BAB 8
GUI Event Handling

8.1 Tujuan

Pada modul ini, Anda akan belajar bagaimana mengendalikan *events triggered* ketika user berinteraksi dengan aplikasi GUI Anda. Setelah menyelesaikan modul ini, Anda akan dapat mengembangkan aplikasi GUI yang dapat merespon interaksi user.

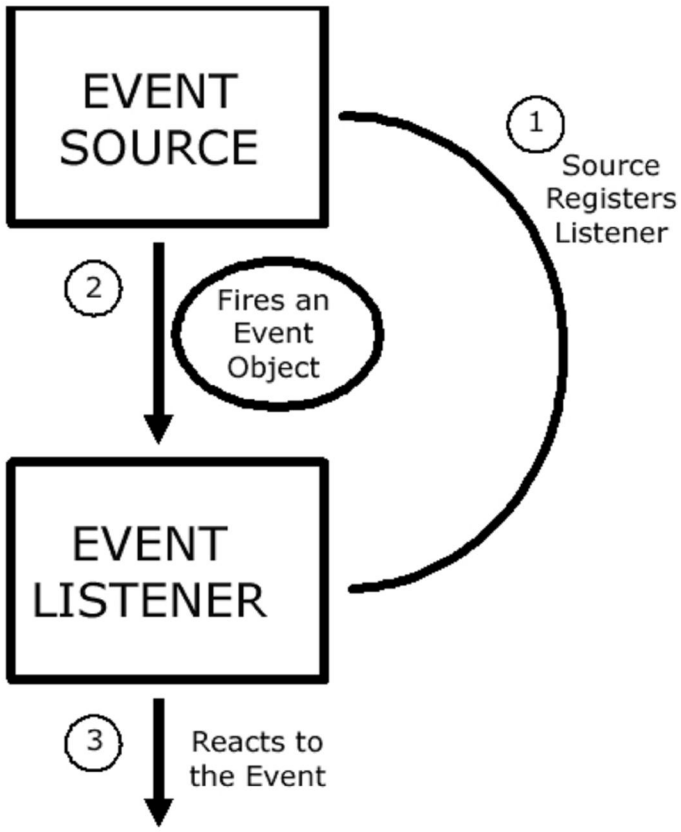
Pada akhir pembahasan, diharapkan pembaca dapat :
1. Menerangkan komponen-komponen delegation event model
2. Mengerti bagaimana delegation event model bekerja
3. Menciptakan aplikasi GUI yang berinteraksi dengan user
4. Mendiskusikan manfaat dari class-class adapter
5. Mendiskusikan keuntungan-keuntungan dari menggunakan *inner* dan *anonymous class*

8.2 Delegation Event Model

Delegasi event model menguraikan bagaimana program Anda dapat merespon interaksi dari user. Untuk memahami model, pertama-tama mari kita pelajari melalui tiga komponen utamanya.

1. Event Source
   Event source mengacu pada komponen GUI yang meng-generate event. Sebagai contoh, jika user menekan tombol, event source dalam hal ini adalah tombol.
2. Event Listener/Handler
   Event listener menerima berita dari event-event dan proses-proses interaksi user. Ketika tombol ditekan, listener akan mengendalikan dengan menampilkan sebuah informasi yang berguna untuk user.
3. Event Object
   Ketika sebuah event terjadi (misal, ketika user berinteraksi dengan komponen GUI), sebuah object event diciptakan. Object berisi semua informasi yang perlu tentang event yang telah terjadi. Informasi meliputi tipe dari event yang telah terjadi, seperti ketika mouse telah di-klik. Ada beberapa class event untuk kategori yang berbeda dari *user action*. Sebuah *event object* mempunyai tipe data mengenai salah satu dari class ini.

Di bawah ini adalah *delegation event model*.



Gambar 8.1: Delegation Event Model

Pada awalnya, sebuah listener seharusnya diregistrasikan dengan sebuah source sehingga dapat menerima informasi tentang event-event yang terjadi pada source tersebut. Hanya listener yang sudah teregistrasi yang dapat menerima pemberitahuan event-event. Ketika telah teregistrasi, sebuah listener hanya tinggal menunggu sampai event terjadi.

Ketika sesuatu terjadi dengan event source, sebuah event object akan menguraikan event yang diciptakan. Event kemudian ditembak oleh source pada listener yang teregistrasi.

Saat listener menerima sebuah event object (pemberitahuan) dari source, dia akan bekerja. Menerjemahkan pemberitahuan dan memproses event yang terjadi.

8.2.1 Registrasi Listeners

Event source mendaftarkan sebuah listener melalui method *add<Type>Listener*.

```
void add<Type>Listener(<Type>Listener listenerObj)
```

<Type> tergantung pada tipe dari event source. Dapat berupa *Key*, *Mouse*, *Focus*, *Component*, *Action* dan lainnya.
Beberapa listeners dapat diregistrasi dengan satu event source untuk menerima pemberitahuan event.

Listener yang telah teregistrasi dapat juga tidak diregistrasikan lagi menggunakan method *remove<Type>Listener*.

```
void remove<Type>Listener(<Type>Listener listenerObj)
```

8.3 Class-Class Event

Sebuah event object mempunyai sebuah class event sebagai tipe data acuannya. Akar dari hirarki class event adalah class *EventObject*, yang dapat ditemukan pada paket *java.util*. Immediate subclass dari class *EventObject* adalah class *AWTEvent*. Class *AWTEvent* didefinisikan pada paket java.awt. Itu merupakan akar dari semua AWT-based events. Berikut ini beberapa dari class-class AWT event.

Class Event	***Deskripsi***
ComponentEvent	Extends *AWTEvent*. Dijalankan ketika sebuah komponen dipindahkan, di-*resize*, dibuat *visible* atau *hidden*.
InputEvent	Extends *ComponentEvent*. Abstrak root class event untuk semua komponen-level input class-class event.
ActionEvent	Extends *AWTEvent*. Dijalankan ketika sebuah tombol ditekan, melakukan double-klik daftar item, atau memilih sebuah menu.
ItemEvent	Extends *AWTEvent*. Dijalankan ketika sebuah item dipilih atau di-*deselect* oleh user, seperti sebuah list atau checkbox.
KeyEvent	Extends *InputEvent*. Dijalankan ketika sebuah *key* ditekan, dilepas atau diketikkan.
MouseEvent	Extends *InputEvent*. Dijalankan ketika sebuah tombol mouse ditekan, dilepas, atau di-klik (tekan dan lepas), atau ketika sebuah kursor mouse masuk atau keluar dari bagian visible dari komponen.
TextEvent	Extends *AWTEvent*. Dijalankan ketika nilai dari text field atau text area dirubah.
WindowEvent	Extends *ComponentEvent*. Dijalankan sebuah object *Window* dibuka, ditutup, diaktifkan, nonaktifkan, *iconified*, *deiconified*, atau ketika *focus* ditransfer kedalam atau keluar window.

Tabel 1.2: Class-Class Event

Catatan, bahwa semua subclass-subclass *AWTEvent* mengikuti konvensi nama berikut ini:

```
<Type>Event
```

8.4 Event Listeners

Event listeners adalah class yang mengimplementasikan interfaces <*Type*>*Listener*. Tabel di bawah menunjukkan beberapa listener interfaces yang biasanya digunakan.

Event Listeners	***Deskripsi***
ActionListener	Bereaksi atas perubahan mouse atau atau keyboard.
MouseListener	Bereaksi atas pergerakan mouse.
MouseMotionListener	Interface MouseMotionListener mendukung MouseListener. Menyediakan method-method yang akan memantau pergerakan mouse,seperti drag dan pemindahan mouse.
WindowListener	Bereaksi atas perubahan window.

Tabel 1.3: Event Listeners

8.4.1 Method ActionListener

Interface ActionListener hanya terdiri dari satu method.

Method ActionListener
`public void actionPerformed(ActionEvent e)`
Mengendalikan *ActionEvent e* yang terjadi.

Tabel 1.3.1: Method ActionListener

8.4.2 Method MouseListener

Di bawah ini adalah method-method *MouseListener* yang seharusnya digunakan dalam penerapan class.

Method-method MouseListener
`public void mouseClicked(MouseEvent e)`
Dipanggil pada saat tombol mouse di click (seperti tekan dan lepas).
`public void mouseEntered(MouseEvent e)`
Dipanggil pada saat kursor mouse memasuki area komponen.
`public void mouseExited(MouseEvent e)`
Dipanggil pada saat kursor mouse meninggalkan area komponen.
`public void mousePressed(MouseEvent e)`
Dipanggil pada saat tombol mouse ditekan di atas komponen
`public void mouseReleased(MouseEvent e)`
Dipanggil pada saat tombol mouse dilepas di atas komponen

Tabel 1.3.2: Method-Method MouseListener

8.4.3 Method-Method MouseMotionListener

MouseMotionListener mempunyai dua method untuk diimplementasikan.

Method-method MouseListener
`public void mouseDragged(MouseEvent e)`
Digunakan untuk memantau pergerakan mouse yang melintasi object pada saat tombol mouse ditekan. Tindakan ini persis sama dengan tindakan pada saat memindahkan sebuah window.
`public void mouseMoved(MouseEvent e)`
Digunakan untuk memantau pergerakan mouse pada saat mouse melintasi area suatu object. Pada saat ini tidak ada mouse yang ditekan, hanya memindahkan pointer mouse melalui object.

Tabel 1.3.3: The MouseMotionListener methods

8.4.4 Method-Method WindowListener

Di bawah ini method-method dari interface *WindowListener*.

Method-method WindowListener
`public void windowOpened(WindowEvent e)`
Dipanggil pada saat object window dibuka (pertama kali window dibuat tampil).
`public void windowClosing(WindowEvent e)`
Dipanggil pada saat user mencoba untuk menutup object *Window* dari menu sistem object.
`public void windowClosed(WindowEvent e)`
Dipanggil pada saat object *Window* ditutup setelah memanggil penempatan (misal, release dari resource-resource yang digunakan oleh source) pada object.
`public void windowActivated(WindowEvent e)`
Dilibatkan ketika object *Window* adalah window yang aktif (window masih dipakai).
`public void windowDeactivated(WindowEvent e)`
Dilibatkan ketika object *Window* tidak lagi merupakan window yang aktif.
`public void windowIconified(WindowEvent e)`
Dipanggil ketika object *Window* di-minimize.
`public void windowDeiconified(WindowEvent e)`
Dipanggil ketika object *Window* kembali setelah di-minimize ke keadaan normal.

Tabel 1.3.4: Method-Method WindowListener

8.4.5 Petunjuk untuk Menciptakan Aplikasi Handling GUI Events

Berikut ini langkah-langkah yang Anda butuhkan untuk mengingat ketika ingin membuat aplikasi GUI dengan event handling.

1. Buatlah sebuah class yang menguraikan dan membuat suatu tampilan dari aplikasi GUI Anda.
2. Buatlah sebuah class yang menerapkan interface listener yang sesuai. Class ini boleh mengacu pada class yang sama seperti pada langkah awal.
3. Dalam menerapkan class, gunakan semua method-method dengan interface listener yang sesuai. Uraikan masing-masing method bagaimana Anda ingin mengendalikan event-event. Anda dapat memberikan implementasi kosong untuk method yang tidak ingin Anda gunakan.
4. Daftarkan object listener, instansiatiate dari class listener pada langkah 2, dengan *source component* menggunakan method *add<Type>Listener*.

8.4.6 Contoh Mouse Events

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

public class MouseEventsDemo extends Frame implements
                        MouseListener, MouseMotionListener {
   TextField tf;
   public MouseEventsDemo(String title){
      super(title);
      tf = new TextField(60);
      addMouseListener(this);
   }
   public void launchFrame() {
      /* Menambah komponen pada frame */
      add(tf, BorderLayout.SOUTH);
      setSize(300,300);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent me) {
      String msg = "Mouse clicked.";
      tf.setText(msg);
   }
   public void mouseEntered(MouseEvent me) {
      String msg = "Mouse entered component.";
      tf.setText(msg);
   }
   public void mouseExited(MouseEvent me) {
      String msg = "Mouse exited component.";
      tf.setText(msg);
   }
   public void mousePressed(MouseEvent me) {
      String msg = "Mouse pressed.";
      tf.setText(msg);
   }
   public void mouseReleased(MouseEvent me) {
      String msg = "Mouse released.";
      tf.setText(msg);
```

```
    }
    public void mouseDragged(MouseEvent me) {
       String msg = "Mouse dragged at " + me.getX() + "," +
                                                  me.getY();
       tf.setText(msg);
    }
    public void mouseMoved(MouseEvent me) {
       String msg = "Mouse moved at " + me.getX() + "," +
                                                  me.getY();
       tf.setText(msg);
    }
    public static void main(String args[]) {
       MouseEventsDemo med = new MouseEventsDemo("Mouse Events
                                                    Demo");
       med.launchFrame();
     }
}
```

8.4.7 Contoh Menutup Window

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


class CloseFrame extends Frame implements WindowListener {
   Label label;

   CloseFrame(String title) {
      super(title);
      label = new Label("Close the frame.");
      this.addWindowListener(this);
   }

   void launchFrame() {
      setSize(300,300);
      setVisible(true);
   }

   public void windowActivated(WindowEvent e) {
   }
   public void windowClosed(WindowEvent e) {
   }
   public void windowClosing(WindowEvent e) {
      setVisible(false);
      System.exit(0);
   }
   public void windowDeactivated(WindowEvent e) {
   }
   public void windowDeiconified(WindowEvent e) {
   }
   public void windowIconified(WindowEvent e) {
   }
   public void windowOpened(WindowEvent e) {
   }
```

```
   public static void main(String args[]) {
      CloseFrame cf = new CloseFrame("Close Window Example");
      cf.launchFrame();
    }
}
```

8.5 Class-class Adapter Class

Menerapkan semua method dari interface yang semuanya akan membutuhkan banyak pekerjaan. Di satu sisi, Anda terkadang lebih sering tertarik menerapkan hanya beberapa method dari interface saja. Untungnya, Java menyediakan untuk kita class-class adapter yang menerapkan semua method dari masing-masing listener interface dengan lebih dari satu method. Implementasi dari method-method semuanya adalah kosong.

8.5.1 Close Window Example

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

class CloseFrame extends Frame{
   Label label;
   CFListener w = new CFListener(this);

   CloseFrame(String title) {
      super(title);
      label = new Label("Close the frame.");
      this.addWindowListener(w);
   }

   void launchFrame() {
      setSize(300,300);
      setVisible(true);
   }

   public static void main(String args[]) {
      CloseFrame cf = new CloseFrame("Close Window Example");
      cf.launchFrame();
    }
}


class CFListener extends WindowAdapter{
    CloseFrame ref;
    CFListener( CloseFrame ref ){
      this.ref = ref;
      }

    public void windowClosing(WindowEvent e) {
         ref.dispose();
         System.exit(1);
   }
}
```

8.6 Inner Class dan Anonymous Inner Class

Bagian ini memberi Anda tinjauan ulang atas konsep yang sudah Anda pelajari di pelajaran pemrograman pertama. *Inner class* dan *anonymous inner class* sangatlah bermanfaat untuk GUI event handling.

8.6.1 Inner Class

Inner class, seperti namanya, adalah sebuah class yang dideklarasikan di dalam class lain. Kegunaan inner classes akan dapat membantu Anda menyederhanakan program, terutama dalam event handling seperti yang ditunjukkan pada contoh.

8.6.2 Contoh Menutup Window

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

class CloseFrame extends Frame{
      Label label;

      CloseFrame(String title) {
            super(title);
            label = new Label("Close the frame.");
            this.addWindowListener(new CFListener());
      }

      void launchFrame() {
            setSize(300,300);
            setVisible(true);
      }

      class CFListener extends WindowAdapter {
            public void windowClosing(WindowEvent e) {
                  dispose();
                  System.exit(1);
            }
      }

      public static void main(String args[]) {
            CloseFrame cf = new CloseFrame("Close Window
                                                Example");
            cf.launchFrame();
      }
}
```

8.6.3 Anonymous Inner Class

Anonymous inner class adalah *inner class* tanpa nama. Kegunaan dari *anonymous inner class* akan menyederhanakan kode-kode Anda lebih lanjut. Di bawah ini merupakan modifikasi dari contoh bagian sebelumnya.

8.6.4 Contoh Menutup Window

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

class CloseFrame extends Frame{
   Label label;

   CloseFrame(String title) {
      super(title);
      label = new Label("Close the frame.");
      this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e){
                  dispose();
                  System.exit(1);
            }
        });
   }

   void launchFrame() {
      setSize(300,300);
      setVisible(true);
   }

   public static void main(String args[]) {
      CloseFrame cf = new CloseFrame("Close Window Example");
      cf.launchFrame();
    }
}
```

8.7 Latihan

8.7.1 Tic-Tac-Toe

Extend program papan Tic-Tac-Toe yang telah Anda kembangkan sebelumnya dan tambahkan event handlers ke kode tersebut untuk membuat program berfungsi penuh. Permainan Tic-Tac-Toe dimainkan dengan dua pemain. Pemain mengambil giliran mengubah. Setiap giliran, pemain dapat memilih kotak pada papan. Ketika kotak dipilih, kotak ditandai oleh simbol pemain (O dan X biasanya digunakan sebagai simbol). Pemain yang sukses menaklukkan 3 kotak membentuk garis horisontal, vertikal, atau diagonal, memenangkan permainan. Permainan akan berakhir ketika pemain menang atau ketika semua kotak telah terisi.

Gambar 8.2 : Program Tic-Tac-Toe