8 BAB VIII LISTENER/EVENT HANDLER

8.1 IDENTITAS

Kajian

Komponen Swing Java non-Visual Editor; Database & Swing

Topik

1. Event Listener: ActionListener, MouseListener
2. Inner Class

Referensi

1. http://docs.oracle.com/javase/tutorial/uiswing/index.html

Kompetensi Utama

1. Mahasiswa memahami penggunaan komponen swing untuk membuat aplikasi desktop.
2. Mahasiswa mampu membuat aplikasi desktop menggunakan komponen swing—sesuai yang dituliskan di topik—tanpa bantuan visual editor.
3. Mahasiswa mampu menggunakan interface listener (ActionListener dan MouseListener) untuk menambahkan event handler pada komponen swing yang telah dibuat.

Lama Kegiatan Praktikum

1. Kegiatan Mandiri : 1 x 100 menit

Parameter Penilaian

1. Tugas Pendahuluan
2. Jurnal Mandiri

8.2 PENGENALAN EVENT HANDLER

Aplikasi GUI merupakan aplikasi berbasis "event". Komponen seperti tombol dan text field akan menunggu aksi dari keyboard ataupun mouse. Aksi tersebut merupakan event terhadap komponen GUI. Langkah pertama yang dilakukan untuk menambahkan event tersebut adalah dengan meng-import paket "awt.event.*" Walaupun komponen yang digunakan merupakan komponen swing, untuk event, paket awt.event tetap yang harus di-import.

Langkah berikutnya adalah meng-implements "xxxListener". Daftarkan event listener untuk widget/komponen yang dikenai event dengan menggunakan method "addxxxListener(this)". Keyword this merupakan representasi objek dari class yang memiliki handler method seperti "xxxPerformed()".

Yang perlu diperhatikan, interface listener harus di-implements oleh kelas atau innerclass yang digunakan untuk menuliskan event dari komponen. Dan penulisannya biasanya bersifat private untuk secure implementation. Daftar dari listener (yang umum digunakan) adalah sebagai berikut:

Interface	Method	Penggunaan
ActionListener	actionPerformed()	Klik Tombol dan lain-lain
AdjustmentListener	adjustmentValueChanged()	Posisi pergerakan scroll bar
ChangeListener	stateChanged()	Pergerakan slider
ComponentListener	componentResized()	Perubahan ukuran komponen
ContainerListener	componentAdded() componentRemoved()	Penambahan dan pengurangan komponen
FocusListener	focusGained() focusLost()	Focus pada text field
ItemListener	itemStateChanged()	Perubahan state pada checkbox
KeyListener	keyPressed() keyReleased() keyTyped()	Penggunaan keyboard
MouseListener	mouseClicked()	Klik mouse
MouseMotionListener	mouseDragged() mouseMoved()	Pergerakan mouse
MouseWheelListener	mouseWheelMoved()	Pergerakan mouse wheel
TextListener	textValueChanged()	Perubahan konten teks
WindowListener	windowActivated() windowClosed() windowClosing() windowDeactivated()	Perubahan state window

Setiap listener merupakan interface yang harus dituliskan kembali setiap methodnya walaupun method tersebut tidak akan digunakan. Untuk menghindari hal ini, terdapat suatu adapter class untuk setiap listener. Adapter class akan meng-implements interface dan menuliskan kembali method pada interface tanpa body methodnya. Class bentukan hanya perlu meng-extends kelas adapter dan meng-override method yang diperlukan.

Contoh perbandingan penggunaan (atas—listener; bawah—adapter):

```
//An example that implements a listener interface directly.
public class MyClass implements MouseListener {
    ...
        someObject.addMouseListener(this);
    ...
    /* Empty method definition. */
    public void mousePressed(MouseEvent e) {
    }

    /* Empty method definition. */
    public void mouseReleased(MouseEvent e) {
    }

    /* Empty method definition. */
    public void mouseEntered(MouseEvent e) {
    }

    /* Empty method definition. */
    public void mouseExited(MouseEvent e) {
    }

    public void mouseClicked(MouseEvent e) {
        ...//Event listener implementation goes here...
    }
}
```

```
/*
 * An example of extending an adapter class instead of
 * directly implementing a listener interface.
 */
public class MyClass extends MouseAdapter {
    ...
        someObject.addMouseListener(this);
    ...
    public void mouseClicked(MouseEvent e) {
        ...//Event listener implementation goes here...
    }
}
```

Jika ingin menggunakan class adapter tapi class tersebut tidak ingin meng-extends class adapter secara langsung, konsep innerclass bisa digunakan. Berikut contohnya:

```
//An example of using an inner class.
public class MyClass extends Applet {
    ...
        someObject.addMouseListener(new MyAdapter());
    ...
    class MyAdapter extends MouseAdapter {
        public void mouseClicked(MouseEvent e) {
            ...//Event listener implementation goes here...
        }
    }
}
```

Contoh di atas, terdapat sebuah class yang harus meng-extends Applet sekaligus MouseAdapter. Hal ini pastinya tidak bisa dilakukan. Dikarenakan MouseAdapter hanya bertindak sebagai event dan tidak memiliki peranan lain pada kelas utama, event tersebut dapat dipindahkan kepada satu class lain yang terdapat pada class utama. Dengan kondisi class lain tersebut lah yang meng-extends MouseAdapter.

Variasi dari inner class adalah menggunakan anonymous inner class. Hal ini seolah2 menjadikan suatu kelas yang dibentuk objeknya menjadi parameter masukan untuk suatu method. Sama dengan contoh di atas, hanya saja contoh di atas, class memiliki nama “MyAdapter” dan terdapat pada blok program yang berbeda. Sedangkan anonymous inner class sebaliknya. Contoh:

```
//An example of using an anonymous inner class.
public class MyClass extends Applet {
    ...
        someObject.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                ...//Event listener implementation goes here...
            }
        });
    ...
    }
}
```

Tidak semua Listener memiliki Adapter. Daftarnya yang paling sering digunakan adalah sebagai berikut:

Listener Interface	Adapter Class	Listener Methods
`ActionListener`	-	`actionPerformed(ActionEvent)`
`ChangeListener`	-	`stateChanged(ChangeEvent)`
`FocusListener`	`FocusAdapter`	`focusGained(FocusEvent)` `focusLost(FocusEvent)`
`ItemListener`	-	`itemStateChanged(ItemEvent)`

`KeyListener`	`KeyAdapter`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

`MenuListener`	-	`menuCanceled(MenuEvent)`
		`menuDeselected(MenuEvent)`
		`menuSelected(MenuEvent)`

`MouseListener`	`MouseAdapter`,
	`MouseInputAdapter`
		`mouseClicked(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`

`TableModelListener`	-	`tableChanged(TableModelEvent)`

Setiap komponen swing memerlukan interface listener yang berbeda-beda. Untuk ComponentListener, FocusListener, KeyListener, MouseListener, MouseMotionListener, MouseWheelListener, HierarchyBoundsListener dan HierarchyListener dapat diaplikasikan ke semua komponen swing

Tapi ada beberapa listener yang hanya support beberapa komponen saja karena event tersebut tidak berasal dari awt container. Berikut daftar komponen umum yang sering digunakan dengan interface yang dapat diaplikasikan pada komponen tersebut:

Komponen	Listener
Button	ActionListener, ChangeListener, ItemListener
Check Box	ActionListener, ChangeListener, ItemListener
Combo Box	ActionListener, ItemListener
Menu Item	ActionListener, ChangeListener, ItemListener
Radio Button	ActionListener, ChangeListener, ItemListener
Table	ListSelectionListener
Text field	ActionListener, CaretListener, DocumentListener
Text area	CaretListener, DocumentListener
Tabbed Pane	ChangeListener

Umumnya, untuk button dan turunannya, digunakan interface ActionListener dengan method "actionPerformed(ActionEvent e)". Bahkan untuk menu, ActionListener juga umum digunakan.

Terdapat 2 cara umum yang digunakan untuk menambahkan listener pada komponen. Cara pertama seperti yang digunakan oleh netbean, yaitu memberikan listener dengan mendeklarasikan method performed yang akan digunakan:

```
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });
```

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

Dengan menggunakan cara di atas, class yang digunakan tidak perlu mengimplements ActionListener. Dalam kasus Netbeans gui builder terlihat bahwa kelas yang digunakan untuk merancang dan menampilkan gui hanya meng-extends JFrame tanpa meng-implements listener apapun.

Cara kedua yang umum digunakan dengan meng-implements listener terkait (tergantung komponen yang akan diberikan event handler) selain meng-extends class yang digunakan sebagai container (bisa secondary ataupun primary container). Penentuan komponen yang dikenai event handler tergantung dengan command yang diberikan. Contoh:

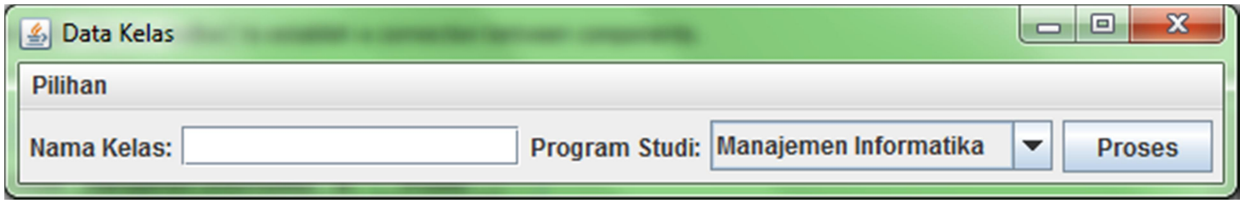
```
public class SimpleButtonDemo extends JPanel implements ActionListener {
    protected JButton b1;

    public SimpleButtonDemo() {

        b1 = new JButton("Tombol 1");
        b1.setActionCommand("lakukan");

        //Tambahkan action listener untuk button
        b1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if ("lakukan".equals(e.getActionCommand())) {
```

```
            System.out.println("Tombol Ditekan");
        }
    }
}
```

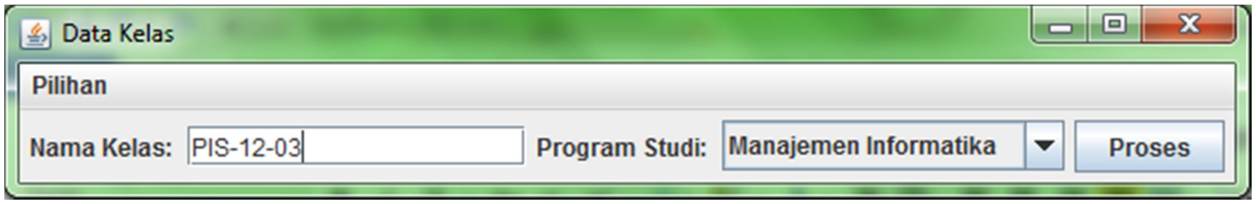
8.3 PRAKTIK

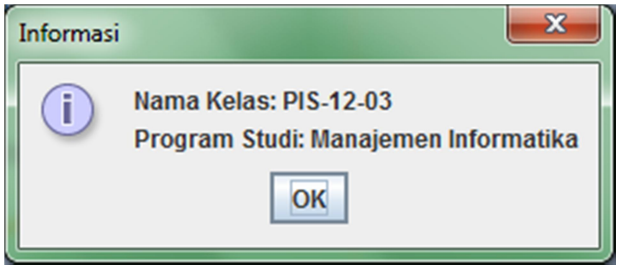
8.3.1 Soal

Buatlah GUI sebagai berikut tanpa menggunakan gui builder:



Pada menu "Pilihan" terdapat menu item "Kosongkan". Gunakan layout bawaan masing-masing container. Tambahkan listener, sehingga saat menekan menu "Kosongkan", semua field menjadi kosong dan saat menekan tombol "Proses", keluar option pane yang menampilkan teks yang dimasukkan ke text field serta informasi prodi yang dipilih. Contoh:





8.3.2 Solusi

Terdapat 2 cara untuk menambahkan event listener ke sebuah kelas GUI, dengan menggabungkan di kelas yang sama dengan kelas GUI, atau membuat kelas tersendiri khusus untuk melakukan handler.

Cara termudah adalah dengan menggabungkan antara GUI dengan action-nya.

Untuk menjalankan solusi ini, pada kelas GUI yang telah dibuat sebelumnya di modul 7, tambahkan interface listener yang diinginkan di kelas tersebut. Dalam hal ini, kita tambahkan listener ActionListener. Setelah menambahkan interface ini, maka method actionPerformed harus di-override, sehingga perubahan dari kelas sebelumnya terlihat sebagai berikut:

```
public class DataKelasGUIHandler extends JFrame implements ActionListener{
…
   public void actionPerformed(ActionEvent e) {

   }
}
```

Tuliskan logika pemrograman di dalam method actionPerformed tersebut. Method ini dijalankan ketika ada suatu aksi di aplikasi GUI, dengan catatan komponen yang sudah dikenai aksi telah dikenali oleh listener (dilakukan dengan memanggil method “addActionListener()” pada objek komponen terkait). Hal ini akan menyulitkan jika ada beberapa tombol yang harus ditekan, beberapa menu item yang bisa di-klik, beberapa radio button yang dapat dipilih dan semua aksi yang dilakukan.

Untuk membedakan setiap aksi yang terjadi, pada komponen terkait, bisa ditambahkan “setActionCommand()” yang nantinya di actionPerformed diambil dengan method “getActionCommand()”.

Untuk membuktikan bahwa actionPerformed bersifat universal, tambahkan listener untuk tombol dan menu item:

```
  public JMenuBar createMenuBar() {
     menuBar = new JMenuBar();
     menu = new JMenu("Pilihan");
     menuBar.add(menu);

     menuItem = new JMenuItem("Kosongkan");
     menu.add(menuItem);
     menuItem.addActionListener(this); //perhatikan baris ini
     return menuBar;
  }

  public void addComponentsToPane(final Container pane) {
     FlowLayout tataLetak = new FlowLayout();
     final JPanel panelKomponen = new JPanel();
     panelKomponen.setLayout(tataLetak);
     tataLetak.setAlignment(FlowLayout.CENTER);

     labelKelas = new JLabel("Nama Kelas: ");
     panelKomponen.add(labelKelas);

     teksKelas = new JTextField(15);
     panelKomponen.add(teksKelas);

     labelProdi = new JLabel("Program Studi: ");
     panelKomponen.add(labelProdi);

     String daftarPilihan[] = {"Manajemen Informatika",
                       "Komputerisasi Akuntansi",
                       "Teknik Komputer"};
     comboProdi = new JComboBox(daftarPilihan);
     panelKomponen.add(comboProdi);

     tombolProses = new JButton("Proses");
     panelKomponen.add(tombolProses);
     tombolProses.addActionListener(this); //perhatikan baris ini

     pane.add(panelKomponen);
  }
```

Lalu ubah actionPerformed menjadi sebagai berikut:

```
public void actionPerformed(ActionEvent e) {
   System.out.println("Hello World");
}
```

Perhatikan ketika dijalankan, saat tombol ditekan dan menu item dipilih, aksi yang dilakukan sama. Untuk membedakan, tambahkan method “setActionCommand()” untuk setiap komponen terkait.

```
public JMenuBar createMenuBar() {
   menuBar = new JMenuBar();
   menu = new JMenu("Pilihan");
   menuBar.add(menu);

   menuItem = new JMenuItem("Kosongkan");
   menu.add(menuItem);
   menuItem.setActionCommand("kosong"); //perhatikan baris ini
   menuItem.addActionListener(this); //perhatikan baris ini
   return menuBar;
}
```

```
public void addComponentsToPane(final Container pane) {
   …
   tombolProses = new JButton("Proses");
   panelKomponen.add(tombolProses);
   tombolProses.setActionCommand("tombol"); //perhatikan baris ini
   tombolProses.addActionListener(this); //perhatikan baris ini
   …
}
```

Parameter untuk method setActionCommand bersifat bebas, selama masih bertipe String. String inilah nantinya yang akan di-filter oleh action performed dengan method “getActionCommand()” sehingga dapat dibedakan aksi dari button atau aksi dari menu item.

```
public void actionPerformed(ActionEvent e) {
   if(e.getActionCommand().equalsIgnoreCase("kosong")){
      //lakukan aksi jika menu item ditekan
   }else if(e.getActionCommand().equals("tombol")){
      //lakukan aksi jika tombol ditekan
   }
}
```

Untuk mem-filternya, dapat digunakan perbandingan string, yaitu equalsIgnoreCase dan equals. Perbedaannya, untuk equalsIgnoreCase, huruf besar dan huruf kecil tidak dianggap berbeda, sedangkan equals sebaliknya. “kosong” dan “tombol” disesuaikan dengan action command yang diberikan ke masing-masing komponen.

Untuk mengeluarkan option pane saat menekan tombol, ubah method actionPerformed sebagai berikut:

```
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equalsIgnoreCase("kosong")) {

    } else if (e.getActionCommand().equals("tombol")) {
        //ambil teks nama kelas dari field teks
        String namaKelas = teksKelas.getText();
        //ambil item dari combo box
        String prodi = (String) comboProdi.getSelectedItem();
        //atur string yang akan ditampilkan
        String pesan = "Nama Kelas: " + namaKelas + "\n"
                + "Program Studi: " + prodi;
        //tampilkan dengan option pane
        JOptionPane.showMessageDialog(null, pesan,
                "Informasi", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Untuk mengosongkan sesuai tampilan default, tambahkan satu method bernama kosong yang akan dipanggil di actionPerformed:

```
public void kosong(){
    teksKelas.setText("");
    comboProdi.setSelectedIndex(0);
}
```

Panggil method tersebut di actionPerformed:

```
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equalsIgnoreCase("kosong")) {
        kosong();
    } else if (e.getActionCommand().equals("tombol")) {
        ……
    }
}
```

Pastikan setiap aksi berjalan semestinya.