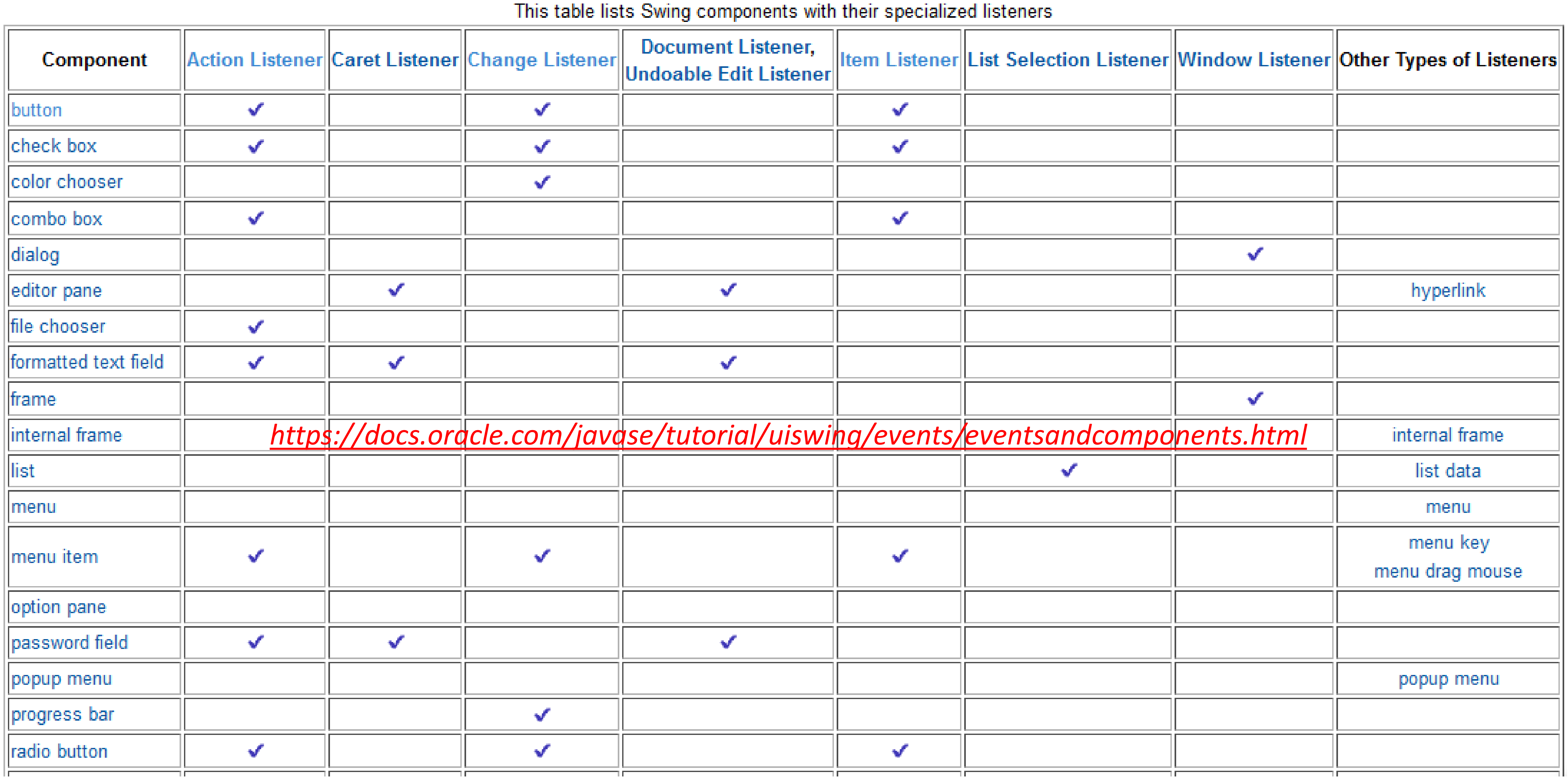
***Internal classes*** yang meng*implements Listener Interface* untuk masing-masing ***event source*** yang ada



Button 1 Be clicked

# Komponen Swing dan Default Listener Terkait

For Better Official Statistics



This table lists Swing components with their specialized listeners

| Component | Action Listener | Caret Listener | Change Listener | Document Listener, Undoable Edit Listener | Item Listener | List Selection Listener | Window Listener | Other Types of Listeners |
|---|---|---|---|---|---|---|---|---|
| button | ✓ | | ✓ | | ✓ | | | |
| check box | ✓ | | ✓ | | ✓ | | | |
| color chooser | | | ✓ | | | | | |
| combo box | ✓ | | | | ✓ | | | |
| dialog | | | | | | | ✓ | |
| editor pane | | ✓ | | ✓ | | | | hyperlink |
| file chooser | ✓ | | | | | | | |
| formatted text field | ✓ | ✓ | | ✓ | | | | |
| frame | | | | | | | ✓ | |
| internal frame | | | | | | | | internal frame |
| list | | | | | | ✓ | | list data |
| menu | | | | | | | | menu |
| menu item | ✓ | | ✓ | | ✓ | | | menu key<br>menu drag mouse |
| option pane | | | | | | | | |
| password field | ✓ | ✓ | | ✓ | | | | |
| popup menu | | | | | | | | popup menu |
| progress bar | | | ✓ | | | | | |
| radio button | ✓ | | ✓ | | ✓ | | | |

https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html

# Event Listener Rule… Event Dispatch Thread (EDT)

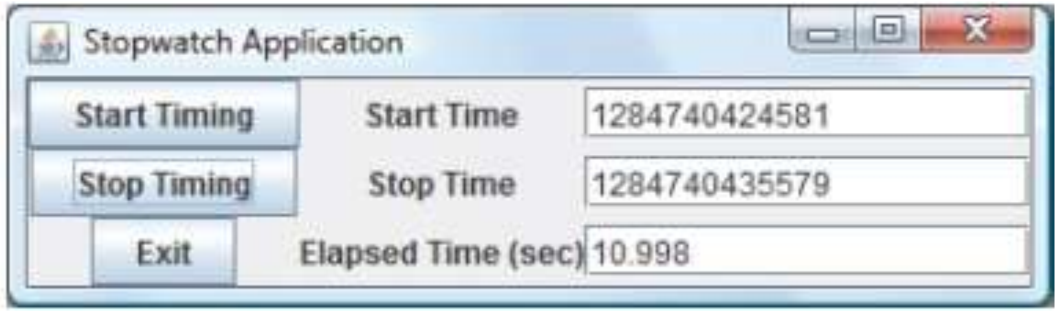
A Swing programmer deals with the following kinds of threads:

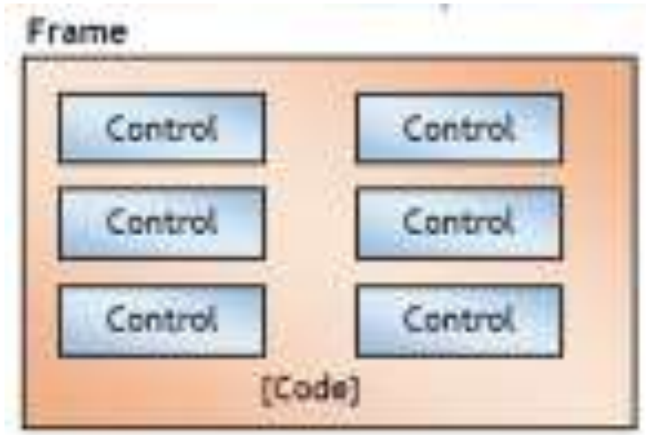
1. *Initial threads*, the threads that execute initial application code.
2. The *event dispatch thread*, **where all event-handling code is executed**. **Most code that interacts with the Swing framework must also execute on this thread.**
3. *Worker threads*, also known as *background threads*, where time-consuming background tasks are executed.

***The programmer does not need to provide code that explicitly creates these threads: they are provided by the runtime or the Swing framework.***
***The programmer's job is to utilize these threads to create a responsive, maintainable Swing program.***

The most important rule to keep in mind **about event listeners is that they should execute very quickly.** Because **all drawing and event-listening methods are executed in the same thread => EDT**, a slow event-listener method can **make the program seem unresponsive and slow to repaint itself**. If you need to perform some lengthy operation as the result of an event, do it **by starting up another thread** (or somehow sending a request to another thread) to perform the operation.

## Struktur Aplikasi Java GUI





*.addActionListener*

Sebuah Aplikasi Java GUI terdiri dari :

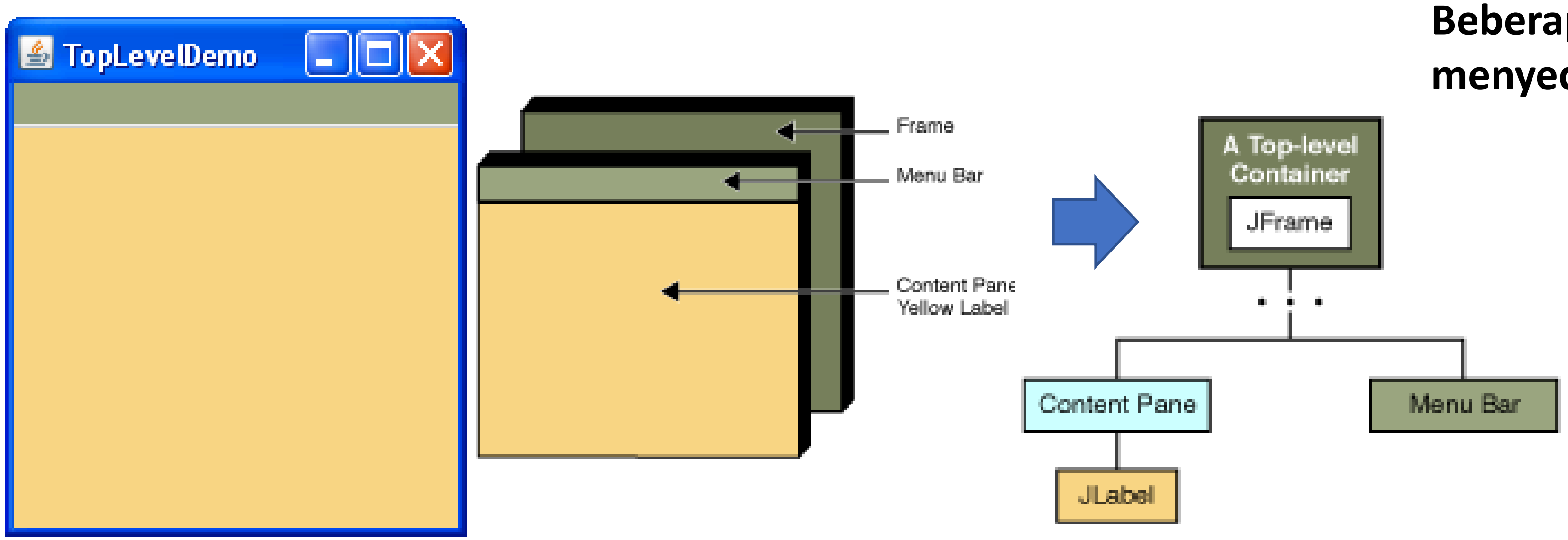
- ***Frame/Container****:* Komponen GUI untuk menampung komponen GUI lain diatasnya. *(ex : JPanel, JFrame, JWindow)*
- ***Controls*** *: komponen Java* GUI yang berada di atas *frame/container,* dimana ***user*** biasanya berinteraksi dengannya .
  - ( *ex: JTextField, JLabel, JButton, JComboBox,* dll)
- ***Frame/Container*** dan ***Controls*** adalah Objek.
- ***Properties*** **:** Menentukan kondisi (*state)* dari sebuah *frame* atau *control*. (*names, caption, size, color, position, dan content*).
- ***Methods*** **:** *built-in procedure* yang bisa dipanggil untuk melakukan sebuah aksi untuk merubah atau menentukan *properties* dari objek tertentu.
- ***Event Methods*****:** *Method* yang berkaitan dengan objek atau *control* tertentu. *Method* ini akan dieksekusi Ketika suatu event terjadi.

2 (dua) ***package*** utama yang biasa digunakan untuk membuat aplikasi Java GUI yaitu ***Abstract Windowing Toolkit (AWT)*** *dan* ***Swing.***

## Menggunakan *Top-Level Container* (JFrame)

Swing menyediakan **3 top-level container class** diantaranya : **JFrame, JDialog, dan JApplet**. Yang Harus diingat :

1. Untuk tampil dilayar, setiap komponen GUI harus menjadi bagian dari struktur **dimana top-level container menjadi lapisan dasarnya.**
2. Setiap komponen GUI hanya bisa dimuat satu kali saja dalam container.
3. Setiap top-level container memiliki **Content Pane** . Setiap **Content Pane** menggunakan **Layout Manager (*bisa menggunakan nested layout)***.
4. Bisa menambahkan menu bar pada top-level container (tetapi diluar content pane).



**Beberapa AWT dan Swing class yang menyediakan Layout Manager**

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

```
JFrame frame = new JFrame();
LayoutManager m = frame.getContentPane().getLayout();
System.out.println(m instanceof BorderLayout); // prints true
```

```
JFrame frame = new JFrame("Contoh JFrame");
frame.setLayout(new GridLayout(2,3));
frame.setSize(200,200);
frame.setVisible(true);
```

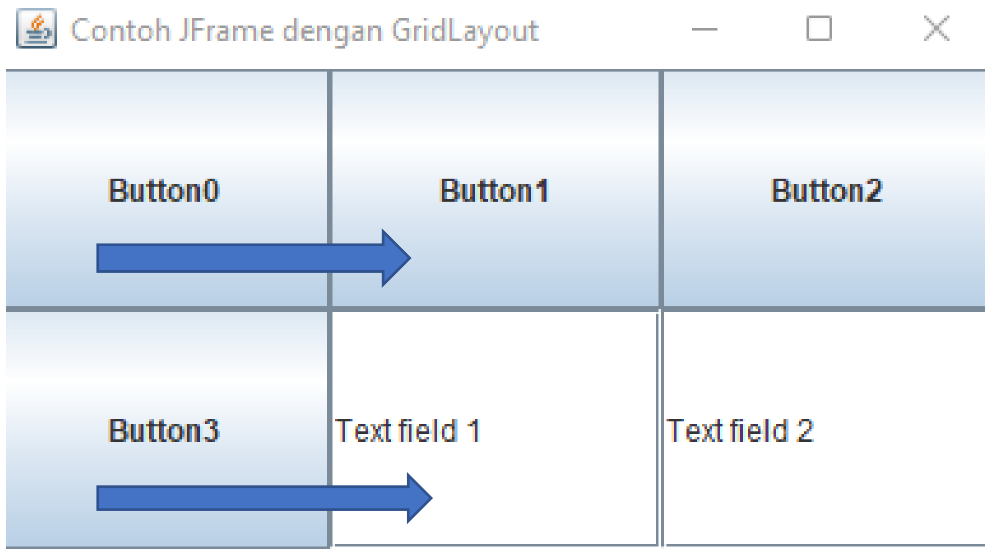
Source : https://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html#contentpane

# Layout Manager => GridLayout

- ✓ Layout dalam bentuk grid (terdiri dari cell2x)
- ✓ 1 cell hanya memuat 1 komponen GUI dan width dan height komponen akan mengikuti width dan height cell.
- ✓ Setiap komponen memiliki width dan height yang sama di setiap cell

***new GridLayout***(int ***rows***, int ***cols***)

```java
import java.awt.GridLayout;
import javax.swing.*;
public class JFrameLayout {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Contoh JFrame dengan GridLayout");
            frame.setLayout(new GridLayout(2,3));
            frame.setSize(200,200);
            JButton[] button = new JButton[4];
            JTextField txt1 = new JTextField("Text field 1", 10);
            JTextField txt2 = new JTextField("Text field 2", 10);
            for (int i=0;i<button.length;i++) {
                button[i] = new JButton("Button"+i);
                frame.add(button[i]);
            }
            frame.add(txt1); frame.add(txt2);
            frame.setVisible(true);
        });
    }
}
```



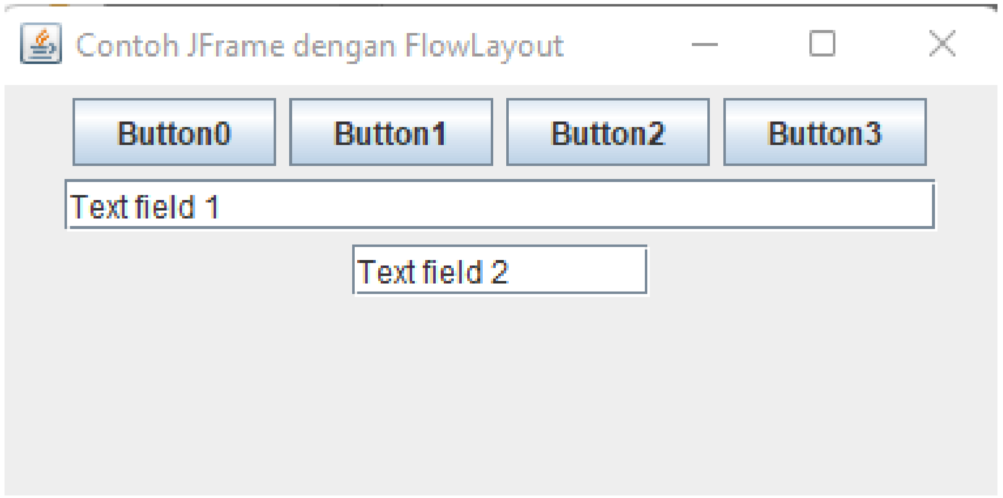
## Layout Manager => FlowLayout

- ✓ Setiap komponen control ditambahkan mulai dari kiri ke kanan (baris).
- ✓ Layout bergantung terhadap width dan height dari Containernya
- ✓ Jika width komponen > width frame, maka komponen berikutnya akan berada di baris berikutnya.

***new FlowLayout()***
***new FlowLayout(int alignment)***

***Default : FlowLayout.CENTER***

```java
import ...6 lines
public class JFrameFlowLayout {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Contoh JFrame dengan FlowLayout");
            frame.setLayout(new FlowLayout());
            frame.setSize(400,200);
            JButton[] button = new JButton[4];
            JTextField txt1 = new JTextField("Text field 1", 30);
            JTextField txt2 = new JTextField("Text field 2", 10);
            for (int i=0;i<button.length;i++) {
                button[i] = new JButton("Button"+i);
                frame.add(button[i]);
            }
            frame.add(txt1); frame.add(txt2);
            frame.setVisible(true);
        });
    }
}
```



# Layout Manager => GridBagLayout

- ✓ **Layout paling kompleks tetapi fleksibel**
- ✓ **Menempatkan komponen dalam grid row dan column,**
- ✓ **Mengizinkan komponen menggunakan multiple rows atau columns (span)**
- ✓ **Rows dan Columns tidak harus memiliki width dan height yang sama seperti GridLayout.**
- ✓ **Size dari komponen yang akan mempengaruhi size dari cell pada grid.**
- ✓ **Menggunakan objek GridBagConstrainsts utk menentukan *size dan position* dari tiap komponen.**

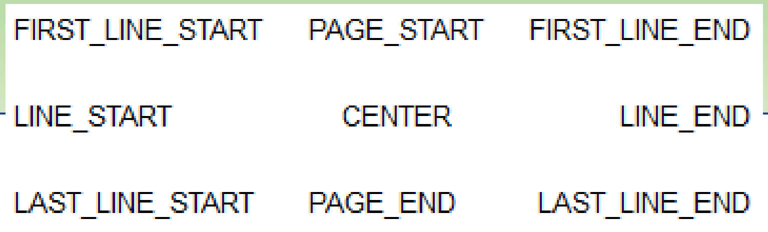
```
JPanel pane = new JPanel(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();

//For each component to be added to this container:
//...Create the component...
//...Set instance variables in the GridBagConstraints instance...
pane.add(theComponent, c);
```

***Strategi Penambahan komponen control pada GridBagLayout :***

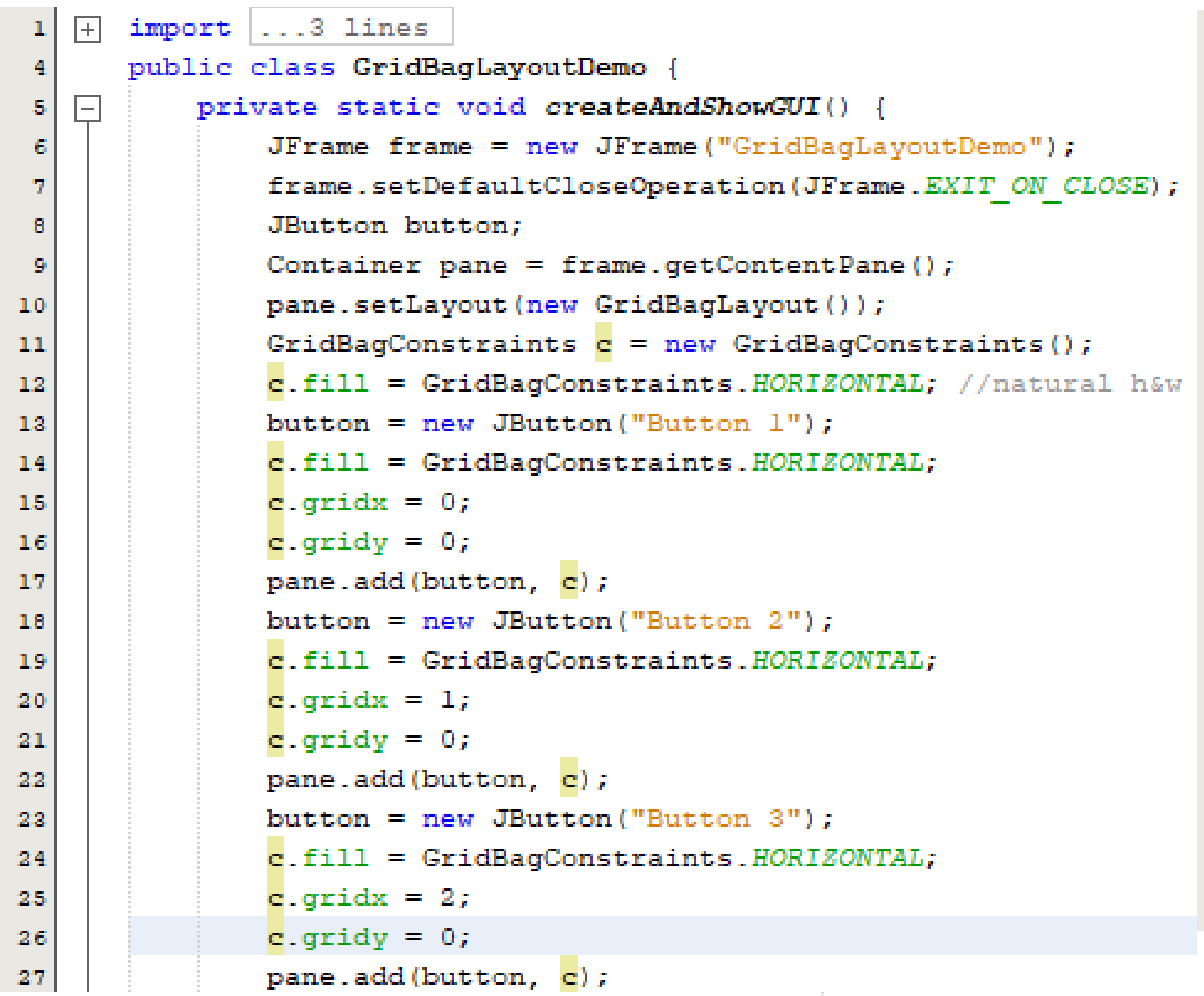
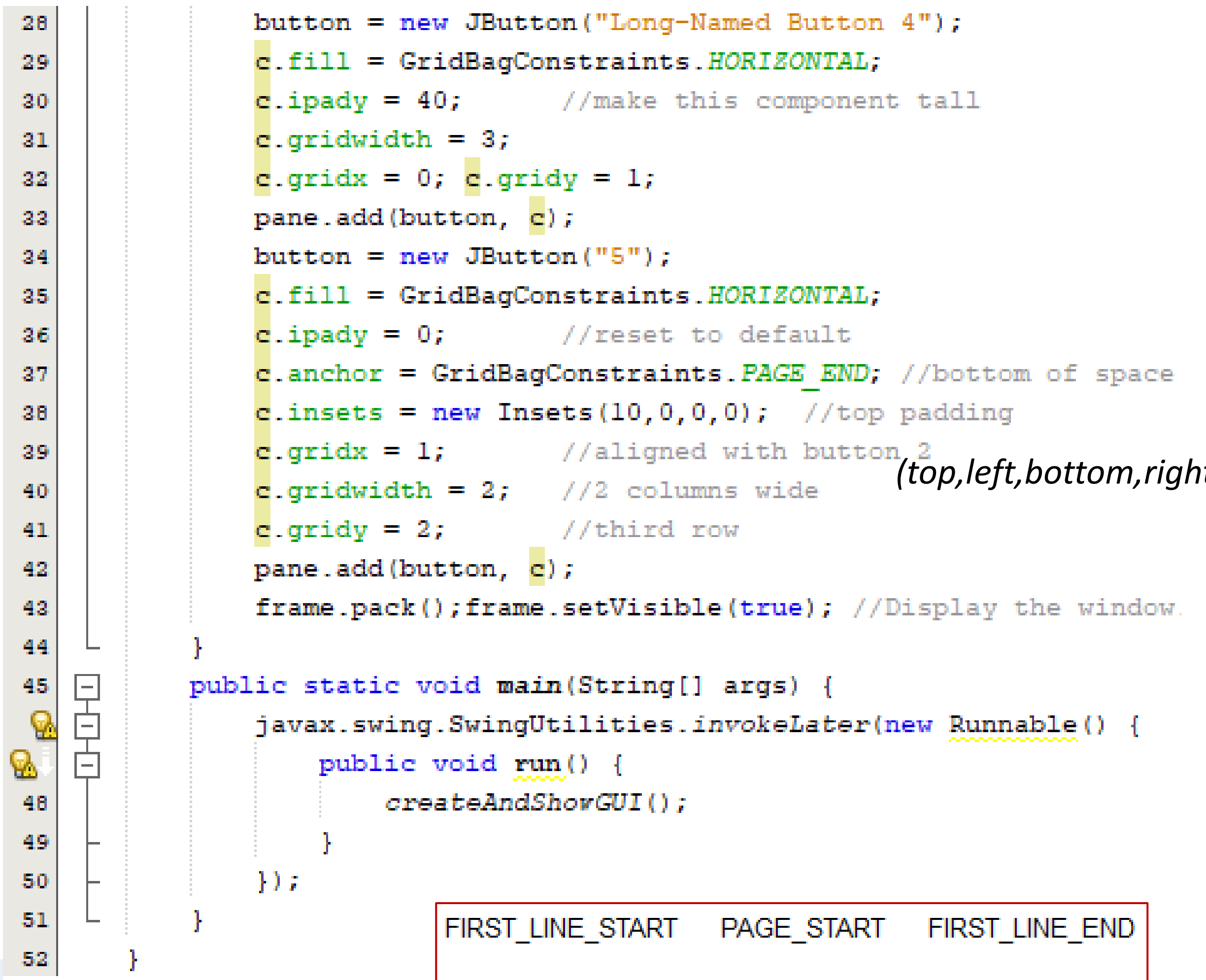
- add semua control pada Kol. pertama dulu ( gridx = 0, gridy=0,1,dst)
- lanjutkan untuk kolom berikutnya (gridx=1, gridy=0,1,dst) dst.
- Jika diperlukan bisa menambahkan panel dengan layout gridlayout untuk menambahkan beberapa control lainnya.

**Properties Constraint :**

- **gridx,gridy** *(upper left top row gridx=0, gridy=0)*
- **gridwidth,gridheight** *(specify num of cols or rows, def=1)*
- **Fill** *(jika size cell area > dari size komponen, tentukan Bagaimana size komponen tsb terhadap area cell)*
- **ipadx, ipady** *(menentukan internal padding, def = 0)*
- **Insets (***menentukan ext.padding komponen*) (top,left,bottom,right)
- **Anchor** *(digunakan utk menentukan posisi komponen pada cell area, def=CENTER)*



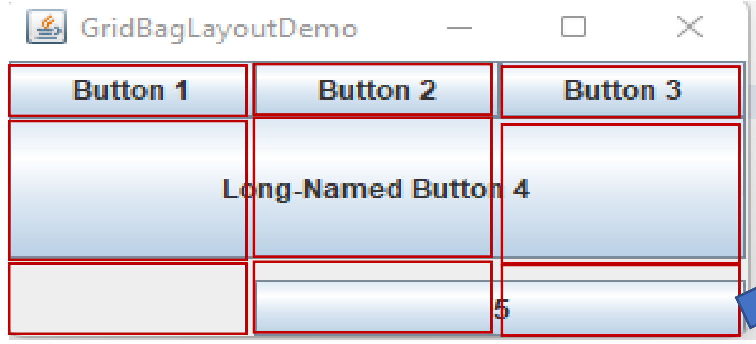
| | | |
|---|---|---|
| FIRST_LINE_START | PAGE_START | FIRST_LINE_END |
| LINE_START | CENTER | LINE_END |
| LAST_LINE_START | PAGE_END | LAST_LINE_END |

```java
import ...3 lines
public class GridBagLayoutDemo {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("GridBagLayoutDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button;
        Container pane = frame.getContentPane();
        pane.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL; //natural h&w
        button = new JButton("Button 1");
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = 0;
        c.gridy = 0;
        pane.add(button, c);
        button = new JButton("Button 2");
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = 1;
        c.gridy = 0;
        pane.add(button, c);
        button = new JButton("Button 3");
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = 2;
        c.gridy = 0;
        pane.add(button, c);
        button = new JButton("Long-Named Button 4");
        c.fill = GridBagConstraints.HORIZONTAL;
        c.ipady = 40;       //make this component tall
        c.gridwidth = 3;
        c.gridx = 0; c.gridy = 1;
        pane.add(button, c);
        button = new JButton("5");
        c.fill = GridBagConstraints.HORIZONTAL;
        c.ipady = 0;        //reset to default
        c.anchor = GridBagConstraints.PAGE_END; //bottom of space
        c.insets = new Insets(10,0,0,0);  //top padding
        c.gridx = 1;        //aligned with button 2
        c.gridwidth = 2;    //2 columns wide
        c.gridy = 2;        //third row
        pane.add(button, c);
        frame.pack();frame.setVisible(true); //Display the window
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

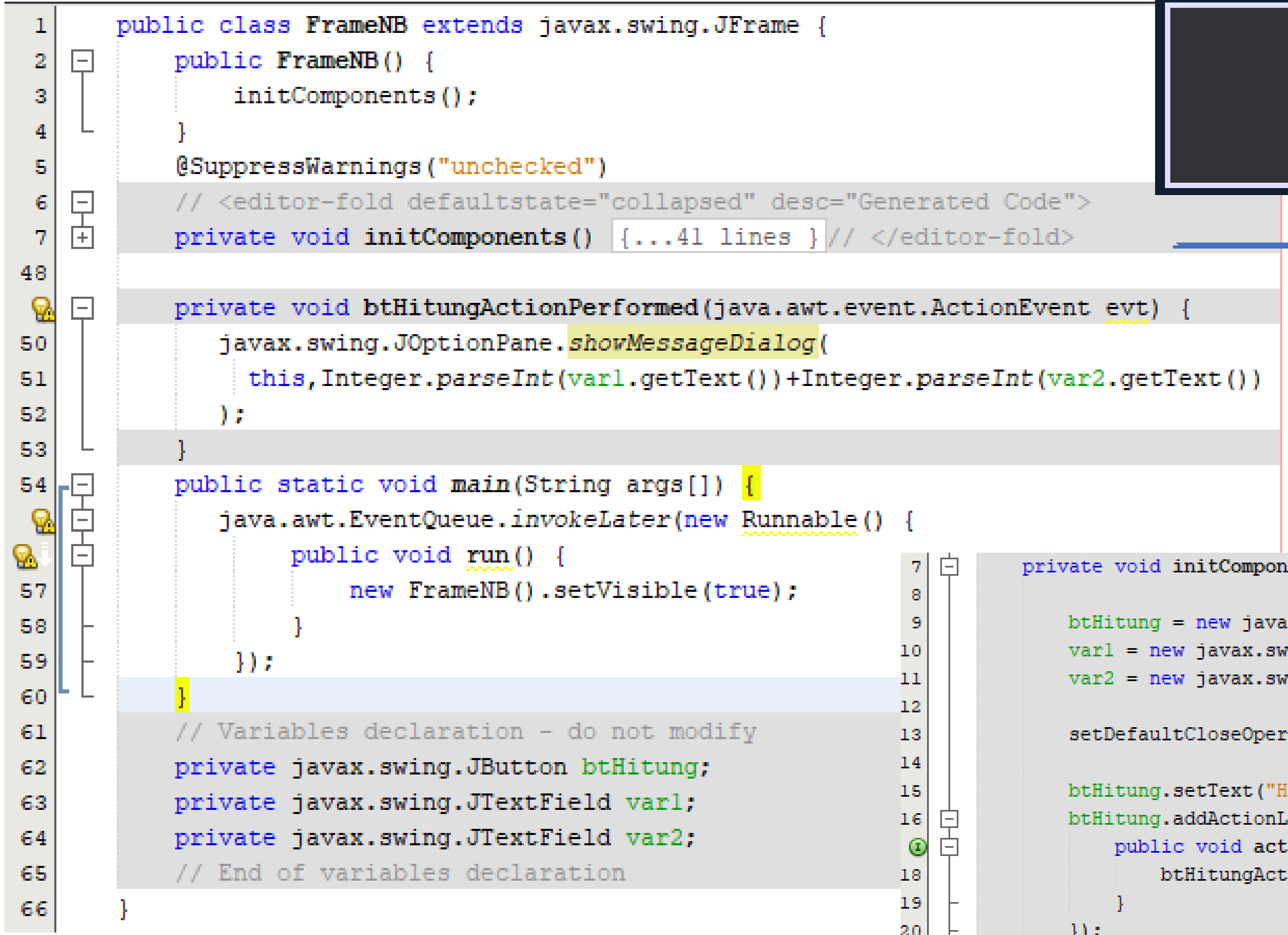
*(top,left,bottom,right)*

3 Kolom

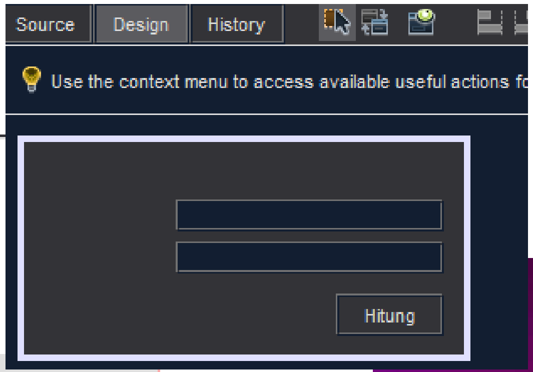


| | | |
|---|---|---|
| FIRST_LINE_START | PAGE_START | FIRST_LINE_END |
| LINE_START | CENTER | LINE_END |
| LAST_LINE_START | PAGE_END | LAST_LINE_END |

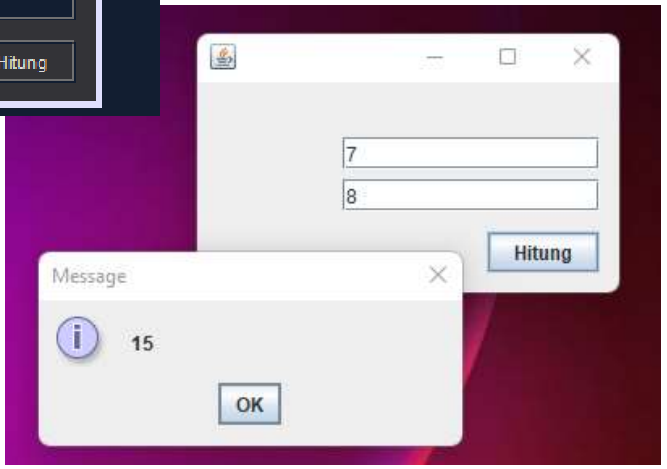
# Membuat Aplikasi GUI menggunakan Netbeans GUI Builder

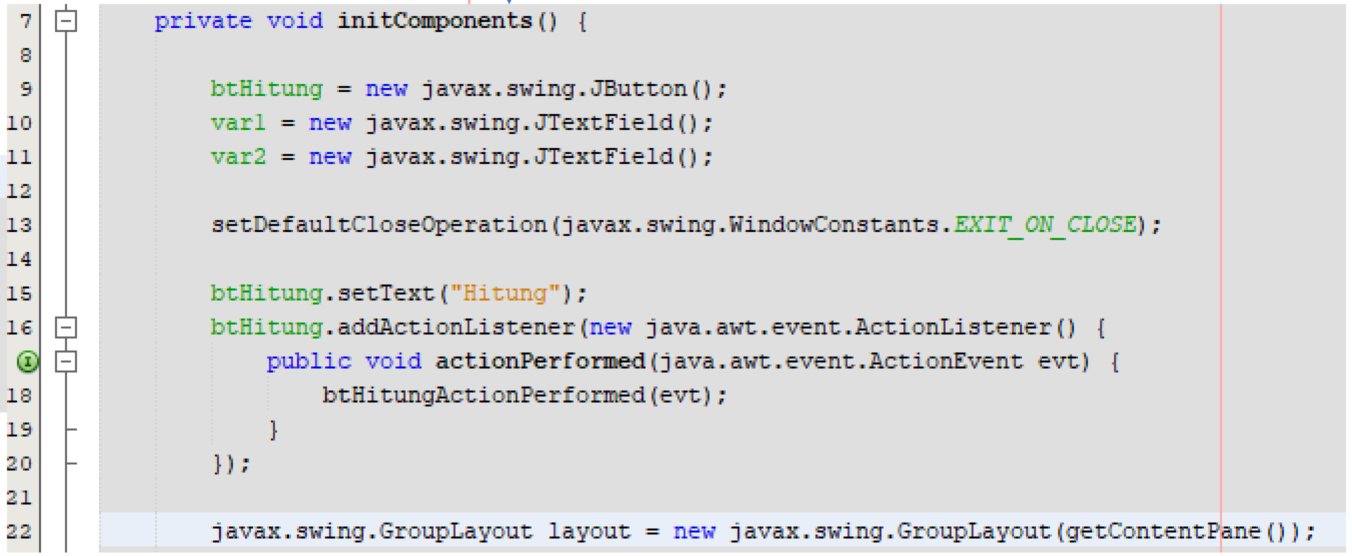


```
public class FrameNB extends javax.swing.JFrame {
    public FrameNB() {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {...41 lines }// </editor-fold>

    private void btHitungActionPerformed(java.awt.event.ActionEvent evt) {
       javax.swing.JOptionPane.showMessageDialog(
          this,Integer.parseInt(var1.getText())+Integer.parseInt(var2.getText())
       );
    }
    public static void main(String args[]) {
       java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrameNB().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton btHitung;
    private javax.swing.JTextField var1;
    private javax.swing.JTextField var2;
    // End of variables declaration
}
```



Use the context menu to access available useful actions fo





```
    private void initComponents() {

        btHitung = new javax.swing.JButton();
        var1 = new javax.swing.JTextField();
        var2 = new javax.swing.JTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        btHitung.setText("Hitung");
        btHitung.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btHitungActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
```

**Tugas Pertemuan 9**

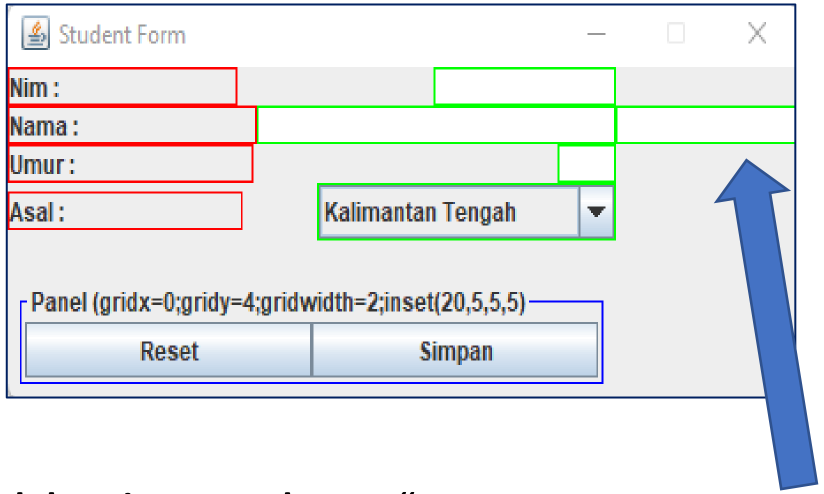
**MVC(Model View Controller), MVP (Model View Presenter) adalah beberapa Architecture Pattern yang digunakan saat membangun aplikasi berbasis GUI. Inti penggunaan pattern ini adalah memisahkan antara business logic, data, dan tampilan (UI) sehingga memudahkan membangun aplikasi saat dikerjakan secara tim, memudahkan proses testing, dan pengembangan lain jika diperlukan.**
**https://sites.google.com/site/averagelosercom/Java/java-model-view-presenter-in-swing**
**http://www.wildcrest.com/Potel/Portfolio/mvp.pdf .**
https://ssaurel.medium.com/learn-to-make-a-mvc-application-with-swing-and-java-8-3cd24cf7cb10
https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
Buatlah sebuah Aplikasi GUI dengan MVC atau MVP Pattern berupa Form mahasiswa sederhana menggunakan Layout GridBagLayout dan nested layout lainnya (GridLayout,FlowLayout, dsj.. ) dan **layout komponen Harus sama seperti contoh GUI Student Form pada gambar** ! **Isian Asal Daerah** cukup pilihan : **Jakarta ;Bogor; Depok; Tangerang; Bekasi;.** Selanjutnya Saat **tombol reset** di pilih maka semua isian akan kembali seperti semula, sedangkan pada **saat tombol simpan** di pilih akan dilakukan **proses validasi** sebelumnya dengan ketentuan :

- NIM HARUS memiliki Panjang 6 digit
- Nama : Panjang **maksimal** *nama depan + nama belakang* adalah **50 digit**
- Umur : hanya bisa diisi oleh Angka
- Semua isian wajib terisi.

Jika validasi benar maka akan ditampilkan sebuah dialog yang bertuliskan

**"Daftar NIM dan Nama Mahasiswa yang telah berhasil disimpan :**
**NIM1-Nama1**
**NIM2-Nama2**
**……….**
**"**

Tetapi jika validasi salah maka akan menampilkan dialog yang bertuliskan **"Terjadi kesalahan input pada : …. "**



Isian Nama Belakang

Terima Kasih

## Common Event-Handling Problems

**Problem:** I'm trying to handle certain events from a component, but the component isn't generating the events it should.

- make sure you registered the right kind of listener to detect the events. See whether another kind of listener might detect the kind of events you need
- Make sure you registered the listener on the right object.
- Did you implement the event handler correctly? For example, if you extended an adapter class, then make sure you used the right method signature. Make sure each event-handling method is public void, that the name spelled right and that the argument is of the right type.

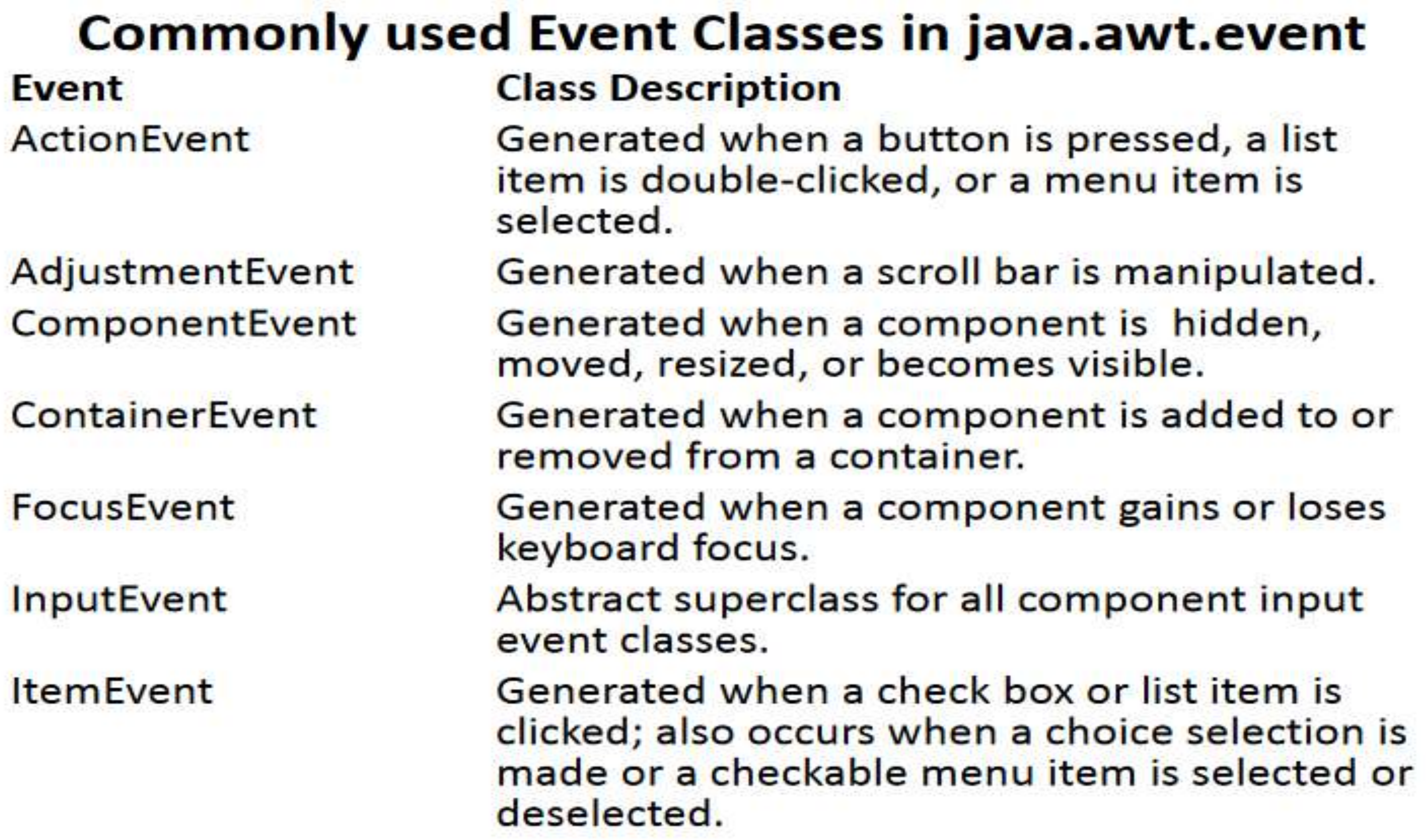
**Problem**: My combo box isn't generating low-level events such as focus events.

- Combo boxes are compound components — components implemented using multiple components. For this reason, combo boxes don't fire the low-level events that simple components fire. For more information, see Handling Events on a Combo Box.

**Problem**: The document for an editor pane (or text pane) isn't firing document events.

- The document instance for an editor pane or text pane might change when loading text from a URL. Thus, your listeners might be listening for events on an unused document. For example, if you load an editor pane or text pane with HTML that was previously loaded with plain text, the document will change to an HTMLDocument instance. If your program dynamically loads text into an editor pane or text pane, make sure the code adjusts for possible changes to the document (re-register document listeners on the new document, and so on).

# Event Classes



## Commonly used Event Classes in java.awt.event

| Event | Class Description |
|---|---|
| ActionEvent | Generated when a button is pressed, a list item is double-clicked, or a menu item is selected. |
| AdjustmentEvent | Generated when a scroll bar is manipulated. |
| ComponentEvent | Generated when a component is  hidden, moved, resized, or becomes visible. |
| ContainerEvent | Generated when a component is added to or removed from a container. |
| FocusEvent | Generated when a component gains or loses keyboard focus. |
| InputEvent | Abstract superclass for all component input event classes. |
| ItemEvent | Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected. |

# Event Classes

For Better Official Statistics

**Continued..**

| | |
|---|---|
| KeyEvent | Generated when input is received from the keyboard. |
| MouseEvent | Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. |
| MouseWheelEvent | Generated when the mouse wheel is moved. |
| TextEvent | Generated when the value of a text area or text field is changed. |
| WindowEvent | Generated when a window is activated, closed, deactivated, deiconified, |

***Concurrency*** ialah kemampuan suatu program untuk menangani *multiple order atau request*. Dimana setiap order atau request yang masuk bisa ditumpuk / dibebani oleh satu proses. Sebagai contoh Aplikasi word yang selalu siap menerima respon terhadap event dari keyboard atau mouse, tidak peduli seberapa sibuk saat memformat sebuah text atau mengupdate tampilan. Software yang bisa melakukan hal seperti itu dikenal dengan concurrent software. Java mendukung *concurrency.*

Threads exist within a process — every process has at least one

https://medium.com/@peterlee2068/concurrency-and-parallelism-in-java-f625bc9b0ca4

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html