

MATERI PELATIHAN

Java Swing

Ifnu Bima

ifnubima@gmail.com

<http://www.nagasakti.or.id/roller/ifnu>

Versi : 0.1-07.03

Daftar Isi

Java Foundation Class.....	1
Feature JFC.....	1
Swing Package.....	2
Swing HelloWorld.....	4
Install Java Development Kit.....	4
Membuat program HelloWorld	4
Melakukan kompilasi program HelloWorld.....	5
Menjalankan program HelloWorld.....	6
Membuat Swing HelloWorld dengan Netbeans 5.5.....	6
Komponen Swing	9
Struktur Komponen Swing.....	10
Bekerja dengan JLabel, JTextField dan JButton.....	10
Bekerja dengan JCheckBox dan JRadioButton.....	15
Bekerja dengan JList dan JComboBox	19
Bekerja dengan Menu, Popup Menu dan Toolbar.....	23
Membuat Menu.....	25
Membuat Popup Menu	30
Membuat Toolbar	33
Membuat Dialog dan JFileChooser.....	37
Membuat pre-defined dialog dengan JOptionPane.....	38
Membuat JFileChooser.....	44
Konsep MVC.....	48
Model dalam Komponen Swing	50
TableModel.....	52
ListModel dan ListSelectionModel.....	55
Menangani Event.....	57
Event Listener dalam Swing	58
ActionListener.....	64
KeyListener.....	67

MouseListener dan MouseMotionListener.....	70
Koneksi Database Dengan JDBC.....	74
Mengetahui JDBC.....	74
Database Driver.....	75
Membuat Koneksi.....	76
Memodifikasi Data dari Database.....	78
Menggunakan PreparedStatement.....	83
DataBinding Menggunakan GlazedLists.....	87
EventList.....	88
EventList dari Record-Record Database.....	89
EventList dalam Model.....	92
EventListModel.....	92
EventComboBoxModel.....	94
EventTableModel.....	95
Satu EventList untuk Semua Model.....	98
Pengurutan Tabel	99
Manual Sorting.....	100
Automatic Sorting.....	102
Menyaring Text dalam Tabel.....	104
TextFilterator.....	105
Mem-filter Data Customer	105
TransformedList dan UniqueList.....	108
Model Master-Detail.....	111
FilterList, Matcher dan MatcherEditor.....	112
Membuat Model Master-Detail.....	114
Penutup.....	117
Referensi dan Bacaan Lebih Lanjut.....	118

Java Foundation Class

Java Foundation Class (JFC) merupakan sekumpulan class-class Java yang digunakan untuk mengembangkan perangkat lunak berbasis GUI (Graphical User Interface). Selain itu, JFC juga mempunyai class-class yang digunakan untuk menambahkan fungsi dan kemampuan interaksi yang variatif dari pemrograman Java. Dari definisi ini, JFC tidak hanya berisi class-class GUI saja tetapi juga class-class lain yang dapat meningkatkan kemampuan pemrograman Java baik dari segi fungsionalitasnya maupun dari segi kemampuan interaksi pemrograman Java yang sangat kaya.

Feature JFC

Fitur-fitur yang dipunyai oleh JFC	
Fitur	Deskripsi
Komponen Swing	Memuat semua class-class yang dibutuhkan untuk membuat aplikasi berbasis GUI, dari tombol, table, tab, menu, toolbar dan sebagainya
Look and Feel (LaF)	Memberikan kemampuan kepada program Java yang dikembangkan menggunakan library swing untuk memilih tema tampilan. Misalnya sebuah program yang sama dapat mempunyai tampilan windows LaF atau Java LaF, atau LaF lain yang dikembangkan oleh komunitas seperti JGoodies.
Accessibility API	Fasilitas untuk mengembangkan aplikasi bagi penyandang cacat, misalnya dukungan untuk membuat huruf braile, kemampuan mengambil input dari layar sentuh dan sebagainya.
Java 2D API	Berisi kumpulan class-class yang dapat digunakan

	untuk memanipulasi object-object 2 dimensi, seperti garis, kotak, lingkaran, kurva dan lain sebagainya. Selain itu Java 2D API juga memberikan kemampuan program yang ditulis menggunakan Java untuk mencetak output ke alat pencetak seperti printer.
Drag-and-drop	Menyediakan kemampuan drag-and-drop antara program Java dan program lain yang ditulis spesifik untuk suatu platform sistem operasi tertentu.
Internationalization (i18n)	Membantu pengembang perangkat lunak untuk membangun aplikasi yang dapat mendukung semua bahasa dan huruf yang ada di dunia.

Tabel Feature JFC

Modul ini akan berkonsentrasi untuk membahas komponen swing. Pemilihan komponen dan library swing yang tepat dapat mempengaruhi kualitas program yang kita buat secara signifikan. Hal ini dikarenakan, dalam dunia Java Standard Edition, lebih spesifik lagi aplikasi Java yang dibangun menggunakan swing, belum ada framework yang benar-benar komprehensif membimbing pengembang untuk membuat aplikasi yang berkualitas.

Pada umumnya aplikasi yang dikembangkan dengan Swing mempunyai kode yang sangat 'kotor', dimana kode yang berisi pengendalian terhadap event komponen swing bercampur aduk dengan kode yang berisi aturan bisnis dan kode yang berisi manipulasi terhadap data.

Swing Package

Swing API sangat bagus dan lengkap, Java 6.0 menyertakan setidaknya tujuh belas (17) buah package yang berisi class-class

swing yang siap digunakan.

javax.accessibility	javax.swing.plaf	javax.swing.text
javax.swing	javax.swing.plaf.basic	javax.swing.text.html
javax.swing.border	javax.swing.plaf.metal	javax.swing.text.rtf
javax.swing.colorchooser	javax.swing.plaf.multi	javax.swing.table
javax.swing.event	javax.swing.plaf.synth	javax.swing.tree
javax.swing.filechooser		javax.swing.undo

Utungnya kita tidak akan menggunakan semua class-class dalam package swing, hanya sebagian kecil saja dari class-class tersebut yang nantinya akan benar-benar kita gunakan. Sehingga kita bisa berkonsentrasi untuk memahami beberapa komponen penting saja. Dalam modul ini nanti kita hanya akan menggunakan beberapa class komponen swing yang penting saja. Beberapa kelas ini sudah cukup sebagai bahan pemembuat perangkat lunak berkualitas.

Komunitas Java juga menyediakan banyak sekali library swing, antara lain dari Swingx dan JGoodies yang mengembangkan library standard swing dengan menambahkan berbagai macam feature menarik. Sedangkan komunitas dari javadesktop.org mengembangkan banyak sekali library swing untuk keperluan khusus. Nyaris semua komponen yang kita perlukan baik komponen umum hingga komponen untuk tujuan khusus banyak tersedia di internet, kita tinggal mencari dan menggunakan.

Praktek yang baik dalam memilih komponen apa yang tepat adalah dengan mencari dahulu informasi di internet. Hal ini sangat bermanfaat untuk mengurangi waktu kita mengembangkan komponen, sehingga kita bisa lebih banyak berkonsentrasi untuk mengembangkan sisi bisnis dan usability dari software yang kita kembangkan. Sebaik apapun software yang kita buat tapi tidak memberikan nilai tambah terhadap masalah yang dihadapi adalah kesia-siaan belaka. Banyak sekali software yang dianggap gagal

memberikan nilai tambah terhadap masalah yang dihadapi hanya karena tampilan GUI-nya sangat susah dipahami dan tidak intuitif.

Swing HelloWorld

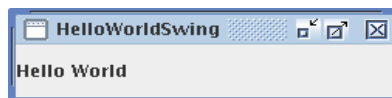
Menggunakan contoh langsung adalah cara yang tepat untuk memulai proses belajar. Cara ini memberikan gambaran kongkrit tentang subject yang akan dipelajari, sehingga proses belajar lebih cepat diserap. Untuk tujuan ini, kita akan membuat sebuah program kecil yang menampilkan kata "HelloWorld" menggunakan komponen swing. Berikut ini adalah langkah-langkah yang harus anda lakukan untuk membuat program "HelloWorld" berbasis komponen swing:

1. Install Java Development Kit (JDK)
2. Membuat program HelloWorld itu sendiri
3. Melakukan kompilasi program HelloWorld
4. Menjalankan program HelloWorld

Install Java Development Kit

Yang perlu kita lakukan dalam langkah ini hanyalah mendownload JDK dari website java.sun.com. kemudian jalankan program instalasinya dan ikuti perintah-perintah dalam langkah-langkah instalasi tersebut. Setelah proses instalasi selesai, kita siap untuk membuat program Java.

Membuat program HelloWorld



Program Java dengan tampilan seperti di atas dapat dibuat dengan dua cara. Cara yang pertama adalah dengan menggunakan text

editor dan menetik kode program. Cara yang kedua adalah dengan menggunakan Netbeans Matisse GUI Builder.

Lakukan langkah-langkah berikut ini untuk membuat program diatas menggunakan text editor:

1. Buka text editor kesayangan anda.
2. Ketikkan kode program di bawah ini dan simpan dengan nama file HelloWorld.java :

```
public class HelloWorld {
public HelloWorld(){ }
public void display(){
    JFrame.setDefaultLookAndFeelDecorated(true);
    JLabel label = new JLabel("HelloWorld");
    JFrame frame = new JFrame();
    frame.getContentPane().add(label);
    frame.setVisible(true);
    frame.pack();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] str){
    HelloWorld hello = new HelloWorld();
    SwingUtilities.invokeLater(
        new Runnable() {
            public void run(){ hello.display(); }
        });
}
}
```

Melakukan kompilasi program HelloWorld

Kompilasi program tersebut dengan cara menjalankan program javac (java compiler). Jika anda bekerja di lingkungan windows buka command prompt, kemudian ketik program berikut ini :

```
c:\latihan> javac HelloWorld.java
```

Jika anda bekerja di lingkungan GNU/Linux, buka console dan ketikkan perintah berikut ini :

```
shell$ javac HelloWorld.java
```

Menjalankan program HelloWorld

Proses kompilasi akan menghasilkan file yang berekstensi .class, file inilah yang akan kita eksekusi. Jika anda bekerja di lingkungan windows lakukan perintah berikut ini:

```
c:\latihan> java HelloWorld
```

Jika anda bekerja di lingkungan GNU/Linux jalankan perintah berikut ini:

```
shell$ java HelloWorld
```

Membuat Swing HelloWorld dengan Netbeans 5.5

Netbeans 5.5 dilengkapi dengan GUI builder yang dikenal dengan Matisse. Dalam modul ini selanjutnya, Matisse akan digunakan untuk menyebut Netbeans GUI Builder. Tools ini sangat powerful dan produktif dalam membuat komponen GUI. Langkah-langkah yang harus anda lakukan untuk membuat Swing HelloWorld dengan Matisse adalah sebagai berikut:

1. Buat project baru dalam Netbeans, caranya pilih menu :

```
File > New Project
```

2. Langkah berikutnya anda harus menentukan kategori project yang akan anda buat, caranya pilih :

```
General > Java Application
```

Anda akan dibawa ke dialog untuk menentukan nama project

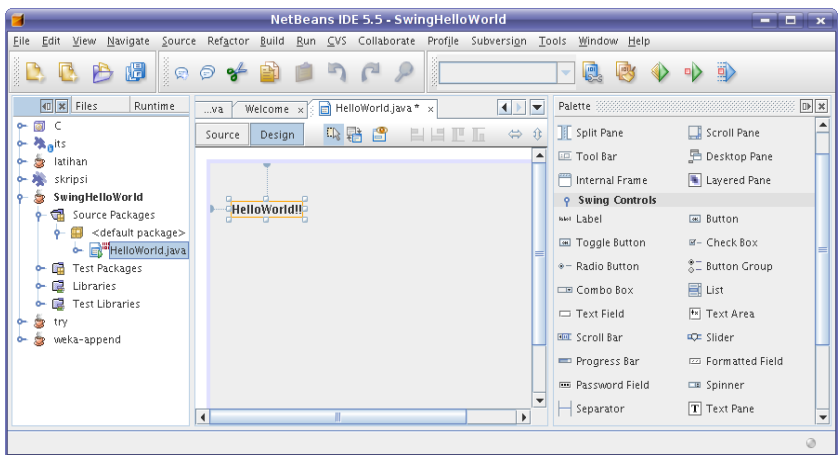
dan folder dimana anda meletakkan project tersebut, pilih folder sesuai keinginan anda.

3. Klik kanan di project yang baru anda buat, popup menu akan muncul, kemudian pilihlah menu :

```
New > JFrame Form...
```

Kemudian masukkan nama class JFrame yang akan dibuat, misalnya HelloWorld.java, klik finish.

4. Tampilan Netbeans akan berganti dengan tampilan GUI builder, dimana di sisi kanan akan terlihat Swing Pallet. Klik item Label di Swing Pallet kemudian klik di atas JFrame, sebuah JLabel akan dibuat.



Jendela Design dan Pallette Netbeans Matisse

5. Untuk memenuhi tujuan kita membuat Swing HelloWorld, kita akan memasukkan string "HelloWorld" ke dalam JLabel yang baru saja kita buat. Caranya, double klik di atas JLabel tersebut, kursor muncul bersama text field dan ketikkan "HelloWorld".
6. Klik kanan di file HelloWorld.java pada jendela explorer di sebelah kiri, pilih menu **Run File...** untuk mengcompile dan

menjalankan class HelloWorld.java atau tekan tombol SHIFT + F6.

Matisse mempunyai sistem Layouting yang sangat fleksible, sistem layout yang digunakan oleh Matisse adalah GroupLayout. Dalam chapter berikutnya kita akan belajar bagaimana menggunakan GroupLayout ini dengan benar dan memanfaatkan keunggulanya dalam menata component GUI dengan sangat rapi.

Swing helloworld ini hanya sebagian kecil saja dari pekerjaan yang harus dilakukan dalam membangun aplikasi desktop berbasis Java. Selanjutnya kita akan membahas penggunaan JLabel, JButton, JCheckBox, JTextField dan JRadioButton untuk membuat aplikasi GUI sederhana dengan menggunakan Matisse.

Komponen Swing

Swing toolkit menyediakan banyak sekali komponen untuk membangun aplikasi GUI desktop. Swing toolkit juga menyediakan class-class untuk menangani interaksi antara aplikasi dan user menggunakan standard input seperti keyboard dan mouse. Komponen-komponen yang disediakan swing mencakup semua GUI toolkit yang lazim digunakan dalam aplikasi desktop, seperti : JLabel, JList, JTree, JButton, JLabel dan masih banyak komponen-komponen lainnya yang sudah teruji dan siap pakai.

Selain komponen GUI, swing juga menyediakan fasilitas untuk proses undo, komponen untuk mengolah text, internationalization, Komponen GUI yang mendukung penyandang cacat (accessibility support) dan fasilitas drag-and-drop.

Look and Feel merupakan fasilitas yang unik dalam swing. Dengan fasilitas Look and Feel ini kita bisa dengan mudah merubah tampilan dari program kita sesuai dengan keinginan dan tujuan kita. Misalnya, agar program terlihat fancy atau agar program terlihat konsisten dalam segala keadaan.

Swing juga menyediakan library Java 2D untuk pengolahan data secara visual, seperti mengolah gambar, object 2D, bahkan animasi. SwingLabs.org menyediakan library Swing Painter yang merupakan pengembangan dari Java 2D, Swing Painter ini memungkinkan aplikasi swing mempunyai tampilan yang indah dan terlihat profesional.

Java 6.0 menambahkan banyak sekali fitur-fitur baru ke dalam package swing, termasuk dukungan untuk library OpenGL menggunakan JOGL, Tray Icon dan Web Service. Dengan adanya

dukungan ini swing menjadi lebih powerful dan mempunyai masa depan yang cerah.

Struktur Komponen Swing

Secara arsitektur, Swing dibangun diatas arsitektur AWT (Abstract Windows Toolkit). AWT adalah GUI toolkit yang dikembangkan oleh Sun engineer sebelum swing muncul. Kelemahan utama AWT adalah fleksibilitas tampilan GUI, seperti painting method yang masih sangat primitif.

Swing dimaksudkan untuk memperbaiki kekurangan dari AWT tanpa harus membuang teknologi yang sudah dibuat dan membuat GUI toolkit baru dari nol.

Komponen AWT diletakkan dalam satu package yaitu java.awt, didalamnya terdapat komponen-komponen GUI dasar, salah satunya adalah Component. Class Component adalah moyang dari sebagian besar komponen AWT maupun Swing. CheckBox, Label, Button dan beberapa komponen AWT lainnya adalah turunan langsung dari class Component. Namun dalam kenyataanya arsitektur demikian tidak memberikan fleksibilitas yang cukup memadai untuk membuat berbagai macam komponen baru yang dibutuhkan dalam desktop application.

Swing muncul dengan membawa teknologi AWT yang telah ditambahkan dengan banyak kemampuan. Nyaris semua komponen GUI dari swing merupakan turunan class Container dan class Container adalah turunan dari class Component.

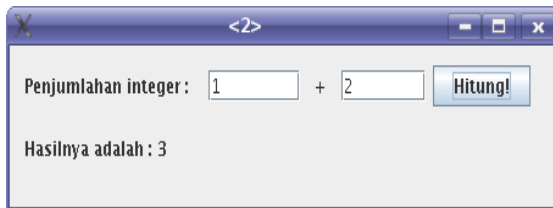
Bekerja dengan JLabel, JTextField dan JButton

Bekerja dengan komponen swing menggunakan Matisse sangat

menyenangkan dan mudah. GroupLayout yang sangat fleksibel memungkinkan kita untuk membuat aplikasi dengan tampilan seperti yang kita harapkan.

Label, textfield dan tombol adalah komponen-komponen dasar yang selalu ada dalam setiap aplikasi berbasis desktop. Ketiga komponen ini mempunyai fungsi yang sangat sederhana, textfield menyimpan data berbentuk text (string) yang relatif pendek, label banyak digunakan untuk memberikan keterangan penjelas terhadap komponen lain dan tombol digunakan user untuk menjalankan satu instruksi tertentu.

Berikut ini adalah contoh aplikasi sederhana yang melakukan penjumlahan dua buah bilangan.



Contoh program menggunakan JLabel, JTextField dan JButton

Untuk membuat aplikasi ini menggunakan Matisse, lakukan langkah-langkah berikut ini:

1. Buat project baru di Netbeans (kalau sudah membuat project, tidak perlu membuat lagi) dengan cara memilih menu :

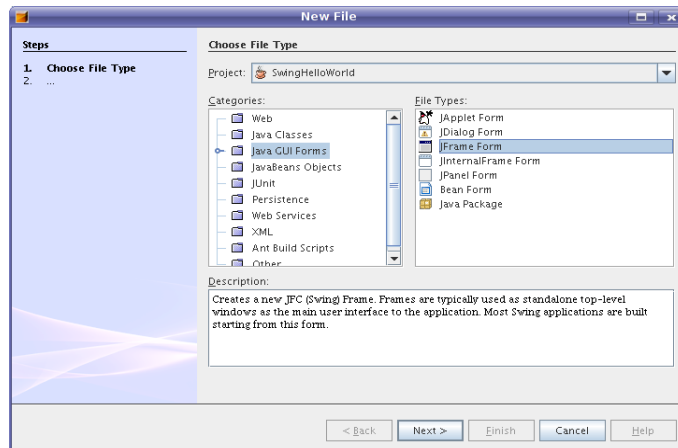
File > New Project

Kemudian ikuti petunjuk yang diberikan dialog.

2. Buat class JFrame baru, caranya dengan memilih menu :

File > New File

Kemudian akan muncul dialog seperti di bawah ini :



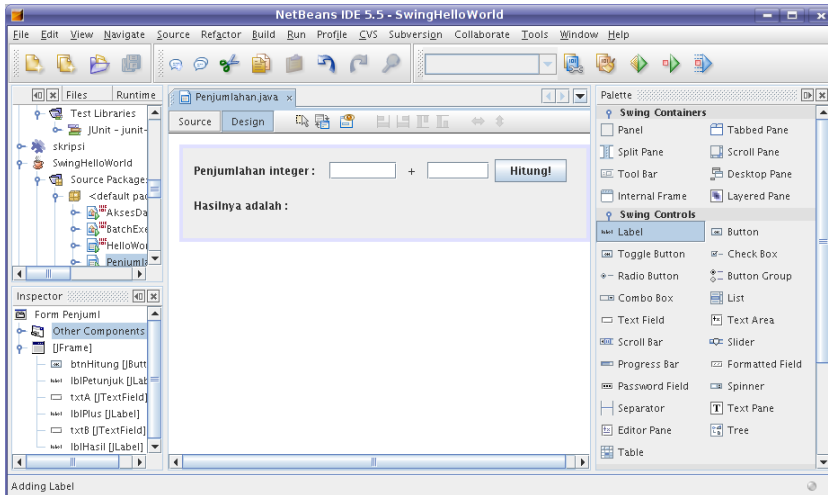
Jendela dialog new file

3. Pilih kategori :

Java GUI Forms > JFrame Form

Seperti terlihat di dialog New File dialog diatas, kemudian beri nama Penjumlahan.java

4. Buat tampilan form seperti gambar bawah ini, caranya dengan klik Jendela Pallete di sebelah kanan untuk memilih komponen apa yang akan dibuat, kemudian klik di jendela Design untuk menempatkan komponen yang sudah dipilih tadi ke dalam form. Hasilnya terlihat seperti pada gambar di bawah ini:



Jendela design Netbens Matisse

5. Ganti nama setiap komponen agar mudah dikenali. Klik kanan diatas setiap komponen yang ada dalam Jendela Design diatas, kemudian pilih menu :

```
Klik kanan > Change Variable Name ...
```

Ganti nama komponen-komponen tersebut (sesuai urutan dari kiri ke kanan, atas ke bawah) menjadi : lblKeterangan, txtA, lblPlus, txtB, btnHitung, lblHasil.

6. Menambahkan variable untuk menampung nilai yang akan dijumlahkan. Klik tombol Source untuk membuka jendela yang menampilkan kode sumber dari program di atas kemudian tambahkan kode di bawah ini tepat dibawah definisi dari class Penjumlahan:

```
private String str = "Hasilnya adalah : ";  
private int a, b;
```

7. Menangani penekanan tombol btnHitung. Klik kanan diatas komponen btnHitung kemudian pilih menu :

penting

Jendela Design menampilkan visualisasi komponen GUI.

Jendela Source menampilkan kode program dari class yang sedang dibuka.

Jendela Swing Pallete berisikan komponen-komponen swing yang bisa kita drag-and-drop ke dalam jendela design.

Jendela Properties digunakan untuk mengedit properti dari komponen yang sedang aktif dalam jendela design.

Jendela Inspector menampilkan semua komponen swing dalam class yang sedang aktif baik yang kelihatan secara visual di jendela design atau tidak.

Events > Action > actionPerformed

Anda akan dibawa ke jendela Source, dan akan menemukan kode program seperti di bawah ini :

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

Ubah kode program diatas menjadi :

```
private void btnHitungActionPerformed(
    java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    a = Integer.parseInt(txtA.getText());
    b = Integer.parseInt(txtB.getText());
    int hasil = a + b;
    lblHasil.setText(str + hasil);
}
```

penting

JLabel dan JTextField mempunyai method `getText` dan `setText` untuk mengambil dan mengeset text yang ditampilkan.

8. Compile dan jalankan program. Tekan tombol SHIFT + F6, atau klik kanan file Penjumlahan.java kemudian pilih menu Run File.

Catatan :

- Method `Integer.parseInt` digunakan untuk merubah String menjadi Integer.
- Method `btnHitungActionPerformed` akan dipanggil setiap kali kita memencet tombol `btnHitung`.

Sekarang anda bisa melihat bahwa bekerja dengan JLabel, JTextField dan JButton sangat sederhana. Untuk latihan, silahkan rubah fungsi yang digunakan dalam program diatas, misalnya perkalian dua bilangan atau pengurangan dua bilangan.

Bekerja dengan JCheckBox dan JRadioButton

JCheckBox dan JRadioButton hanya bisa mempunyai dua buah kemungkinan nilai, benar atau salah. Kedua komponen ini digunakan untuk merepresentasikan data yang berupa pilihan. JCheckBox digunakan jika pilihannya berupa multiple selection, sedangkan JRadioButton digunakan jika pilihannya berupa single selection.

JRadioButton digunakan misalnya untuk merepresentasikan pilihan jenis kelamin. JCheckBox digunakan misalnya untuk merepresentasikan pilihan hobby.

ButtonGroup diperlukan untuk mengumpulkan JRadioButton yang mempunyai grup pilihan yang sama. Misalnya grup pilihan jenis kelamin digunakan untuk mengumpulkan JRadioButton yang merepresentasikan pilihan laki-laki dan JRadioButton yang merepresentasikan pilihan perempuan dalam satu group. Jika JRadioButton tidak diletakkan dalam satu group, maka pilihan laki-laki dan pilihan perempuan bisa dipilih bersamaan.

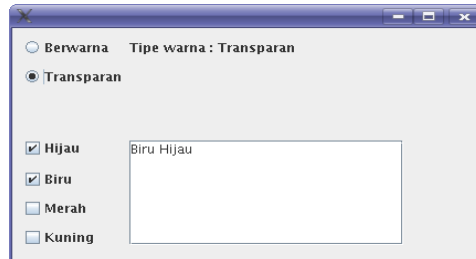
Status dari JRadioButton dan JCheckBox dapat diketahui dengan melihat nilai kembalian dari method isSelected, jika dipilih maka nilai kembalian method isSelected adalah benar, dan false jika sebaliknya.

Setiap JRadioButton dan JCheckBox mempunyai text yang menerangkan pilihan yang diwakilinya. Method getText dan setText digunakan untuk memanipulasi text.

Dibawah ini adalah contoh program yang menggunakan JCheckBox dan JRadioButton.

penting

JCheckBox dan JRadioButton sebaiknya digunakan hanya jika item pilihannya sedikit dan tidak bersifat dinamis



Contoh aplikasi menggunakan JCheckBox dan JRadioButton

Di bagian atas aplikasi ini, terdapat dua JRadioButton untuk merepresentasikan pilihan tipe warna, transparan atau berwarna. Dibawahnya terdapat pilihan warna yang dapat dipilih lebih dari satu buah menggunakan JCheckBox.

Untuk membuat program diatas ikuti langkah-langkah berikut ini:

1. Buat class baru bertipe JFrame Form, kemudian beri nama Pilihan.java
2. Buat tampilan diatas menggunakan Matisse. komponen yang harus dibuat adalah :
 - Dua object JRadioButton : radioBerwarna dan radioTransparan.
 - Satu object ButtonGroup : groupTipeWarna.
 - Empat object JCheckBox : chkHijau, chkBiru, chkMerah, chkKuning.
 - Satu object JTextArea : txtWarna.
 - Satu object JScrollPane : scrollWarna

Untuk melihat semua komponen yang ada dalam Jendela Design, gunakan Jendela Inspector di sisi kiri bawah.

3. Masukkan object radioBerwarna dan radioTransparan ke dalam

object groupTipeWarna. Caranya dengan :

- a) Memilih komponen radioBerwarna di Jendela Design
- b) Klik tab code di Jendela Properties
- c) Pilih properti : Post-Creation Code
- d) Masukkan kode berikut ini kedalam dialog yang muncul :

```
groupTipeWarna.add(radioBerwarna);
```

Lakukan langkah yang sama terhadap object radioTransparan.

4. Menangani event ketika JRadioButton diklik. Caranya dengan :

- a) Memilih komponen radioBerwarna di Jendela Design
- b) Klik kanan komponen radioBerwarna, kemudian pilih menu:

```
Event > Action > actionPerformed
```

- c) Anda akan dibawa ke dalam Jendela Code, dan menemukan kode berikut ini :

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

Ubahlah kode diatas menjadi :

```
private void radioBerwarnaActionPerformed(
    java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(radioBerwarna.isSelected()){
        lblTipeWarna.setText("Tipe warna : " +
            radioBerwarna.getText());
    }
}
```

Lakukan langkah yang sama terhadap radioTransparan.

penting

JRadioButton yang mempunyai group yang sama, harus dimasukkan dalam sebuah object ButtonGroup yang sama.

5. Buat sebuah private method untuk menangani event pemilihan terhadap JCheckBox. Method `tampilkanWarna` ini nantinya akan dipanggil setiap kali salah satu dari JCheckBox dipilih. yang dilakukan oleh metod `tampilkanWarna` adalah mengecek status setiap JCheckBox, apakah sedang dipilih atau tidak. Jika sedang dipilih maka text dari JCheckBox tersebut akan ditampilkan dalam `txtWarna`.

Class `StringBuffer` digunakan untuk menampung nilai text dari JCheckBox yang statusnya terpilih.

penting

Class `StringBuffer` sangat dianjurkan untuk digunakan sebagai class untuk memanipulasi `String`.

Penggabungan string menggunakan operator `+` sangat tidak dianjurkan, apalagi jika ukuran object `String`-nya sudah cukup besar.

```
private void tampilkanWarna(){
    StringBuffer warna = new StringBuffer();
    if(chkBiru.isSelected()){
        warna.append(chkBiru.getText() + " ");
    }
    if(chkHijau.isSelected()){
        warna.append(chkHijau.getText() + " ");
    }
    if(chkKuning.isSelected()){
        warna.append(chkKuning.getText() + " ");
    }
    if(chkMerah.isSelected()){
        warna.append(chkMerah.getText() + " ");
    }
    txtWarna.setText(warna.toString());
}
```

6. Menangani event pemilihan JCheckBox. Caranya sebagai berikut :
 - a) Pilih komponen `chkHijau` di `Jendela Design`.
 - b) Klik kanan komponen `chkHijau` untuk memunculkan context (popup) menu.

c) Pilih menu :

Event > Action > actionPerformed

d) Anda akan dibawa ke Jendela Code, kemudian dalam method `chkHijauActionPerformed` tersebut panggil method `tampilkanWarna`. seperti di bawah ini :

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

Lakukan hal ini untuk semua `JCheckBox`.

7. Compile dan jalankan program dengan menekan tombol `SHIFT + F6`.

Cara lain dalam menampilkan pilihan adalah dengan menggunakan `JList` dan `JComboBox`. Kedua komponen ini mempunyai fleksibilitas yang lebih tinggi dan lebih mudah digunakan jika object yang dimasukkan dalam pilihan lebih kompleks. `JList` dan `JComboBox` bisa mempunyai `ComponentEditor` agar pilihan yang ditampilkan tidak hanya berupa text, bisa berupa warna atau icon. Bagian berikutnya akan membahas bagaimana bekerja menggunakan `JList` dan `JComboBox`.

Bekerja dengan `JList` dan `JComboBox`

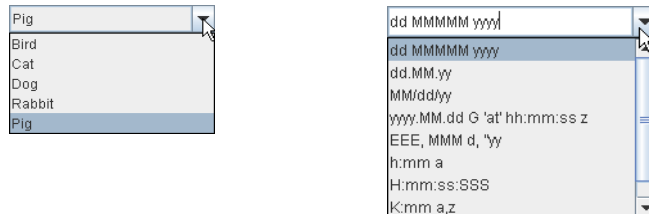
`JComboBox` memerlukan tempat yang minimalis dibandingkan dengan `JRadioButton`, selain itu `JComboBox` mempunyai bentuk `ComboBox` yang dapat diedit, sehingga memungkinkan user untuk memilih pilihan yang tidak ada dalam item `JComboBox`.

penting

`JComboBox` dan `JList` digunakan jika item pilihan bersifat dinamis.

`JComboBox` dapat mempunyai bentuk yang dapat diedit sehingga user dapat memasukkan pilihan yang tidak ada dalam daftar.

`JList` dapat menerima pilihan lebih dari satu.

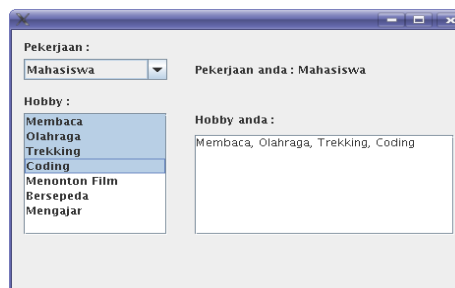


Contoh JComboBox

JList memungkinkan multiple selection dengan menekan tombol : SHIFT + Left Click atau CTRL + Left Click. Kemampuan ini membantu user jika harus melakukan multiple selection.

JComboBox dan JList sangat fleksibel, kita dapat menambah dan menghapus item di dalamnya dengan sangat mudah. Sehingga cocok digunakan untuk merepresentasikan pilihan yang item pilihannya bersifat dinamis.

Aplikasi di bawah ini adalah contoh penggunaan JComboBox dan JList.



Contoh program menggunakan JComboBox dan JList

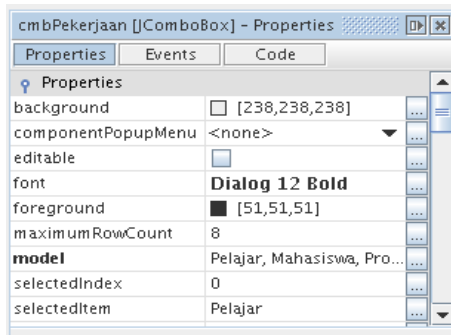
Bagian pertama program ini terdapat sebuah JComboBox dan JLabel, setiap kali item di dalam JComboBox dipilih, JLabel di sebelahnya akan menampilkan item yang dipilih tersebut.

Bagian kedua program ini terdapat sebuah JList dan JTextArea. Setiap kali item-item di dalam JList dipilih, JTextArea akan menampilkan item-item yang dipilih tersebut dipisahkan dengan

koma (,).

Ikuti langkah-langkah berikut ini untuk membuat program di atas:

1. Buatlah class JFrame Form baru dan beri nama ListAndCombo.java.
2. Buat tampilan program diatas menggunakan Matisse, kemudian tambahkan komponen-komponen:
 - a) Empat buah JLabel : lblPekerjaan, lblPilihanPekerjaan, lblHobby, lblPilihanHobby.
 - b) Satu buah JComboBox : cmbPekerjaan
 - c) Satu buah JList : lstHobby
 - d) Satu buah JTextArea : txtPilihanHobby
3. Merubah isi JComboBox. Untuk merubah isi dari JComboBox dan JList kita akan menggunakan Jendela Properties, Jendela ini letaknya di sebelah kanan bawah, dibawah Jendela Pallette dan akan muncul hanya jika jendela Design yang dipilih.



Jendela Properties

Pilih komponen JComboBox di Jendela Design, Jendela Properties akan menampilkan properties dari JComboBox.

Pada bagian model di dalam Jendela Properties masukkan item Pelajar, Mahasiswa, Programmer, Technical Writer dan Tester.

penting

Jendela Properties tidak hanya berisi properties dari komponen swing yang sedang aktif tetapi juga berisi Tab Events dan Tab Code.

Tab Events digunakan untuk memasukkan kode yang akan dieksekusi ketika event tertentu dikenakan terhadap komponen swing.

Tab Code digunakan untuk mendefinisikan kode apa yang harus dieksekusi dalam kondisi tertentu, misalnya setelah object komponen swing diinisialisasi.

Setiap item dipisahkan dengan koma (,).

4. Merubah isi JList. Pilih JList di Jendela Design maka Jendela Properties untuk JList akan muncul. Di bagian model isikan item : Membaca, Olahraga, Trekking, Coding, Menonton Film, Bersepeda dan Mengajar. Setiap item dipisahkan dengan koma (,).
5. Menangani pemilihan JComboBox. Klik kanan JComboBox di Jendela Design, kemudian pilih menu :

Events > Action > actionPerformed

Jendela Code akan terbuka, tambahkan code seperti di bawah ini :

```
private void cmbPekerjaanActionPerformed(
    java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    lblPilihanPekerjaan.setText(
        "Pekerjaan anda : " +
        cmbPekerjaan.getSelectedItem());
}
```

method `getSelectedItem` dari JComboBox digunakan untuk memperoleh item yang sedang di pilih dalam JComboBox.

6. Menangani event pemilihan dari JList. Event yang digunakan untuk menangani pemilihan item dari JList berbeda dengan JComboBox. JList akan mengaktifkan `ListSelection` event ketika user memilih item dalam JList. Untuk menangani event ini, lakukan langkah-langkah berikut :

- a) Klik kanan pada JList di dalam Jendela Design, kemudian pilih menu :

Events > ListSelection > valueChanged

- b) Dalam jendela kode yang ketik kode seperti berikut ini :

```

private void lstHobbyValueChanged(
    javax.swing.event.ListSelectionEvent evt) {
    // TODO add your handling code here:
    Object[] selectedItems =
        lstHobby.getSelectedValues();
    if(selectedItems == null ||
        selectedItems.length == 0)
        txtPilihanHobby.setText("");
    else{
        StringBuffer strValues = new StringBuffer();
        for(Object item : selectedItems){
            strValues.append(item.toString() + ", ");
        }
        txtPilihanHobby.setText(
            strValues.substring(0, strValues.length() -
                2));
    }
}

```

Catatan :

- Method `getSelectedValues` dari `JList` mengembalikan item-item yang terpilih.

Bekerja dengan Menu, Popup Menu dan Toolbar

Menu, Popup menu dan Toolbar digunakan untuk melakukan navigasi dalam aplikasi. dengan ketiga komponen itu navigasi dalam aplikasi menjadi lebih fleksibel dan mudah digunakan oleh user. Menu dan Toolbar pada umumnya diletakkan di bagian atas dari aplikasi agar mudah ditemukan oleh user. Sedangkan Popup

penting

JMenuBar dan JToolBar hanya bisa ditambahkan ke dalam `JFrame`.

JMenuItem adalah struktur terluar dari **Menu** yang tidak bisa mempunyai **child**.

JToolBar pada umumnya menampung **JButton** yang diberi **Icon** dan mempunyai **background** transparan.

Menu bisa muncul di mana saja sesuai dengan konteks aplikasi.

JMenuBar adalah class yang digunakan untuk menampung JMenu. JMenu dapat menampung satu atau lebih JMenuItem. JMenuItem merupakan bagian terluar dari struktur menu yang tidak bisa mempunyai child. JSeparator digunakan untuk memisahkan antara satu menu item dan menu item yang lain. Jika didalam menu terdapat sub menu, gunakan JMenu untuk menampung sub menu tersebut. Selain JMenuItem, JMenu juga dapat menerima class JCheckBoxMenuItem dan JRadioButtonMenuItem.

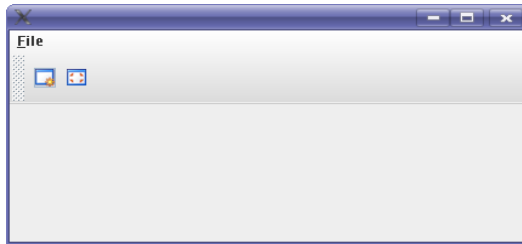
JPopupMenu mempunyai banyak kesamaan dibandingkan dengan JMenuBar. Perbedaan utamanya adalah : JMenuBar hanya bisa berada di atas sebuah jendela JFrame. Sedangkan JPopupMenu bisa muncul di mana saja sesuai dengan konteks dari aplikasi.

Perbedaan lainnya terletak di dalam penggunaan umum keduanya. JMenuBar berisikan menu/instruksi yang bersifat umum dan berlaku untuk semua keadaan dalam aplikasi. Sedangkan JPopupMenu akan mempunyai menu/instruksi yang berbeda-beda berdasarkan dari konteks aplikasi. Oleh karena itu JPopupMenu terkadang disebut juga sebagai konteks menu.

Toolbar memberikan cara yang lebih praktis dibandingkan menu, bahkan bisa dikatakan bahwa toolbar adalah cara cepat untuk mengakses menu. Oleh karena itu, setiap item dalam toolbar biasanya juga tersedia dalam menu. Pada umumnya toolbar diwakili hanya dengan gambar/icon yang melambangkan perintah dari toolbarnya. Di internet banyak tersedia toolbar icon gratis yang dapat kita gunakan.

Berbeda dengan JMenuBar dan JPopupMenu yang hanya bisa menerima menu item, JToolBar dapat menampung JButton atau control lainya. Seperti contohnya : JCheckBox, JRadioButton, JToggleButton dan lainya. Normalnya, JToolBar akan diisi dengan

JButton yang dihilangkan text-nya dan diganti dengan icon. Kita juga perlu merubah dekorasi JButton agar tampilannya terlihat cantik dan menarik.



Contoh program dengan Menu, Popup Menu dan Toolbar

Untuk membuat program seperti di atas, ada beberapa tahap yang perlu dilakukan. Tahap pertama adalah membuat Menu, yang kedua adalah membuat Popup Menu dan yang ketiga adalah membuat Toolbar.

Membuat Menu

Bekerja dengan Menu dalam Java melibatkan enam komponen swing, antara lain :

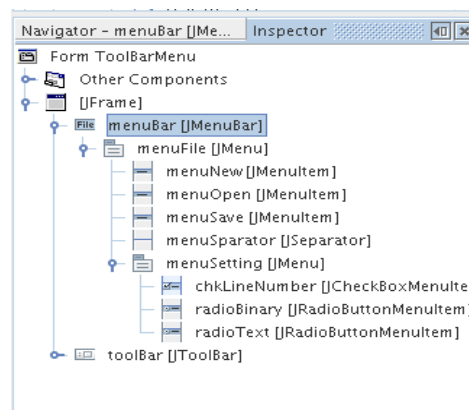
1. JMenuBar : Class yang menampung semua menu, hanya bisa menampung JMenu sebagai child.
2. JMenu : Class yang mempunyai child menu item. Biasanya JMenu ini yang jadi child langsung dengan JMenuBar
3. JMenuItem : Ujung dari menu, disinilah object Action diletakkan, sehingga ketika kita memilih JMenuItem ada action tertentu yang dijalankan aplikasi.
4. JCheckBoxMenuItem : Ujung dari menu, namun bentuknya lebih mirip JCheckBox.
5. JRadioButtonMenuItem : Ujung dari menu, namun bentuknya lebih mirip JButton.

6. JSeparator : pemisah antar JMenuItem atau antar JMenu

Setiap komponen menu mempunyai fungsi dan kegunaan masing-masing. Jika kita perhatikan, menu dalam aplikasi umumnya mempunyai shortcut untuk mengakses menu tanpa menggunakan bantuan mouse. Misalnya menu File biasanya dapat diakses menggunakan tombol ALT + F, menu Format dapat diakses dengan ALT + O. Fasilitas shortcut menu ini disebut sebagai Keyboard Mnemonic. File mempunyai mnemonic F, Format mempunyai mnemonic o, dan seterusnya. Pada umumnya tampilan mnemonic dari sebuah menu diwakili dengan huruf yang bergaris bawah.

Matisse mempunyai fasilitas yang sangat OK untuk bekerja dengan Menu, fasilitas drag-and-dropnya membantu banyak pekerjaan membangun aplikasi berbasis GUI secara signifikan.

Dalam program diatas kita akan membuat struktur menu sebagai berikut:



Struktur menu dari aplikasi

Ikuti langkah-langkah berikut ini untuk membuat struktur menu seperti diatas:

1. Buat sebuah class JFrame dan beri nama ToolbarMenu.java
2. Menambahkan JMenuBar ke dalam JFrame. Pilih komponen

penting

Jendela Inspector akan memperlihatkan semua komponen swing baik yang terlihat atau tidak terlihat dalam jendela design.

Jendela Inspector sangat berguna ketika kita bekerja dengan Menu.

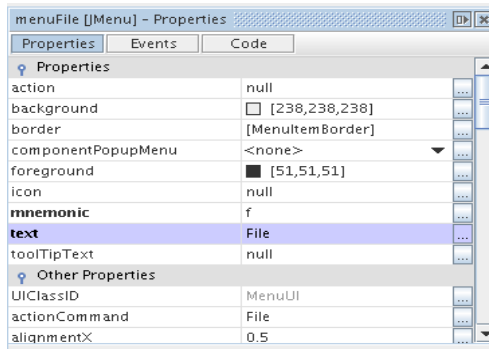
Proses penambahan, pengurangan dan pengaturan posisi menu semua dilaksanakan dari Jendela Inspector

Menu Bar dari Jendela Pallette kemudian klik JFrame di Jendela Design. Sebuah class JMenuItem akan ditambahkan di dalam JFrame. Ganti namanya menjadi menuBar.

- Menambahkan JMenuItem ke dalam JMenuItem. Klik kanan JMenuItem yang baru saja kita buat di Jendela Inspector, kemudian pilih menu :

Add > JMenuItem

Ganti nama JMenuItem tersebut menjadi menuFile. Kemudian alihkan perhatian anda ke Jendela Properties



Jendela Properties dari class JMenuItem

Isi properti text dengan string "File". Kemudian set isi properti mnemonic dengan string "f", hal ini akan menyebabkan tampilannya menuFile menjadi File dan user dapat menekan tombol ALT + F untuk mengaktifkan menu menuFile.

- Menambahkan JMenuItem. Langkah berikutnya adalah menambahkan JMenuItem ke dalam JMenuItem menuFile yang telah dibuat di langkah sebelumnya. caranya, klik kanan di JMenuItem menuFile di Jendela Inspector, kemudian pilih menu :

Add > JMenuItem

Tambahkan berturut-turut menuNew, menuOpen dan menuSave. Pilih JMenuItem dari Jendela Inspector, kemudian

untuk masing-masing JMenuItem set text dan mnemonic yang sesuai dari Jendela Properties.

5. Menambahkan JSeparator. Dalam struktur menu yang bagus, menu yang mempunyai fungsi serupa diletakkan dalam urutan berderdekatan dan dipisahkan dengan separator (pemisah). Langkah menambahkan JSeparator tidak berbeda dengan langkah menambahkan JMenuItem, klik kanan di JMenuItem menuFile kemudian pilih menu:

Add > JSeparator

6. Menambahkan JMenuItem. Berikutnya kita akan menambahkan JMenuItem baru ke dalam JMenuItem menuFile. JMenuItem yang baru ini akan bertindak sebagai sub menu. Caranya juga sama : klik kanan di JMenuItem menuFile kemudian pilih menu :

Add > JMenuItem

Beri nama menuSetting, set text dan mnemonic yang sesuai pada Jendela Properties.

7. Menambahkan JCheckBoxMenuItem. Perilaku JCheckBoxMenuItem tidak berbeda jauh dengan JCheckBox biasa, bedanya hanyalah JCheckBoxMenuItem berada dalam struktur menu. Cara menambahkan JCheckBoxMenuItem sama dengan komponen lain : klik kanan JMenuItem menuSetting kemudian pilih menu :

Add > JCheckBoxMenuItem

Beri nama chkLineNumber, set text dan mnemonic yang sesuai pada Jendela Properties.

JCheckBoxMenuItem sedikit spesial dibandingkan dengan JMenuItem, karena JCheckBoxMenuItem memiliki properties selected. Properties selected ini digunakan untuk menentukan apakah JCheckBoxMenuItem dalam keadaan terpilih atau tidak.

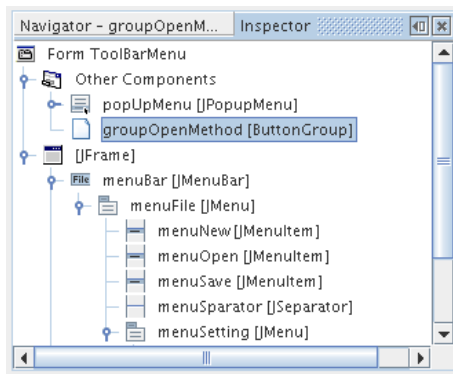
8. Menambahkan JRadioButtonMenuItem. Dalam contoh ini kita akan mempunyai dua buah JRadioButtonMenuItem, radioBinary dan radioText. Keduanya dibuat dengan langkah yang sama dengan komponen lain, klik kanan di JMenu menuSetting, kemudian pilih menu :

Add > JRadioButtonMenuItem

Set text dan mnemonic yang sesuai dari Jendela Properties.

9. Menambahkan ButtonGroup. Seperti halnya JRadioButton, JRadioButtonMenuItem juga memerlukan ButtonGroup agar hanya satu buah JRadioButtonMenuItem yang bisa dipilih. Cara menambahkan ButtonGroup sangat mudah, klik item ButtonGroup dari Jendela Pallette kemudian klik Jendela Design, maka otomatis ButtonGroup akan ditambahkan. Ganti namanya menjadi groupOpenMethod.

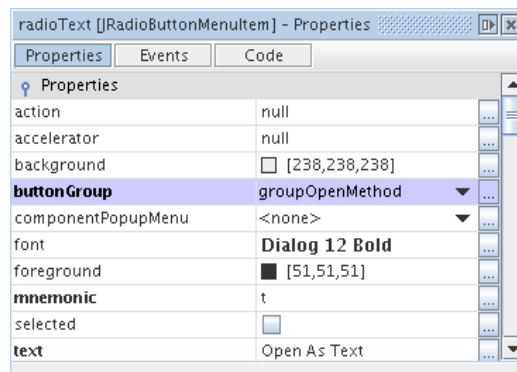
Dalam Jendela Inspector, ButtonGroup yang baru dibuat tadi akan berada dalam kategori Other Components, seperti terlihat dalam gambar di bawah ini :



ButtonGroup berada dalam kategori Other Components

10. Menambahkan JRadioButtonMenuItem ke dalam ButtonGroup. Pilih masing-masing JRadioButtonMenuItem dari Jendela

Inspector, kemudian perhatikan jendela Properties dari JRadioButtonMenuItem tersebut, pada bagian groupButton pilih item groupOpenMethod, seperti terlihat dalam gambar di bawah ini :



Properties dari JRadioButtonMenuItem

11. Compile dan jalankan class ToolbarMenu.java. Klik kanan class ToolbarMenu dari Jendela Design kemudian pilih menu Run File atau tekan tombol SHIFT + F6.

Bekerja dengan Menu menggunakan Matisse sangatlah menyenangkan dan produktif. Hal ini berbeda sekali jika harus mengetik satu demi satu kode untuk menyusun struktur menu seperti contoh program diatas.

Membuat Popup Menu menggunakan Matisse juga sama mudahnya. Hanya saja kita harus menentukan dimana dan dengan cara apa popup menu itu muncul, apakah dengan penekanan tombol tertentu dari keyboard atau ketika tombol mouse ditekan.

Membuat Popup Menu

Popup menu pada dasarnya tidak jauh berbeda dibandingkan dengan menu biasa, hanya saja popup menu dapat muncul di mana saja, tidak hanya di bagian atas JFrame seperti halnya JMenuBar.

Selain itu kita harus menentukan kapan popup muncul, pada umumnya popup akan muncul ketika user melakukan klik kanan terhadap suatu komponen swing. Misalnya, ketika suatu table di klik kanan terdapat popup yang muncul, dan sebagainya.

Popup menu terutama digunakan sebagai “context sensitive menu”, dimana menu yang ditampilkan oleh popup menu tergantung konteks dari aplikasi, semisal : komponen apa yang dikenai aksi klik kanan, bagaimana keadaan data dalam komponen tersebut dan sebagainya.

Aplikasi yang memerlukan interaksi yang sangat intens dengan user sebaiknya menggunakan popup menu untuk memudahkan user mengakses action tertentu. Hal ini jauh lebih praktis dibanding user harus mengakses menu dalam JMenuBar di bagian atas JFrame.

Popup menu dalam contoh program diatas akan muncul ketika user melakukan klik kanan terhadap JFrame. menu yang ditampilkanya pun hanya ada tiga buah: cut, copy dan paste.

Ikuti langkah-langkah berikut ini untuk membuat Popup menu :

1. Buka class ToolbarMenu.java, yang telah dibuat dalam langkah sebelumnya, dalam Jendela Design.
2. Klik Jendela Palette dan pilih JPopupMenu, kemudian klik Jendela Design. Secara otomatis JPopupMenu akan ditambahkan dalam class ToolbarMenu.java. JPopupMenu tidak terlihat dalam Jendela Design, namun anda bisa mengkasasnya melalui Jendela Inspector.
3. Menambahkan JMenuItem. Seperti halnya JMenuBar, JPopupMenu dapat memiliki child berupa JMenuItem, JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem dan JSeparator. Menambahkan JMenuItem ke dalam JPopupMenu sangat

sederhana, caranya : klik kanan pada JPopupMenu di Jendela Design, kemudian pilih menu :

Add > JMenuItem

Ganti nama objectnya menjadi menuCut, beralihlah ke Jendela Properties kemudian set text dan mnemonic yang sesuai.

Lakukan langkah ini untuk JMenuItem yang lain, menuCopy dan menuPaste.

4. Memunculkan JPopupMenu. Ketika tombol kanan mouse di klik diatas JFrame, JPopupMenu akan tampil. Agar behavior tersebut berjalan, kita perlu menangani event mouseClicked terhadap JFrame. Caranya :

- a) Klik kanan JFrame di Jendela Design, kemudian pilih menu :

Events > Mouse > mouseClicked

- b) Di dalam jendela source yang terbuka masukkan kode berikut ini :

```
private void formMouseClicked(
    java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    if(evt.getButton() == MouseEvent.BUTTON3){
        popUpMenu.show(
            (Component)evt.getSource(),
            evt.getX(),evt.getY());
    }
}
```

Kondisi if diatas digunakan apakah tombol yang diklik mouse adalah tombol sebelah kanan, jika nilai kembalian method getButton sama dengan nilai BUTTON3 maka benar tombol kanan yang ditekan.

Method show digunakan untuk memunculkan popup menu,

parameter pertama diisi dengan Component dimana nantinya popup menu akan ditampilkan, sedangkan parameter kedua dan ketiga diisi dengan letak koordinat popup menu akan ditampilkan.

5. Simpan file `ToolbarMenu.java`, compile dan jalankan. Kemudian coba munculkan popup menu dengan mengklik kanan `JFrame`.

Popup menu sangat berguna jika aplikasi yang kita kembangkan membutuhkan interaksi yang intensif dengan user. Popup menu menyediakan cara mudah untuk mengakses menu/action yang sesuai dengan konteks dari aplikasi.

Membuat Toolbar

Toolbar memberikan dimensi lain dalam mengakses menu dibandingkan menu ataupun popup menu. Pada umumnya Toolbar merupakan cara singkat untuk mengakses menu. Menu yang diwakili toolbar adalah menu yang bersifat umum dan tidak terikat pada konteks tertentu.

Kegunaan lain dari toolbar adalah mempercantik tampilan aplikasi, karena toolbar biasanya adalah tombol yang didekorasi dengan icon yang menarik. Selain itu toolbar juga memberikan kesempatan kepada user untuk mengkustomisasi tampilan dari aplikasi. Karena layout toolbar sangat fleksibel, user bisa memindah-mindahkan letak toolbar di dalam aplikasi, di atas, dibawah atau disamping, atau bahkan mengambang (floating) diatas jendela yang sedang aktif.

Dalam contoh program diatas kita akan membuat sebuah `JToolBar` dengan dua buah `JButton` yang telah didekorasi dengan icon cantik. Icon yang digunakan banyak tersedia di internet, format file yang dipilih adalah `.png`, karena format file ini paling bagus dalam menangani transparansi komponen.

Sebelum mulai membuat JToolBar, kita perlu mempersiapkan terlebih dahulu icon yang akan digunakan sebagai dekorasi JButton. Ikuti langkah-langkah berikut ini :

1. Buatlah sebuah java package baru untuk menampung semua icon yang akan digunakan. caranya klik kanan di jendela Projects bagian nama project, pilih menu :

New > Java Package

Beri nama images untuk java package yang baru saja kita buka.

2. Memasukkan Icon ke dalam package. Untuk memasukkan image ke dalam package kita perlu tahu dimana project disimpan, misalkan project disimpan dalam folder :

c:\jawaswing

Buka file explorer, kemudian navigasi ke folder

c:\jawaswing\src\images

Copy semua icon yang diperlukan ke dalam folder diatas.

3. Build project. Langkah ini diperlukan untuk mengcompile semua file .java menjadi file .class. Selain itu proses ini juga akan mengkopi file selain file .java (termasuk file icon) ke dalam folder build\classes. Jika proses ini tidak dilaksanakan, maka ketika program dijalankan, file icon tidak akan ditemukan dan program menjadi error .

Setelah proses persiapan selesai, lakukan langkah-langkah berikut ini untuk membuat Toolbar :

1. Buka class ToolbarMenu.java yang sudah dibuat di langkah sebelumnya.
2. Buat sebuah object JToolBar, caranya : klik item JToolBar dari Jendela Pallete, kemudian klik JFrame di Jendela Design. Secara otomatis sebuah object JToolBar akan dimasukkan ke

penting

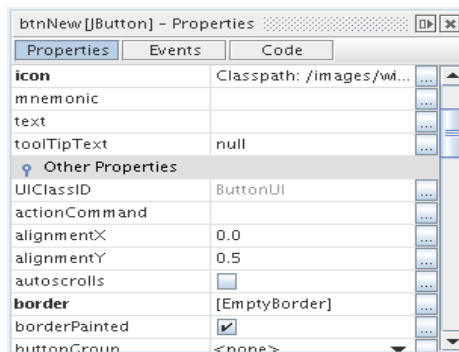
Build Project akan berhasil jika tidak ada satupun error dalam kode program.

Sebelum melakukan build project pastikan terlebih dahulu tidak ada error dalam kode.

Lakukan build setiap kali menambahkan file non-java ke dalam folder source file. Agar file tersebut ikut tercopy ke dalam folder build\classes

dalam JFrame. Ganti namanya menjadi toolBar.

3. Menambahkan JButton dalam JToolBar. Klik item JButton dalam Jendela Pallete kemudian klik komponen JToolBar yang baru saja kita buat tadi. JButton baru akan diletakkan diatas JToolBar, ganti nama JButton tersebut menjadi btnNew. Letakkan lagi satu buah JButton diatas JToolBar dan beri nama btnMaximize.
4. Mendekorasi Tampilan JButton. Agar tampilan JButton terlihat cantik, kita perlu mengeset beberapa nilai dari properti JButton, seperti terlihat pada gambar di bawah ini :



Jendela Properties JButton

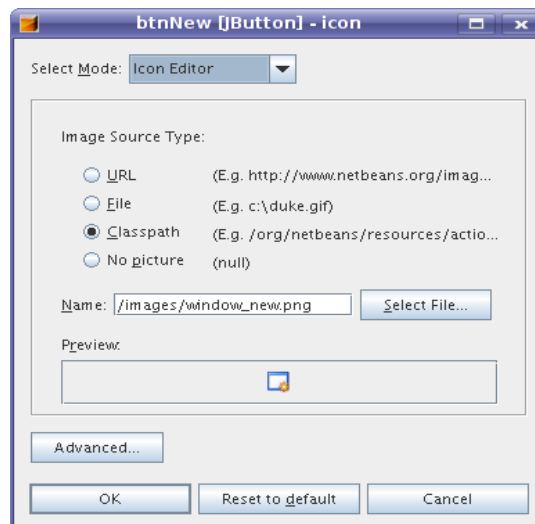
- a) Text, hapus nilai textnya.
- b) Border, pilih bordernya menjadi empty border dan set nilai bordernya menjadi [5,5,5,5]. Tujuan pemberian empty border ini agar tombol berukuran lebih besar dibandingkan dengan icon yang akan digunakan nanti, dan setiap mouse melewati JButton, ada efek transisi yang cantik.

Untuk mengedit border dari JButton, Matisse menyediakan Jendela Border untuk memilih border yang kita inginkan untuk JButton. Border yang dipilih bisa single border, atau composite border yang terdiri dari beberapa border.



Jendela Border Editor dari JButton

- c) Opaque, uncheck nilai opaque. Bertujuan agar tombolnya berwarna transparan, sehingga mempunyai warna background yang sama dengan background JToolBar.
- d) Icon, ganti iconya dengan icon yang telah disiapkan. Untuk memasukkan icon ke dalam JButton, tekan tombol di samping pilihan Icon di dalam Jendela Properties, kemudian akan muncul Dialog Icon Editor seperti di bawah ini :



Jendela icon editor

Pilih radio button Classpath, kemudian tekan tombol Select File dan pilih salah satu icon yang telah disiapkan. Tekan OK. Lakukan langkah-langkah yang sama terhadap JButton yang lain.

5. Compile dan jalankan class ToolbarMenu untuk melihat hasilnya.

Membuat Dialog dan JFileChooser

Dialog memerankan peran yang penting dalam aplikasi berbasis desktop. Interaksi antara user dengan aplikasi terkadang tidak berjalan dengan baik karena user memberikan aksi yang tidak valid kepada aplikasi. Ketika hal tersebut terjadi, aplikasi harus memberitahukan kepada user apa yang telah terjadi dan bagaimana seharusnya user memperbaikinya. Model interaksi seperti ini tepat dilaksanakan menggunakan dialog.

Skenario lain adalah ketika aplikasi memerlukan input dari user agar aplikasi bisa terus melaksanakan tugasnya, misalnya meminta konfirmasi apakah user yakin akan melaksanakan sebuah aksi penting terhadap aplikasi seperti delete, update atau add data.

Dialog juga memberikan pembatasan kepada user, sebelum dialog selesai diproses, user tidak akan bisa berinteraksi dengan bagian aplikasi lainnya. Dialog mencegah hal ini terjadi dengan memastikan bahwa jendela yang bisa diaktifkan hanyalah jendela dialog, sedangkan jendela aplikasi yang lain tidak dapat diaktifkan selama jendela dialog masih aktif.

Aplikasi sangat sering menggunakan dialog untuk berinteraksi dengan user, tetapi jenis interaksinya selalu seragam dan berulang-ulang. Swing menyediakan dialog yang didesign untuk keperluan yang sering muncul dalam aplikasi, seperti JOptionPane dan JFileChooser. Swing juga menyediakan class JDialog jika kita ingin

membuat dialog custom sesuai keinginan kita.

Membuat pre-defined dialog dengan JOptionPane

JOptionPane menyediakan beberapa dialog yang siap pakai dan sering digunakan dalam aplikasi. JOptionPane sangat memudahkan kita dalam meminta user suatu input tertentu atau memberitahu user apa yang terjadi dalam aplikasi.

JOptionPane mempunyai banyak static method untuk menampilkan popup dialog dengan mudah. Terdapat empat method utama yang dapat kita gunakan sebagai landasan membuat dialog. Keempat method tersebut secara rinci digambarkan dalam table berikut ini:

Method	Deskripsi
showConfirmDialog	Meminta konfirmasi dari user, seperti yes/no/cancel
showInputDialog	Meminta input dari user, baik berupa input text menggunakan JTextField maupun pilihan menggunakan JComboBox
showMessageDialog	Memberitahukan user tentang apa yang baru saja terjadi
showOptionDialog	Gabungan dari ketiga jenis dialog diatas

Table method JOptionPane

Swing juga menyediakan method showInternalXXX yang digunakan jika kita bekerja dengan JInternalFrame.

Parameter dari keempat method tersebut mengikuti pola yang konsisten. Terurut dari kiri ke kanan, berikut ini parameter-parameter yang bisa diterima oleh method-method dalam class JOptionPane:

1. parentComponent

Mendefisikan komponen yang akan menjadi parent dari dialog box ini. Frame dari parent component tersebut akan menjadi frame dari dialog dan dialog akan ditampilkan di tengah-tengah parent component. Jika nilai dari parentComponent diset null,

maka dialog akan menggunakan frame default dan dialog akan diletakkan ditengah-tengah layar monitor (tergantung L&F).

2. message

Pesan yang deskriptif menerangkan perihal dialog yang muncul. Pada umumnya message berupa pesan String yang akan diletakkan dalam dialog, namun jenis object lain juga diijinkan digunakan sebagai message. Object-object yang diijinkan akan diperlakukan berbeda, object-object tersebut antara lain

a) Object[]

Setiap object akan ditampilkan dalam dialog berurut dari atas ke bawah. Aturan ini berlaku rekursif untuk semua object didalam array.

b) Component

Jika object yang dimasukkan sebagai message bertipe Component, maka Component tersebut akan ditampilkan ditengah-tengah dialog.

c) Icon

Icon akan dimasukkan ke dalam sebuah JLabel kemudian ditampilkan di sebelah kiri dari dialog.




d) others

Object lainnya akan ditampilkan dalam dialog dengan mengambil nilai kembalian dari method toString dari setiap object.

3. messageType

Mendefisikan jenis dari pesan. Pada umumnya memberikan custom icon untuk setiap jenis pesan. Setiap L&F manager

akan memperlakukan setiap jenis pesan dengan berbeda, namun perbedaannya tidak akan terlalu mencolok. Pilihan yang mungkin dan icon yang mewakilinya adalah:

- a) ERROR_MESSAGE 
- b) INFORMATION_MESSAGE 
- c) WARNING_MESSAGE 
- d) PLAIN_MESSAGE (tanpa icon)

4. optionType

Mendefisikan tombol-tombol yang akan ditampilkan di bagian bawah dari dialog.

- a) DEFAULT_OPTION
- b) YES_NO_OPTION
- c) YES_NO_CANCEL_OPTION
- d) OK_CANCEL_OPTION

Namun kita tidak dibatasi untuk hanya menggunakan empat jenis set tombol diatas, kita dapat mendefisikan tombol-tombol yang akan muncul sesuai kebutuhan.

5. options

Deskripsi yang lebih detail dari set tombol yang digunakan dialog. Nilai yang lazim adalah sebuah array String berisi text yang akan ditampilkan di setiap tombol. Namun Object lain juga dapat diterima, antara lain:

- a) Component

Component akan diletakkan dalam baris tombol secara langsung.

- b) Icon

Sebuah JButton akan dibuat dan didekorasi dengan icon ini.

c) other

Object dengan tipe selainya akan dirubah ke dalam bentuk String dengan mengambil nilai kembalian dari method toString dari object tersebut.

6. icon

Icon yang digunakan untuk mendekorasi dialog. Jika icon ini didefinisikan maka akan menimpa icon default yang didefinisikan oleh messageType.

7. title

Judul dari dialog yang diletakkan di bagian paling atas dari dialog.

8. initialValue

Nilai default dari pilihan yang mungkin ada dalam dialog.

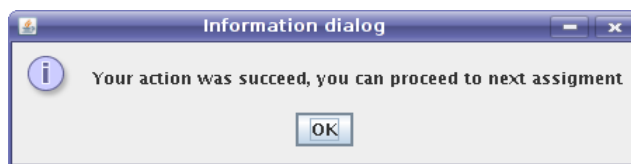
Untuk lebih jelasnya, berikut ini beberapa contoh kode penggunaan JOptionPane beserta hasil tampilanya :

```
JOptionPane.showMessageDialog(null,
    "Simple plain dialog","Plain dialig",
    JOptionPane.PLAIN_MESSAGE);
```



Tampilan dialog sederhana

```
JOptionPane.showMessageDialog(null,  
    "Your action was succeed, " +  
    "you can proceed to next assignment",  
    "Information dialog",  
    JOptionPane.INFORMATION_MESSAGE);
```



Tampilan dialog dengan tipe dialog Information

```
JOptionPane.showMessageDialog(null,  
    "You need to be sure to do this action!",  
    "Dialog Peringatan", JOptionPane.WARNING_MESSAGE);
```



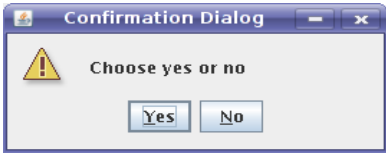
Dialog dengan tipe Warning

```
JOptionPane.showMessageDialog(null,  
    "Something goes wrong and generate error message",  
    "Error Dialog", JOptionPane.ERROR_MESSAGE);
```



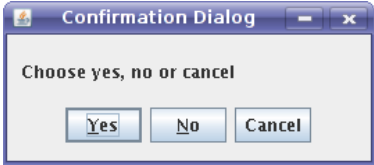
Dialog dengan tipe Error

```
JOptionPane.showMessageDialog(null,
    "Choose yes or no", "Confirmation Dialog",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.WARNING_MESSAGE);
```



Option dialog dengan tipe Information dan pilihan YES_NO

```
JOptionPane.showMessageDialog(null,
    "Choose yes, no or cancel", "Confirmation Dialog",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.PLAIN_MESSAGE);
```



OptionDialog dengan tipe Plain dan pilihan YES_NO_CANCEL

```
JOptionPane.showInputDialog(null,
    "Input your name here", "Input Dialog",
    JOptionPane.INFORMATION_MESSAGE);
```



InputDialog dengan tipe message Information

```
String[] options = {"Apple", "Mango", "Grape", "Guava"};
JOptionPane.showInputDialog(null,
    "Choose this one Option", "Input dialog",
    JOptionPane.WARNING_MESSAGE, null, options, "Apple");
```



InputDialog dialog dengan tipe Warning, Options berupa array of String dan initialValue = 'Apple'

Membuat JFileChooser

JFileChooser digunakan untuk bernavigasi dalam file system, kemudian memilih satu atau lebih file atau folder dari list file dan folder. JFileChooser pada dasarnya adalah pengembangan dari dialog yang dapat digunakan untuk memilih file. JFileChooser dapat digunakan sebagai dialog untuk menyimpan file atau untuk membuka file.

JFileChooser hanya memberikan fasilitas untuk memilih file atau folder, sedangkan mekanisme untuk menyimpan atau membuka file dilakukan sendiri menggunakan library I/O.

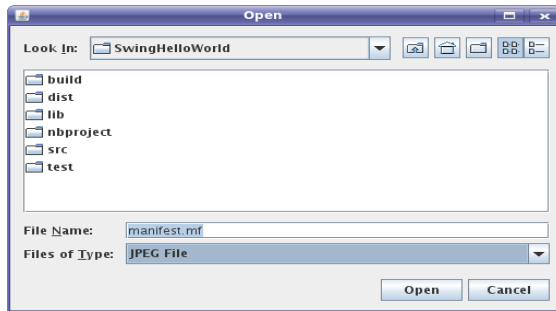
Aplikasi berikut ini adalah contoh penggunaan JFileChooser untuk membuka dan menyimpan file.



Contoh program menggunakan JFileChooser

Tampilan JFileChooser ketika tombol open ditekan adalah seperti di

bawah ini :



Tampilan JFileChooser

Untuk membuat aplikasi diatas lakukan langkah-langkah berikut ini :

1. Buat class JFrame Form baru, beri nama Chooser.java
2. Masukkan dua buah JTextField : txtOpen dan txtSave, dua buah JButton : btnOpen dan btn save, sebuah JLabel : lblStatus. Sesuaikan penataan komponen sesuai dengan gambar diatas.
3. Tambahkan sebuah object JFileChooser sebagai field dari class Chooser, beri nama chooser.

```
public class Chooser{
    JFileChooser chooser = new JFileChooser();
    //kode lain di sini
}
```

4. FileNameExtentionFilter digunakan sebagai file filter dalam JFileChooser. Metode filteringnya adalah mencocokkan ekstensi file dalam file system dengan ekstensi yang ada dalam FileNameExtentionFilter. Contoh kode di bawah ini akan menyebabkan JFileChooser mempunyai pilihan "JPEG File", dan jika pilihan tersebut dipilih, maka file dengan ekstensi "jpg", "jpeg", "JPG" atau "JPEG" saja yang akan ditampilkan oleh

JFileChooser.

```

FileNameExtensionFilter JPEGFilter =
    new FileNameExtensionFilter(
        "JPEG File", "jpg", "jpeg", "JPG", "JPEG");
chooser.addChoosableFileFilter(JPEGFilter);

```

5. Set direktori yang akan dituju oleh JFileChooser. Untuk mengetahui dimana direktori aktif aplikasi, kita bisa menggunakan system property "user.dir". Kode berikut ini akan menyebabkan JFileChooser dibuka pada direktori aktif aplikasi :

```

String dir = System.getProperty("user.dir");
chooser.setCurrentDirectory(new File(dir));

```

6. Menghandle event penekanan tombol btnSave. Ketika tombol btnSave ditekan, chooser akan menampilkan dialog save file, kemudian mengambil nama file yang dipilih dan menampilkannya dalam txtSave, serta menampilkannya dalam lblStatus. Berikut ini kodenya :

```

private void btnSaveActionPerformed(ActionEvent evt) {
    // TODO add your handling code here:
    int ret = chooser.showSaveDialog(this);
    if(ret == JFileChooser.APPROVE_OPTION){
        File f = chooser.getSelectedFile();
        lblStatus.setText("Status : saving file" +
            f.getAbsolutePath());
        txtSave.setText(f.getAbsolutePath());
    }
}

```

7. Menghandle penekanan tombol btnOpen. Kode untuk menangani penekanan tombol btnOpen mirip dengan kode untuk menangani penekanan tombol btnSave, perbedaanya

adalah btnOpen akan menampilkan dialog open file, berikut ini kodenya :

```
private void btnBrowseActionPerformed(ActionEvent evt){
    // TODO add your handling code here:
    int ret = chooser.showOpenDialog(this);
    if(ret == JFileChooser.APPROVE_OPTION){
        File f = chooser.getSelectedFile();
        lblStatus.setText("Status : opening file" +
            f.getAbsolutePath());
        txtOpen.setText(f.getAbsolutePath());
    }
}
```

8. Compile dan jalankan aplikasinya dengan menekan tombol SHIFT + F6

Bekerja dengan JOptionPane dan dengan JFileChooser sangat sederhana. Keduanya menggunakan modal dialog untuk mengambil input dari user. Modal dialog akan mencegah user mengakses bagian aplikasi lain sebelum dialog ditutup, atau dalam hal ini memutuskan pilihan apa yang diambil oleh user.

Masih banyak lagi komponen swing yang disediakan oleh JDK, anda tinggal melanjutkan membaca dari referensi yang diberikan modul ini pada bagian akhir untuk melanjutkan pembelajaran anda tentang Java desktop.

Konsep MVC

MVC adalah arsitektur aplikasi yang memisahkan kode-kode aplikasi dalam tiga lapisan, Model, View dan Control. MVC termasuk dalam arsitektural design pattern yang menghendaki organisasi kode yang terstruktur dan tidak bercampur aduk. Ketika aplikasi sudah sangat besar dan menangani struktur data yang kompleks, harus ada pemisahan yang jelas antara domain model, komponen view dan kontroler yang mengatur penampilan model dalam view.

Arsitektur MVC ini memungkinkan adanya perubahan dalam domain model tanpa harus mengubah code untuk menampilkan domain model tersebut. Hal ini sangat bermanfaat ketika aplikasi mempunyai domain model dan view komponen sangat besar dan kompleks.

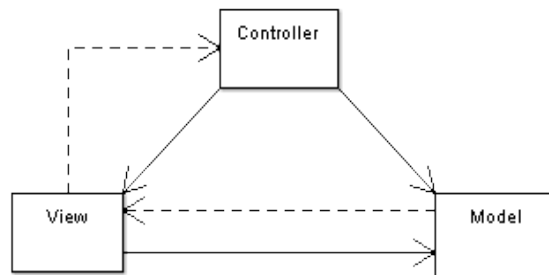


Diagram interaksi antar komponen dalam arsitektur MVC
(Wikipedia.org)

Model adalah representasi dari object yang sedang diolah oleh aplikasi, dalam Java, model ini biasanya direpresesentasikan sebagai Java Bean. Java Bean adalah class Java biasa atau POJO (Plain Old Java Object). Syarat sebuah POJO dianggap sebagai Java Bean adalah :

1. Mempunyai constructor default, constructor yang tidak mempunyai parameter.
2. Semua field-field yang bisa diakses dilengkapi dengan getter dan setter method.

Lebih jelasnya lihat kode dari class Customer di bawah ini :

```
public class Customer {  
    private int id;  
    private String nama, status, agama, jenisKelamin,  
        pekerjaan;  
    public Customer() { }  
    public int getId() { return id; }  
    public void setId(int id) { this.id = id; }  
    public String getName() { return nama; }  
    public void setName(String nama) { this.nama = nama;}  
    //getter dan setter method untuk field lainnya di sini  
}
```

Kode diatas adalah representasi Model dalam Java untuk Entity Customer. Beberapa orang terkadang salah mengartikan model ini sebagai data akses domain. Dimana data dari sumber data, misalnya database, diambil dan diolah. Pada hakekatnya Model adalah representasi data dari object sebenarnya, bukan kumpulan kode untuk mengakses data dari database.

Pendekatan terbaik adalah memisahkan kode untuk melakukan akses sumber data ke dalam lapisan tersendiri, lapisan ini biasanya disebut sebagai service. Service diimplementasikan dalam bentuk class-class yang disebut sebagai manager, misalnya SQLManager, PrintManager, ReportManager, XMLManager, WebServiceManager dan seterusnya. Dengan begitu kode akan menjadi lebih rapi dan terstruktur. Manfaat paling terasa adalah kemudahan pencarian kesalahan dan penambahan modul-modul baru tidak harus

merombak seluruh struktur aplikasi.

View adalah komponen untuk merepresentasikan Model dalam bentuk visual. Semisal komponen swing, seperti : JTable, JList, JComboBox dan sebagainya. View juga bertanggung jawab untuk menangkap interaksi user terhadap sistem, semisal : klik mouse, penekanan tombol keyboard, barcode scanning dan sebagainya.

Controller sebenarnya hanya sekumpulan kode-kode untuk mensinkronisasi keadaan Model dan View. Jika ada perubahan data dari Model, Controller harus mengupdate tampilan View. Dan sebaliknya jika user memberikan event terhadap View, Controller harus mengupdate Model sesuai dengan hasil interaksi user terhadap View.

Model dalam Komponen Swing

Sebagian besar komponen swing mempunyai model. JButton mempunyai model yaitu ButtonModel yang memegang 'state' dari JButton – apa keyboard mnemonicnya, apakah JButton tersebut sedang dipilih atau tidak dan seterusnya. Ada pula komponen swing yang mempunyai lebih dari satu model. JList mempunyai ListModel yang memegang isi dari JList dan ListSelectionModel untuk mencatat item JList yang sedang dipilih.

Pada banyak kasus normal kita tidak perlu pusing memikirkan model ini. Semisal kita tidak perlu memikirkan model dari JButton karena pada kasus umum kita tidak perlu memodifikasi model dari JButton.

Lalu, kenapa model komponen swing dibuat? Alasan utamanya adalah fleksibilitas untuk menentukan bagaimana data disimpan dan diambil dari komponen swing. Misalnya kita mempunyai aplikasi spreadsheet yang menggunakan komponen JTable, karakteristik utama spreadsheet adalah banyak cell yang kosong,

dengan begitu kita bisa memilih model data yang sesuai dengan karakteristik tersebut.

Contoh lainnya adalah JTable yang digunakan untuk menampilkan data dari database dengan jumlah baris luar biasa banyak. Kita bisa mengatur agar tampilan JTable dibuat halaman-per-halaman dalam menampilkan baris data, tidak semua data ditampilkan dalam satu halaman, hal ini ditujukan untuk efisiensi dan mempertahankan agar aplikasi tetap responsif walau bekerja dengan data yang besar.

Model dalam komponen swing juga mempunyai keuntungan lain, yaitu tidak perlu ada dua data terpisah, untuk struktur data aplikasi dan untuk komponen swing.

Kegunaan Model yang cukup penting juga adalah adanya konsep event-listener, dimana jika terjadi event perubahan data dalam model, semua listener yang terdaftar dalam model tersebut akan diberitahu dan tindakan yang tepat dapat diambil untuk menangani event yang muncul. Sebagai contoh, untuk menambahkan item dalam JList kita bisa memanggil method addItem dari JList. Penambahan item dalam JList ini akan mengakibatkan ListModel memicu event dalam JList dan listener lainnya. Komponen swing—dalam hal ini JList—akan diupdate tampilannya untuk merefleksikan perubahan item dalam ListModel.

Walaupun terkadang banyak yang menyebut arsitektur komponen swing sebagai MVC, tetapi pada dasarnya arsitektur komponen swing tidak sepenuhnya MVC. Komponen swing secara umum dibuat agar View dan Controller diletakkan dalam satu tempat (class) yaitu class UI yang disediakan oleh Look-and-Feel. Arsitektur komponen swing lebih tepat disebut sebagai “Arsitektur dengan Model yang terpisah”.

Selanjutnya kita akan membahas beberapa model yang seringkali harus kita kustomisasi sesuai dengan kebutuhan. Sedangkan model

yang nyaris tidak pernah kita rubah—`ButtonModel`—tidak dibahas dalam bagian ini.

TableModel

`TableModel` adalah class model yang paling sering dikustomisasi. Karakteristik data dari `JTable` yang berbentuk koleksi data dua dimensi membutuhkan perhatian khusus agar efisien digunakan dalam aplikasi. Jika kita tidak hati-hati, maka aplikasi kita bisa menjadi sangat lambat dan tidak efisien.

`TableModel` adalah interface yang digunakan oleh `JTable` untuk mendefinisikan ciri-ciri dari data tabular yang akan ditampilkan oleh `JTable`. Misalnya : jumlah kolom, nama kolom, class dari object dalam kolom, jumlah baris dan nilai setiap cell. Dengan adanya data-data ini `JTable` dapat secara efisien menentukan bagaimana menampilkan data tersebut.

Berikut ini adalah kode untuk menampilkan koleksi object `Customer`. Class `ArrayList<Customer>` adalah implementasi dari generics, konsep dalam Java yang digunakan untuk mendefinisikan isi dari koleksi. `ArrayList<Customer>` artinya adalah membuat sebuah object koleksi `ArrayList` yang harus diisi oleh object `Customer` dan tidak bisa diisi oleh object lainya, misalnya `String`.

```
public class CustomerTableModel implements TableModel{
    private ArrayList<Customer> customers =
        new ArrayList<Customer>();
    private Set<TableModelListener> listeners =
        new HashSet<TableModelListener>();
    public CustomerTableModel(List<Customer> cust){
        customers.addAll(cust);
    }
}
```

```
public int getRowCount() { return customers.size();}
public int getColumnCount() { return 6; }
public String getColumnName(int columnIndex) {
    switch(columnIndex){
        case 0 : return "Id";
        case 1 : return "Nama";
        case 2 : return "Status";
        case 3 : return "Agama";
        case 4 : return "Jenis Kelamin";
        case 5 : return "Pekerjaan";
        default : return "";
    }
}
public Class getColumnClass(int columnIndex) {
    switch(columnIndex){
        case 0 : return Integer.class;
        case 1 : return String.class;
        case 2 : return String.class;
        case 3 : return String.class;
        case 4 : return String.class;
        case 5 : return String.class;
        default : return String.class;
    }
}
public boolean isCellEditable(int rowIndex,
    int columnIndex) { return false; }
public Object getValueAt(int rowIndex,
    int columnIndex) {
    Customer currentCustomer =
        customers.get(rowIndex);
```

```

switch(rowIndex){
    case 0 : return currentCustomer.getId();
    case 1 : return currentCustomer.getNama();
    case 2 : return currentCustomer.getStatus();
    case 3 : return currentCustomer.getAgama();
    case 4 : return currentCustomer.getJenisKelamin();
    case 5 : return currentCustomer.getPekerjaan();
    default : return "";
}
}
public void setValueAt(Object aValue,
    int rowIndex, int columnIndex) { }
public void addTableModelListener(
    TableModelListener l) { listeners.add(l); }
public void removeTableModelListener(
    TableModelListener l) { listeners.remove(l); }
}

```

Terlihat cukup repot bukan? untungnya telah tersedia dua class yang mengimplement interface TableModel ini. AbstractTableModel telah mengimplementasi sebagian besar abstract method dari interface TableModel, hanya tinggal tiga method saja yang masih bersifat abstract yaitu :

```

public int getRowCount();
public int getColumnCount();
public Object getValueAt(int row, int column);

```

Yang perlu diperhatikan bahwa dalam AbstractTableModel, method isCellEditable selalu mengembalikan nilai false, artinya semua cell tidak dapat diedit. Kemudian method setValueAt adalah method kosong belaka, artinya jika kita memanggil method ini tidak akan terjadi apa-apa.

Class kedua adalah `DefaultTableModel` yang telah mengimplementasi semua method abstract dari interface `TableModel`. Representasi data `DefaultTableModel` menggunakan dua jenis data tabular, yaitu array dua dimensi, `Object[][]`, dan `Vector` dari `Vector`, `Vector<Vector<Object>>`. Jika kita mempunyai struktur data selain kedua jenis tersebut kita harus melakukan konversi data ke dalam salah satu bentuk struktur data tersebut. Cara yang lebih cerdas adalah mendefinisikan sendiri class yang mengimplement interface `TableModel` seperti class `CustomerTableModel` diatas.

Setelah `TableModel` selesai didefinisikan kita tinggal memanggil method `setTableModel` dari object `JTable`, atau membuat object `JTable` baru menggunakan constructor yang menerima argumen `TableModel`. Contohnya seperti potongan kode di bawah ini :

```
JTable table = new JTable(new DefaultTableModel());
JTable table1 = new JTable();
table1.setTableModel(new DefaultTableModel());
```

ListModel dan ListSelectionModel

`JList` adalah komponen swing yang mempunyai dua model sekaligus, `ListModel` dan `ListSelectionModel`. `ListModel` digunakan untuk mendefinisikan item/element yang dikandung oleh `JList`. Sedangkan `ListSelectionModel` digunakan untuk mendefinisikan bagaimana representasi data jika terjadi proses pemilihan di `JList`.

Seperti halnya `TableModel`, `ListModel` mempunyai dua class yang mengimplement `ListModel`, `AbstractListModel` dan `DefaultListModel`. Kita bisa menggunakan salah satu dari tiga tipe tersebut untuk membuat object `ListModel`. Cara pertama dengan membuat class baru yang mengimplement `ListModel`. Cara kedua dengan membuat class baru yang menextends `AbstractListModel`

dan cara ketiga dengan langsung menggunakan DefaultListModel.

Struktur data JList tidak terlalu rumit seperti JTable, dan pada umumnya, cukup hanya dengan menggunakan DefaultListModel sudah memenuhi sebagian besar kebutuhan penggunaan JList.

Berikut ini contoh bagaimana membuat ListModel untuk data customer, contoh ini menggunakan cara kedua untuk membuat object ListModel, yaitu dengan cara membuat class baru yang mengextends AbstractListModel :

```
public class CustomerListModel extends AbstractListModel{
    private ArrayList<Customer> customer =
        new ArrayList<Customer>();
    public CustomerListModel(List<Customer> cust){
        customers.addAll(cust);
    }
    public Object getValueAt(int index) {
        return customers.get(index);
    }
    public int getSize() { return customers.size(); }
}
```

Implementasi ListModel sangat mudah dan tidak serumit TableModel, namun implementasi dari ListSelectionModel sangat rumit, karena kita harus mengimplementasi dua puluh buah method. Lebih baik menggunakan implementasi standard dari ListSelectionModel yaitu DefaultListSelectionModel.

Menangani Event

Event dan Listener adalah implementasi dari pattern Observer dalam Java. Pattern Observer sangat berguna digunakan untuk mendesign komunikasi yang konsisten antara object yang berdiri sendiri dan object-object yang bergantung padanya.

Observer design pattern melibatkan dua object utama, object pertama berlaku sebagai Subject dan object lainnya berlaku sebagai Observer. Object Subject merupakan pusat perhatian dari object Observer, perubahan keadaan dari object Subject selalu dipantau oleh Observer.

Observer dapat melakukan register-unregister terhadap Subject. Jika Observer tertarik dengan perilaku dan keadaan dari Subject, Observer dapat meregister dirinya kepada Subject. Begitu juga sebaliknya jika Observer tidak tertarik terhadap keadaan atau perilaku Subject, Observer tidak perlu melakukan resgistrasi atau kalau sudah terlanjur reguister dapat melakukan unregister.

Subject mempunyai banyak aspek perilaku dan keadaan yang dapat dipantau oleh Observer. Untuk setiap aspek, Subject menyediakan method untuk register-unregister dan menyediakan interface yang harus diimplement oleh Observer yang ingin memantau aspek tersebut.

Pada satu titik tertentu, Subject akan memberitahu (notify) Observer tentang perilaku atau keadaanya. Subject akan mengumpulkan informasi tentang keadaan atau perilakunya kemudian mengirimkan pesan kepada Observer lewat interface yang telah disepakati keduanya, pola ini dikenal juga sebagai Event-Passing.

Pattern Observer dimaksudkan untuk mengurangi ketergantungan satu object terhadap object lain, istilah kerennya adalah Decoupling. Dengan mekanisme register-unregister, Observer dapat secara lebih leluasa memutuskan untuk memantau Subject tertentu atau tidak. Mekanisme notify memudahkan Subject memberitahu keadaan dan perilakunya kepada Observer yang sedang memantaunya.

Di bagian berikutnya kita akan melihat bagaimana pattern Observer diimplementasikan dalam swing. Akan dijelaskan pula bagaimana swing mengimplementasikan mekanisme register-unregister dan notify dalam menangani interaksi user terhadap komponen swing.

Event Listener dalam Swing

Pattern Observer melibatkan dua object Subject dan Observer, dalam swing Observer dikenal sebagai Listener. Kemudian, ketika Subject akan memberitahu (notify) Observer tentang apa yang sedang terjadi dalam object Subject, ada satu informasi yang akan di-passing oleh Subject ke Observer, informasi ini disebut sebagai Event object. Sedangkan kejadian ketika Subject melakukan notify kepada Observer disebut sebagai Event triggering.

Agar penjelasan diatas mudah dipahami, kita akan membuat aplikasi sederhana yang mengimplementasikan pattern Observer. Aplikasi sederhana ini terdiri dari dua class utama yaitu Subject dan Observer.

Class Subject akan menjalankan sebuah loop tanpa batas, di dalam loop tersebut Subject akan meminta input dari user berupa sebuah kata yang diakhiri dengan penekanan enter. Ketika user menekan enter, Subject akan notify Observer. Dalam proses notifikasi tersebut, Subject mengumpulkan informasi tentang event

pemasukan kata oleh user, informasi tersebut berupa : kata apa yang dimasukkan dan object subject dimana event pemasukan kata tersebut terjadi (source). Kemudian Observer akan menerima informasi dari Subject dan mencetak informasi tersebut ke standard output. Berikut ini tampilan dari aplikasi sederhana ini :

```
type a word : ifnu
print from observer : first observer
    event from : subject observed
    key presed is ifnu
```

Subject akan mencetak string “type a word :” dan menunggu user untuk memasukkan satu kata dan menekan enter. Misalnya dalam contoh diatas “ifnu”. Kemudian Subject akan menghimpun informasi tentang sumber event (Subject itu sendiri) dan kata yang diketikkan user (ifnu). Setelah itu, Subject memberitahu (notify) Observer bahwa telah terjadi event pemasukan kata dengan menyertakan informasi yang telah dihimpun Subject.

Observer menerima informasi dari Subject bahwa telah terjadi event pemasukan kata oleh user, selanjutnya Observer akan menjalankan tindakan-tindakan untuk menangani event tersebut. Tindakan tersebut adalah : mencetak informasi Observer, source, dan kata yang dimasukkan oleh user.

Di dalam class Subject terdapat field

1. Koleksi Observer (listeners)
2. Nama (name)
3. Kata yang dimasukkan user (wordEntered)

Kemudian ada juga method :

1. Constructor yang menerima parameter String, parameter ini digunakan sebagai pengenalan (name) dari object Subject.

2. Register-unregister Observer (registerListener, removeListener)
3. Method private untuk menotify Observer (triggerListener)
4. Method untuk menerima input kata dari user (runProgram)

Berikut ini kode lengkapnya :

```
public class Subject {
    private Set<KeyboardPressedListener> listeners =
        new HashSet<KeyboardPressedListener>();
    private String wordEntered;
    private String name;
    public Subject(String subjectName){
        name = subjectName;
    }
    public void runProgram(){
        while(true){
            Console c = System.console();
            wordEntered = c.readLine("type a word : ");
            if(wordEntered.equals("exit"))
                break;
            else
                triggerListener();
        }
    }
    private void triggerListener(){
        KeyboardPressedEvent event =
            new KeyboardPressedEvent();
        event.setSource(this);
        event.setWord(wordEntered);
        for(KeyboardPressedListener l : listeners){
            l.keyPressed(event);
        }
    }
}
```

```

public void registerObserver(
    KeyboardPressedListener l){ listeners.add(l); }
public void removeObserver(
    KeyboardPressedListener l){
    listeners.remove(l);
}
public String toString(){ return name; }
}

```

Interface `KeyboardPressedListener` digunakan sebagai “kontrak” antara `Subject` dan `Observer`. Interface ini menjamin bahwa `Observer` yang akan memantau event pemasukan kata dari user dalam `Subject` mempunyai method `keyPressed`. Method `keyPressed` ini nanti yang akan dipanggil oleh `Subject` ketika event pemasukan kata oleh user terjadi di `Subject`.

```

public interface KeyboardPressedListener {
    public void keyPressed(KeyboardPressedEvent e);
}

```

Class `Observer` mengimplement interface `KeyboardPressedListener` dan nantinya akan didaftarkan ke `subject` sebagai `Observer`. Method `keyPressed` diimplementasikan dengan mencetak informasi yang diperoleh dari `Subject` ke standard output.

```

public class Observer implements KeyboardPressedListener{
    private String name;
    public Observer(String name){ this.name = name; }
    public void keyPressed(KeyboardPressedEvent e) {
        System.out.println("print from observer : " +
            name + "\n\tevent from : " + e.getSource()
            + "\n\tkey presed is " + e.getWord() );}}

```

Class `KeyboardPressedEvent` adalah Java Bean biasa yang menyimpan informasi kejadian pemasukan kata oleh user,

didalamnya hanya ada field source dan word serta getter-setter method untuk kedua field tersebut.

```
public class KeyboardPressedEvent {
    private Object source;
    private String word;
    public Object getSource() { return source; }
    public void setSource(Object src) { source = src;}
    public String getWord() { return word; }
    public void setWord(String wrd) { word = wrd; }
}
```

Sampai disini, class-class diatas masih berdiri sendiri dan belum ada class yang mempunyai method main. Nah, disinilah kode class MainClass untuk menyatukan semua object diatas menjadi aplikasi utuh.

```
public class MainClass {
    public static void main(String[] str){
        Subject subject =
            new Subject("subject observed");
        Observer observer =
            new Observer("first observer");
        subject.registerObserver(observer);
        subject.runProgram();
    }
}
```

Langkah-langkah dalam menggunakan pattern Observer ini adalah :

1. Membuat object subject dari class Subject

```
Subject subject = new Subject("subject observed");
```

2. Membuat object observer dari class Observer

```
Observer observer = new Observer("first observer");
```

3. Daftarkan object observer ke object subject

```
subject.registerObserver(observer);
```

4. Jalankan program utamanya

```
subject.runProgram();
```

Pattern Observer ini digunakan secara intensif dalam komponen swing. Terutama untuk menanggapi event dari input peripheral—keyboard, mouse, barcode reader—yang terjadi di komponen swing—JTextField, JButton, JTable—. Dalam bagian-bagian selanjutnya kita akan belajar bagaimana menangani event pada komponen swing.

ActionListener

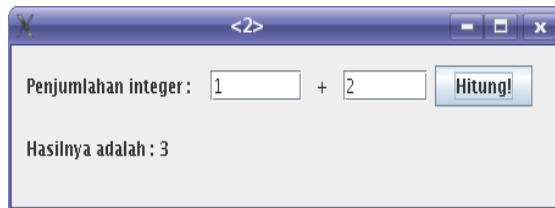
ActionListener digunakan untuk mendengarkan action dari event :

- Klik terhadap JButton
- Pemilihan menu item
- Penekanan tombol enter dalam JTextField

Method dalam ActionListener hanya satu yaitu actionPerformed yang menerima argumen object ActionEvent. ActionEvent berisi informasi-informasi penting ketika Action event terjadi, termasuk tombol modifiers apa yang sedang ditekan. Tombol modifiers antara lain : CTRL, ALT, META dan SHIFT. Method untuk menentukan tombol modifiers apa yang sedang aktif adalah getModifiers. Method getActionCommand digunakan untuk mengambil command string yang didefinisikan oleh JButton.

Di bagian sebelumnya kita telah bekerja menggunakan komponen swing, dan sudah berlatih bagaimana menangani event klik mouse

terhadap JButton. Mari kita lihat lagi aplikasi sederhana berikut :



Contoh aplikasi sederhana yang menangani event `actionEvent` pada JButton

Menangani klik mouse pada JButton dalam Netbeans cukup dengan memilih JButton di Jendela Design kemudian klik kanan dan pilih menu :

```
Events > Action > ActionPerformed
```

Setelah itu anda akan dibawa ke jendela kode dan baris berikut ini akan dibuat secara otomatis oleh Netbeans :

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

Kemudian kita akan menempatkan kode untuk menangani penekanan tombol di bagian bawah baris `//TODO`. Sebenarnya Netbeans men-generate beberapa lagi kode di bagian yang tidak dapat diedit, berikut ini cuplikannya :

```
btnHitung.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent evt){  
        btnHitungActionPerformed(evt);  
    }  
});
```

Method `addActionListerner` ini mempunyai fungsi yang sama

dengan method `registerListener` pada class `Subject` yang kita bahas di bagian sebelumnya. `addActionListerner` berfungsi untuk mendaftarkan `Observer` ke `Subject`. Perbedaan terbesar disini adalah Netbeans tidak membuat public class `Observer` baru untuk mengimplementasi interface `ActionListener`. Tetapi Netbeans membuat anonymous innerclass yang mengimplement interface `ActionListener`.

Perhatikan petikan kode berikut ini :

```
new ActionListener() {  
    public void actionPerformed(ActionEvent evt){  
        btnHitungActionPerformed(evt);  
    }  
}
```

Yang dilakukan oleh kode diatas sebenarnya adalah:

1. Membuat Class baru yang tidak punya nama (anonymous)
2. Class baru tersebut turunan dari `Object` dan mengimplement interface `ActionListener`
3. Mengimplementasi method abstrac `actionPerformed`
4. Method `actionPerformed` dari class tak bernama ini akan memanggil method `btnHitungActionPerformed` dari class parentnya.

Secara umum Netbeans akan membuat sebuah anonymous innerclass seperti diatas untuk setiap penanganan event. Dari sisi kerapian kode metode ini sangat bagus, karena Netbeans menyembunyikan kerumitan kode untuk menangani event seperti diatas. Kita tidak perlu susah-susah membuat sebuah public class `Observer` tersendiri untuk menangani event. Cukup dengan anonymous innerclass.

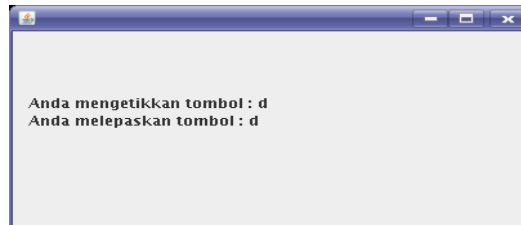
KeyListener

KeyListener akan mendengarkan penekanan tombol oleh komponen yang berada dalam keadaan fokus. Semua komponen swing dapat menerima KeyListener sebagai Observer. KeyListener dapat mendengarkan tiga event berbeda : penekanan tombol, pelepasan tombol dan pengetikan tombol. Ketiganya ditangani oleh method yang berbeda-beda, yaitu :

- `keyPressed` : dipanggil ketika terjadi penekanan tombol keyboard.
- `keyReleased` : dipanggil ketika tombol keyboard dilepaskan.
- `keyTyped` : dipanggil ketika tombol diketikkan, alias ditekan kemudian dilepaskan. Method ini dipanggil jika tombol yang ditekan mempunyai representasi karakter unicode, seperti tombol angka dan tombol huruf. Sedangkan penekanan tombol modifiers seperti ALT, CTRL, ARROW, CAPSLOCK, NUMLOCK, INS dan lainnya tidak akan mengakibatkan method ini dipanggil.

Ketiga method diatas menerima parameter `KeyEvent`. Untuk mengetes tombol apakah yang ditekan oleh user, digunakan method `getKeyCode`. Kemudian hasil kembalian method `getKeyCode` dibandingkan dengan field static kepunyaan class `KeyEvent` yang diawali dengan huruf VK, seperti : `VK_ENTER`, `VK_A`, `VK_B`, `VK_1`, `VK_LEFT_ARROW` dan seterusnya.

Method `getKeyChar` digunakan untuk menentukan karakter apa yang diwakili oleh tombol yang ditekan. Jika tombol yang ditekan adalah tombol modifiers maka method `getKeyChar` akan mengembalikan karakter `KeyEvent.CHAR_UNDEFINED`.



Aplikasi sederhana yang menangani penekanan tombol keyboard

Untuk membuat aplikasi yang mendengarkan penekanan tombol keyboard seperti diatas lakukan langkah-langkah berikut ini :

1. Buat class JFrame baru, beri nama FrameKeyPressed.
2. Tambahkan dua buah JLabel, beri nama lblStatus dan lblKeyTyped.
3. lblStatus digunakan untuk menandakan adanya event keyPressed dan keyReleased yang terjadi berurutan
4. lblKeyTyped digunakan untuk menandakan adanya tombol yang diketik.
5. Pilih JFrame di Jendela Design, klik kanan, dan pilih menu :

Events > Key > keyPressed

6. Jendela Code akan terbuka, modifikasi method formKeyPressed menjadi seperti berikut ini :

```
private void formKeyPressed(KeyEvent evt) {  
    // TODO add your handling code here:  
    if(evt.getKeyChar() == KeyEvent.CHAR_UNDEFINED)  
        lblStatus.setText(  
            "Anda menekan tombol : CHAR_UNDEFINED");  
    else  
        lblStatus.setText("Anda menekan tombol : " +
```

```

        evt.getKeyChar();
    }

```

7. Pilih JFrame di Jendela Design, klik kanan, dan pilih menu :

Events > Key > keyReleased

8. Jendela Code akan terbuka, modifikasi method formKeyReleased menjadi seperti berikut ini :

```

private void formKeyReleased(KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyChar() == KeyEvent.CHAR_UNDEFINED)
        lblStatus.setText(
            "Anda melepaskan tombol : CHAR_UNDEFINED");
    else
        lblStatus.setText("Anda melepaskan tombol : " +
            evt.getKeyChar());
}

```

9. Pilih JFrame di Jendela Design, klik kanan, dan pilih menu :

Events > Key > keyTyped

10. Jendela Code akan terbuka, modifikasi method formKeyTyped menjadi seperti berikut ini :

```

private void formKeyTyped(KeyEvent evt) {
    // TODO add your handling code here:
    lblKeyType.setText("Anda mengetikkan tombol : " +
        evt.getKeyChar());
}

```

MouseListener dan MouseMotionListener

MouseListener mendengarkan interaksi mouse terhadap komponen swing. MouseListener dapat didaftarkan pada semua komponen

swing. `MouseListener` mendengarkan event :

- `mousePressed` : event ini terjadi ketika user menekan salah satu tombol mouse diatas komponen swing.
- `mouseReleased` : setelah tombol ditekan, komponen swing akan menerima pelepasan tombol mouse. Tetapi jika tombol mouse dilepaskan pada saat pointer mouse tidak berada diatas komponen swing, maka event ini tidak akan terjadi.
- `mouseClicked` : event ini muncul ketika user melakukan click tombol mouse diatas komponen swing.
- `mouseEntered` : ketika mouse memasuki area diatas komponen swing, event ini akan dipicu.
- `mouseExited` : muncul ketika mouse akan meninggalkan area diatas komponen swing

Ketika user menekan tombol mouse (click), event `mousePressed` dibuat, kemudian `mouseReleased` dan akhirnya `mouseClicked` muncul terakhir.

`MouseMotionListener` juga dapat didaftarkan sebagai listener pada semua komponen swing. `MouseMotionListener` dipisahkan dari `MouseListener` karena penanganan event `mouseMove` yang lebih berat dan intensif. `MouseMotionListener` mendengarkan dua event:

- `mouseMoved` : terjadi ketika user menggerakkan mouse diatas komponen swing
- `mouseDragged` : terjadi ketika user menekan tombol mouse sekaligus menggerakannya diatas komponen swing

Semua method diatas menerima argumen berupa class `MouseEvent`. Method `getClickCount` digunakan untuk menentukan

jumlah click yang terjadi dalam waktu yang berdekatan. Method `getClickCount` juga digunakan untuk menentukan apakah klik yang terjadi adalah single klik atau double klik.

Method `getButton` digunakan untuk menentukan tombol mana yang ditekan oleh user. Pada umumnya mouse yang tersedia di pasaran mempunyai tiga tombol yang dapat di klik, tombol kiri, tombol tengah dan tombol kanan. Method `getButton` akan mengembalikan nilai `MouseEvent.BUTTON1` jika tombol kiri ditekan, `MouseEvent.BUTTON2` jika tombol tengah ditekan dan `MouseEvent.BUTTON3` jika tombol kanan ditekan.

Method `getX` dan `getY` akan mengembalikan koordinat dimana `MouseEvent` terjadi. Koordinat yang digunakan adalah koordinat relatif. Koordinat (0,0) berada di pojok kiri atas dari komponen swing, semakin kebawah nilai Y semakin besar dan semakin ke kanan nilai X semakin besar. Nilai koordinatnya dinyatakan dalam satuan pixel.

Aplikasi di bawah ini adalah sebuah `JFrame` yang mempunyai `JLabel` di dalamnya. Ketika terjadi event `mouseClick` dan `mouseMove`, `JLabel` akan menampilkan dimana event tersebut terjadi. Jika event klik yang muncul, maka text dari `JLabel` akan berisi "clicked at (x,y)", sedangkan event move hanya akan memunculkan koordinat "(x,y)" saja.



Contoh aplikasi sederhana yang mengani `MouseEvent`

Lakukan langkah-langkah berikut ini untuk membuat aplikasi seperti diatas :

1. Buat class `JFrame` baru, beri nama `FrameMouseMove`.

2. Letakkan sebuah JLabel baru, beri nama lblStatus.
3. Pilih JFrame di Jendela Designer, klik kanan dan pilih menu :

Set Layout > Null Layout

Langkah ini bertujuan untuk membuat agar JFrame menggunakan null layout. Kalau tidak menggunakan null layout kita tidak bisa meletakkan JLabel pada sembarang posisi.

4. Pilih kembali JFrame, klik kanan dan pilih menu

Events > Mouse > mouseClicked

5. modifikasi kode pada Jendela Code yang tampil menjadi seperti di bawah ini :

```
private void formMouseClicked(MouseEvent evt) {  
    // TODO add your handling code here:  
    lblStatus.setText("clicked at (" + evt.getX() +  
        "," + evt.getY() + ")");  
    lblStatus.setLocation(evt.getX(),evt.getY());  
}
```

Kode diatas menangani penekanan tombol mouse, kemudian mengubah text JLabel dan memindahkan JLabel ke posisi dimana event mouseClicked terjadi.

6. Pilih JFrame lagi di Jendela Design, klik kanan dan pilih menu :

Events > MouseMotion > mouseMoved

7. Modifikasi kode yang muncul pada Jendela Code menjadi seperti di bawah ini :

```
private void formMouseMoved(MouseEvent evt) {  
    // TODO add your handling code here:  
    lblStatus.setText("(" + evt.getX() + "," +  
        evt.getY() + ")");  
}
```

```
lblStatus.setLocation(evt.getX(), evt.getY());  
}
```

8. Compile dan jalankan aplikasi di atas.

Masih banyak lagi Event-Listener yang disediakan oleh JDK. Dari bab di atas kita sudah dapat mengerti dengan baik konsep Event-Listener dan pattern Observer yang mendasarinya. Dengan kemampuan ini kita bisa dengan mudah mengerti bagaimana event-listener yang lain bekerja.

Event-Listener juga dapat dijalankan terhadap Java Bean menggunakan PropertyChangeListener dan PropertyEvent. Konsep ini dapat digunakan untuk mengamati perubahan pada field Java Bean.

Koneksi Database Dengan JDBC

Mengenal JDBC

Java Database Connectivity adalah API yang digunakan Java untuk melakukan koneksi dengan aplikasi lain atau dengan berbagai macam database. JDBC memungkinkan kita untuk membuat aplikasi Java yang melakukan tiga hal: konek ke sumber data, mengirimkan query dan statement ke database, menerima dan mengolah resultset yang diperoleh dari database.

JDBC mempunyai empat komponen :

1. JDBC API

JDBC API menyediakan metode akses yang sederhana ke sumber data relational (RDBMS) menggunakan pemrograman Java. dengan menggunakan JDBC API, kita bisa membuat program yang dapat mengeksekusi SQL, menerima hasil ResultSet, dan mengubah data dalam database. JDBC API juga mempunyai kemampuan untuk berinteraksi dengan lingkungan terdistribusi dari jenis sumber data yang berbeda-beda.

JDBC API adalah bagian dari Java Platform yang disertakan dalam library JDK maupun JRE. JDBC API sekarang ini sudah mencapai versi 4.0 yang disertakan dalam JDK 6.0. JDBC API 4.0 dibagi dalam dua package yaitu : `java.sql` dan `javax.sql`.

2. JDBC Driver Manager

Class DriverManager dari JDBC bertugas untuk mendefisikan object-object yang dapat digunakan untuk melakukan koneksi ke sebuah sumber data. Secara tradisional DriverManager

telah menjadi tulang punggung arsitektur JDBC.

3. JDBC Test Suite

JDBC Test Suite membantu kita untuk mencari driver mana yang cocok digunakan untuk melakukan sebuah koneksi ke sumber data tertentu. Tes yang dilakukan tidak memerlukan resource besar ataupun tes yang komprehensif, namun cukup tes-tes sederhana yang memastikan fitur-fitur penting JDBC dapat berjalan dengan lancar.

4. JDBC-ODBC Bridge

Bridge ini menyediakan fasilitas JDBC untuk melakukan koneksi ke sumber data menggunakan ODBC (Open DataBase Connectivity) driver. Sebagai catatan, anda perlu meload driver ODBC di setiap komputer client untuk dapat menggunakan bridge ini. Sebagai konsekuensinya, cara ini hanya cocok dilakukan di lingkungan intranet dimana isu instalasi tidak menjadi masalah.

Dengan keempat komponen yang dipunyainya, JDBC menjadi tools yang dapat diandalkan untuk melakukan koneksi, mengambil data dan merubah data dari berbagai macam sumber data. Modul ini hanya akan membahas dua komponen pertama dari keempat komponen yang dipunyai oleh JDBC, yaitu JDBC API dan DriverManager. Sumber data yang digunakan adalah Relational Database.

Database Driver

JDBC memerlukan database driver untuk melakukan koneksi ke suatu sumber data. Database driver ini bersifat spesifik untuk setiap jenis sumber data. Database driver biasanya dibuat oleh pihak pembuat sumber datanya, namun tidak jarang juga

komunitas atau pihak ketiga menyediakan database driver untuk sebuah sumber data tertentu.

Perlu dipahami sekali lagi bahwa database driver bersifat spesifik untuk setiap jenis sumber data. Misalnya, Database Driver MySql hanya bisa digunakan untuk melakukan koneksi ke database MySql dan begitu juga database driver untuk Postgre SQL juga hanya bisa digunakan untuk melakukan koneksi ke database Postgre SQL.

Database driver untuk setiap DBMS pada umumnya dapat didownload dari website pembuat DBMS tersebut. Beberapa vendor DBMS menyebut Database driver ini dengan sebutan Java Connector (J/Connector). Database driver biasanya dibungkus dalam file yang berekstensi jar. Setiap database driver harus mengimplement interface `java.sql.Driver`.

Membuat Koneksi

Melakukan koneksi ke database melibatkan dua langkah: Memload driver dan membuat koneksi itu sendiri. Cara memload driver sangat mudah, pertama letakkan file jar database driver ke dalam classpath. Kemudian load driver dengan menambahkan kode berikut ini:

```
Class.forName("com.mysql.jdbc.Driver");
```

Nama class database driver untuk setiap DBMS berbeda, anda bisa menemukan nama class tersebut dalam dokumentasi driver database yang anda gunakan. Dalam contoh ini, nama class database driver dari MySql adalah `com.mysql.jdbc.Driver`.

Memanggil method `Class.forName` secara otomatis membuat instance dari database driver, class `DriverManager` secara otomatis juga dipanggil untuk mengelola class database driver ini. Jadi anda tidak perlu menggunakan statement `new` untuk membuat instance

dari class database driver tersebut.

Langkah berikutnya adalah membuat koneksi ke database menggunakan database driver yang sudah di-load tadi. Class DriverManager bekerja sama dengan interface Driver untuk mengelola driver-driver yang di-load oleh aplikasi, jadi dalam satu sesi anda bisa meload beberapa database driver yang berbeda.

Ketika kita benar-benar melakukan koneksi, JDBC Test Suite akan melakukan serangkaian tes untuk menentukan driver mana yang akan digunakan. Parameter yang digunakan untuk menentukan driver yang sesuai adalah URL. Aplikasi yang akan melakukan koneksi ke database menyediakan URL pengenalan dari server database tersebut. Sebagai contoh adalah URL yang digunakan untuk melakukan koneksi ke MySQL :

```
jdbc:mysql://[host]:[port]/[schema]
```

contoh konkritnya :

```
jdbc:mysql://localhost:3306/latihan
```

Setiap vendor DBMS akan menyertakan cara untuk menentukan URL ini di dalam dokumentasi. Anda tinggal membaca dokumentasi tersebut tanpa harus khawatir tidak menemukan informasi yang anda perlukan.

Method DriverManager.getConnection bertugas untuk membuat koneksi:

```
Connection conn =  
    DriverManager.getConnection(  
        "jdbc:mysql://localhost:3306/latihan");
```

Dalam kebanyakan kasus anda juga harus memasukkan parameter username dan password untuk dapat melakukan koneksi ke dalam database. Method getConnection menerima Username sebagai parameter kedua dan password sebagai parameter ketiga, sehingga

kode diatas dapat dirubah menjadi :

```
Connection conn =  
    DriverManager.getConnection(  
        "jdbc:mysql://localhost:3306/latihan",  
        "root","");
```

Jika salah satu dari driver yang diload berhasil digunakan untuk melakukan koneksi dengan URL tersebut, maka koneksi ke database berhasil dilaksanakan. Class Connection akan memegang informasi koneksi ke database yang didefinisikan oleh URL tersebut.

Setelah sukses melakukan koneksi ke database, kita dapat mengambil data dari database menggunakan perintah query ataupun melakukan perubahan terhadap database. bagian berikut ini akan menerangkan bagaimana cara mengambil dan memanipulasi data dari database.

Mengambil dan Memanipulasi Data dari Database

Proses pengambilan data dari database memerlukan suatu class untuk menampung data yang berhasil diambil, class tersebut harus mengimplement interface ResultSet.

Object yang bertipe ResultSet dapat mempunyai level fungsionalitas yang berbeda, hal ini tergantung dari tipe dari result set. Level fungsionalitas dari setiap tipe result set dibedakan berdasarkan dua area:

- Dengan cara bagaimana result set itu dapat dimanipulasi
- Bagaimana result set itu menangani perubahan data yang dilakukan oleh proses lain secara bersamaan (concurrent).

JDBC menyediakan tiga tipe result set untuk tujuan berbeda:

1. `TYPE_FORWARD_ONLY` : result set tersebut tidak bisa berjalan mundur, result set hanya bisa berjalan maju dari baris pertama hingga baris terakhir. result set hanya menggambarkan keadaan data ketika query dijalankan atau ketika data diterima oleh result set. Jika setelah itu ada perubahan data dalam database, result set tidak akan diupdate alias tidak ada perubahan dalam result set tipe ini.
2. `TYPE_SCROLL_INSENSITIVE` : result set dapat berjalan maju mundur. result set dapat berjalan maju dari row pertama hingga terakhir atau bergerak bebas berdasarkan posisi relatif atau absolute.
3. `TYPE_SCROLL_SENSITIVE` : result set dapat berjalan maju mundur. result set dapat berjalan maju dari row pertama hingga terakhir atau bergerak bebas berdasarkan posisi relatif atau absolute.

Instance dari object bertipe `ResultSet` diperlukan untuk menampung hasil kembalian data dari database. Sebelum kita bisa memperoleh instance dari `ResultSet`, kita harus membuat instance dari class `Statement`. Class `Statement` mempunyai method `executeQuery` yang digunakan untuk menjalankan perintah query dalam database kemudian mengembalikan data hasil eksekusi query ke dalam object `ResultSet`.

Berikut ini adalah contoh kode untuk membuat instance class `Statement`, kemudian menjalankan query untuk mengambil data dari database yang hasilnya dipegang oleh `ResultSet` :

```
Statement statement =
    conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
ResultSet rs =
    statement.executeQuery("select * from Customers");
```

ResultSet akan meletakkan kursornya (posisi pembacaan baris) di sebuah posisi sebelum baris pertama. Untuk menggerakkan kursor maju, mundur, ke suatu posisi relatif atau ke suatu posisi absolute tertentu, gunakan method-method dari ResultSet:

- next() -- mengarahkan kursor maju satu baris.
- previous() -- mengarahkan kursor mundur satu baris.
- first() -- mengarahkan kursor ke baris pertama.
- last() -- mengarahkan kursor ke baris terakhir.
- beforeFirst() -- mengarahkan kursor ke sebelum baris pertama.
- afterLast() -- mengarahkan kursor ke setelah baris terakhir.
- relative(int rows) -- mengarahkan kursor relatif dari posisinya yang sekarang. Set nilai rows dengan nilai positif untuk maju, dan nilai negatif untuk mundur.
- absolute(int rowNumber) -- mengarahkan kursor ke posisi tertentu sesuai dengan nilai rowNumber, dan tentu saja nilainya harus positif.

Interface ResultSet menyediakan method getter untuk mengakses nilai dari setiap kolom dalam baris yang sedang aktif. Parameter fungsi getter bisa menerima nilai index dari kolom ataupun nama kolomnya. Namun begitu, penggunaan nilai index lebih efisien dibanding menggunakan nama kolom.

Nilai index dimulai dengan satu hingga banyaknya kolom. Penggunaan nama kolom adalah case insensitive, artinya huruf kecil atau huruf besar tidak menjadi masalah.

getString digunakan untuk mengambil kolom dengan tipe data char, varchar atau tipe data string lainnya. getInt digunakan untuk mengambil kolom dengan tipe data integer.

Berikut ini adalah contoh program lengkap dari melakukan koneksi hingga mengambil data dari database.

```

Class.forName("com.mysql.jdbc.Driver");
Connection conn =
    DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/latihan",
        "root","");
Statement statement =
    conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
ResultSet rs =
    statement.executeQuery("select * from Customers");
while(rs.next()){
    System.out.println(rs.getInt("id"));
    System.out.println(rs.getString("Nama"));
}

```

Method executeQuery hanya dapat menjalankan perintah SQL select, gunakan method executeUpdate untuk menjalankan perintah insert, update dan delete. Hasil dari eksekusi insert, update dan delete tidak mengembalikan result set, tetapi mengembalikan sebuah nilai integer yang merepresentasikan status hasil eksekusi method executeUpdate.

Berikut ini contoh insert, update dan delete :

```
result = statement.executeUpdate(  
    "update Customers set nama = 'robby' where  
    nama='andy'");  
result = statement.executeUpdate(  
    "delete Customers where nama='andy'");
```

Menggunakan `executeQuery` dan `executeUpdate` sangat mudah dan fleksible, namun sangat tidak efisien, `PreparedStatement` menawarkan keunggulan dalam bentuk efisiensi.

Menggunakan PreparedStatement

Memanggil method `executeUpdate` berulang-ulang, misalnya melakukan insert ratusan atau ribuan baris, sangat tidak efisien. Hal ini disebabkan karena DBMS harus memproses setiap query yang dikirimkan dalam beberapa langkah: memarsing query, mengcompile query dan kemudian baru mengeksekusi query tersebut.

`PreparedStatement` menawarkan solusi yang lebih baik dalam menangani keadaan tersebut. `PreparedStatement` menyaratkan query yang akan dieksekusi didefinisikan terlebih dahulu ketika `PreparedStatement` dibuat. Kemudian query tersebut dikirimkan ke dalam database untuk dicompile terlebih dahulu sebelum digunakan. Konsekuensinya, `PreparedStatement` bukan hanya mempunyai query, tetapi mempunyai query yang sudah dicompile. Ketika `PreparedStatement` dijalankan, DBMS tidak perlu melakukan kompilasi ulang terhadap query yang dijalankan `PreparedStatement`. Hal inilah yang menyebabkan `PreparedStatement` jauh lebih efisien dibandingkan menggunakan method `Statement.executeUpdate`.

Berikut ini contoh pembuatan `PreparedStatement` menggunakan class `Connection` yang telah dibuat sebelumnya :

```
PreparedStatement ps = conn.prepareStatement(
    "update Customers set nama = ? where nama = ?");
```

Perhatikan tanda ? yang ada dalam query diatas, tanda ? disebut sebagai parameter. Kita bisa memberikan nilai yang berbeda ke dalam parameter dalam setiap pemanggilan PreparedStatement.

Method `setString`, `setFloat`, `setInt` dan beberapa method lain digunakan untuk memasukkan nilai dari setiap parameter. Method tersebut mempunyai dua parameter, parameter pertama adalah int yang digunakan untuk menentukan parameter PreparedStatement mana yang akan diberi nilai. Parameter kedua adalah nilai yang akan dimasukkan ke dalam PreparedStatement, tipe data dari parameter kedua tergantung dari method yang digunakan. Berdasarkan kode diatas, berikut ini contoh penggunaan method `PreparedStatement.setString` :

```
ps.setString(1,"andy");
ps.setString(2,"rizal");
```

Kode diatas memberikan contoh bagaimana memasukkan nilai ke dalam parameter PreparedStatement. Baris pertama memasukkan String "andy" ke dalam parameter pertama dan baris kedua memasukkan String "rizal" ke parameter kedua. Sehingga pemanggilan query oleh PreparedStatement berdasarkan kode diatas sama dengan query statement di bawah ini :

```
"update Customers set nama = 'andy' where nama = 'rizal'"
```

Berikut ini contoh lengkap penggunaan PreparedStatement untuk melakukan update dan insert data :

```
PreparedStatement pInsert = conn.prepareStatement(
    "insert into Customer(nama) values(?)");
pInsert.setString(1,"dian");
pInsert.executeUpdate();
```

```
PreparedStatement pUpdate = conn.prepareStatement(  
    "update Customer set nama=? where nama=?");  
pUpdate.setString(1,"andry");  
pUpdate.setString(2,"andri");  
pUpdate.executeUpdate();
```

Dalam contoh diatas, insert dan update data hanya dilaksanakan sekali saja, hal ini tidak memberikan gambaran yang tepat untuk melihat keunggulan PreparedStatement dibandingkan Statement.executeUpdate.

Misalnya kita ingin meng-insert seratus baris data dalam sebuah loop, kita bisa menggunakan fasilitas batch execution dari PreparedStatement. batch execution mengumpulkan semua eksekusi program yang akan dilaksanakan, setelah semuanya terkumpul batch execution kemudian mengirimkan kumpulan eksekusi program secara bersamaan ke DBMS dalam satu kesatuan. Metode ini sangat efisien karena mengurangi overhead yang diperlukan program untuk berkomunikasi dengan DBMS.

Dalam contoh di bawah ini kita akan menggunakan batch execution untuk melakukan insert data sebanyak seratus kali.

```
PreparedStatement pInsert = conn.prepareStatement(  
    "insert into Customer(nama) values(?)");  
for(int i=0;i<100;i++){  
    pInsert.setString(1,"user ke " + i);  
    pInsert.addBatch();  
}  
pInsert.executeBatch();
```

Setiap kali iterasi, method setString dipanggil untuk mengisi sebuah string ke dalam PreparedStatement, kemudian method addBatch dipanggil untuk mengumpulkan batch dalam satu wadah. Setelah iterasi selesai, method executeBatch dipanggil untuk

melaksanakan semua keseratus instruksi insert secara berurut dengan sekali saja melaksanakan koneksi ke database.

Setelah mengenal JDBC, yang kita perlukan berikutnya adalah menampilkan data dalam komponen swing seperti JTable dan JList. komponen swing tersebut juga bertugas untuk mendapatkan interaksi dengan user. Jika user menjalankan instruksi update, insert atau delete melalui komponen swing, maka data yang ada dalam database juga harus berubah. Kontrol dua arah akan sangat mudah jika dikelola dengan library databinding.

DataBinding Menggunakan GlazedLists

DataBinding adalah library yang menyederhanakan programmer dalam menampilkan Model ke dalam komponen swing dan mengupdate Model ketika user melakukan interaksi dengan komponen swing. Dalam pattern MVC, Model dan View terpisah, sehingga ketika Model berubah, kita harus memberitahu View bahwa Model telah berubah agar View memperbarui tampilannya untuk merefleksikan perubahan yang terjadi. Begitu juga ketika user berinteraksi dengan View, kita harus mengupdate Model.

Pola sinkronisasi data antar Model dan View ini terkadang sangat pelik dan rentan menimbulkan bug. DataBinding membantu programmer untuk mengelola sinkronisasi antar Model dan View, tidak saja membantu tetapi juga mengurangi waktu kita untuk mencari dan mengatasi bug yang mungkin muncul.

GlazedLists adalah salah satu jenis library databinding yang sangat mudah penggunaannya. GlazedLists Mengurangi jumlah kode untuk mensinkronisasi antara Model dan View secara sangat signifikan serta menambahkan kemampuan sorting dan filtering ke dalam komponen swing.

Dalam chapter sebelumnya kita belajar bagaimana menggunakan Model untuk menampilkan data dalam komponen swing. Untuk menampilkan data dalam table kita harus membuat object yang bertipe TableModel, begitu juga kita harus membuat object yang bertipe ListModel untuk menampilkan data dalam JList. Dalam chapter ini kita akan membahas bagaimana caranya menampilkan Model dalam komponen swing sekaligus mengupdate model ketika

user berinteraksi dengan komponen swing menggunakan library databinding.

EventList

EventList adalah sebuah interface yang meng-extends interface `java.util.List`, dalam implementasinya EventList menggunakan `ArrayList`. GlazedLists menggunakan EventList ini sebagai collection yang digunakan secara konsisten untuk semua tipe collection. `AbstractList` adalah list yang secara langsung mengimplement interface EventList sedangkan `BasicEventList` dan `TransformedList`, menegextends `AbstractList`.

EventList mempunyai hubungan yang erat dengan `EventListener`. Ketika terjadi perubahan dalam EventList, misalnya method `add`, atau `remove` dipanggil, EventList akan memicu sebuah event yang kemudian akan ditangkap oleh `EventListener` yang bertugas untuk mempertahankan Model dalam komponen swing tetap up-to-date.

Secara umum EventList sama dengan `ArrayList` atau `Vector`, namun EventList mempunyai beberapa feature tambahan, antara lain:

- **Event Listener** : EventList akan memicu event jika terjadi perubahan dalamnya, sehingga model dari komponen Swing akan tetap up-to-date.
- **Concurrency** : EventList mempunyai lock sehingga kita bisa membagi EventList ke beberapa thread secara aman.

Struktur class GlazedLists dimulai dengan class `AbstractEventList` yang mengimplement `EventList`. Class konkret dari `AbstractEventList` adalah `BasicEventList` dan `TransformedList`. `BasicEventList` tidak mempunyai kelebihan apa-apa, hanya implementasi dasar dari `EventList`, sedangkan `TransformedList` merupakan class yang digunakan untuk mengubah-ubah wujud dari

EventList, misalnya melakukan sorting, filtering dan konvert object dalam EventList dari satu bentuk ke bentuk yang lain. Lebih lanjut tentang TransformedList akan dibahas di bagian sorting dan filtering.

EventList dari Record-Record Database

Dalam prakteknya, EventList akan menampung koleksi dari object yang merepresentasikan sebuah Entity, misalnya: Customer, Product, ProductType dan sebagainya. Sehingga jika kita mempunyai sebuah database yang datanya berbentuk tabular, kita harus mempunyai mekanisme untuk melakukan mapping sederhana antara table dan Entity.

Berikut ini adalah contoh mapping sederhana table CUSTOMER dengan sebuah Entity class Customer:

```
CREATE TABLE `CUSTOMERS` (  
  `Id` int(11) NOT NULL auto_increment,  
  `Nama` varchar(100) NOT NULL default '',  
  `jenisKelamin` varchar(50) NOT NULL default 'laki-  
laki',  
  `agama` varchar(20) NOT NULL default 'islam',  
  `status` varchar(20) NOT NULL default 'single',  
  `pekerjaan` varchar(100) NOT NULL default 'Mahasiswa',  
  PRIMARY KEY (`Id`)  
) ENGINE=MyISAM AUTO_INCREMENT=18 DEFAULT CHARSET=latin1;
```

Di bawah ini adalah class Customer yang digunakan untuk menampung record-record dari table Customer di atas.

```

public class Customer {
    private int id;
    private String nama, status, agama, jenisKelamin,
        pekerjaan;
    //getter dan setter method di sini
}

```

Untuk melakukan mapping antara table CUSTOMER dan object Customer diperlukan sebuah class sederhana yang akan mengambil record-record dari table CUSTOMER dan membuat sebuah object Customer dari setiap record tersebut. Kita namakan class ini misalnya DBAccess, kodenya :

```

public class DBAccess {
    private static Connection c;
    private static Statement selectStatement;
    static{
        try{
            Class.forName("com.mysql.jdbc.Driver");
            c = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/latihan",
                "root", "");
            selectStatement = c.createStatement(
                ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } catch(ClassNotFoundException e){
            e.printStackTrace();
        } catch(SQLException e){ e.printStackTrace(); }
    }
}

```

```

public static EventList<Customer> getCustomers()
    throws SQLException{
    EventList<Customer> list =
        new BasicEventList<Customer>();
    ResultSet rs = selectStatement.executeQuery(
        "select * from CUSTOMERS");
    while(rs.next()){
        list.add(getCustomer(rs));
    }
    return list;
}
public static Customer getCustomer(ResultSet rs)
    throws SQLException{
    Customer customer = new Customer();
    customer.setId(rs.getInt(1));
    customer.setNama(rs.getString(2));
    customer.setJenisKelamin(rs.getString(3));
    customer.setAgama(rs.getString(4));
    customer.setStatus(rs.getString(5));
    customer.setPekerjaan(rs.getString(6));
    return customer;
}
}

```

Class DBAccess akan melakukan koneksi ke database ketika diload, hal ini ditandai dengan adanya statement static{} di dalam class DBAccess. Setelah sukses di load, kita bisa menggunakan method getCustomers untuk memperoleh sebuah object EventList yang sudah berisi semua record-record dari table CUSTOMERS yang telah dimapping ke dalam object Customer. Method getCustomer digunakan untuk mengambil baris yang sedang aktif dari sebuah ResultSet kemudian membuat instant dari class customer dan

melakukan mapping record table CUSTOMER ke dalam object Customer.

Setelah EventList sukses dibuat, maka langkah selanjutnya adalah membuat model dari komponen swing.

EventList dalam Model

JTable, JComboBox dan JList adalah komponen swing yang masing-masing memiliki Model. Model dari ketiga komponen tersebut membutuhkan collection untuk menyimpan data, hal ini jelas logis karena ketiga komponen tersebut menampilkan data yang jamak, tidak data tunggal. Library GlazedLists menyediakan class model yang mengimplement masing-masing model dari ketiga komponen tersebut dan menggunakan EventList sebagai collection dari model-model tersebut.

EventListModel

EventListModel mengimplement interface ListModel dan menerima EventList sebagai argumen dari constructornya. Dengan mudah kita dapat menggunakan EventListModel sebagai model untuk komponen JList.

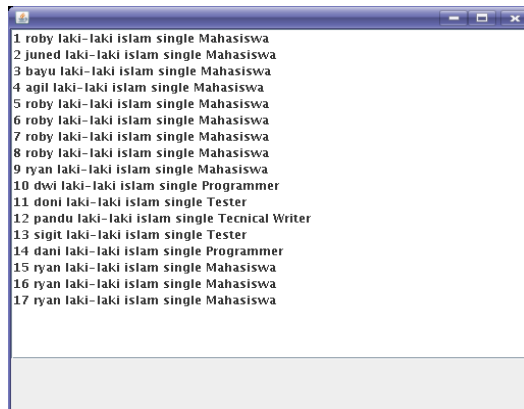
JList menampilkan object-object di dalam model sebagai list satu dimensi, sehingga diperlukan mekanisme untuk menampilkan sebuah object dalam sebuah String. Mekanisme ini disediakan oleh method toString dari class Object, kita override method toString dari class Customer diatas menjadi :

```
public String toString() {
    return id + " " + nama + " " + jenisKelamin + " " +
        agama + " " + status + " " + pekerjaan;
}
```

Sekarang method toString dari class Customer akan

mengembalikan gabungan dari semua nilai dari field-field yang dipunyai oleh class Customer.

Berikut ini adalah sebuah program sederhana untuk menampilkan isi table CUSTOMER ke dalam sebuah JList menggunakan library GlazedLists. Kita akan menggunakan class DBAccess yang telah dibuat sebelumnya.



Aplikasi GlazedLists yang menampilkan Customer dalam JList

Untuk membuat aplikasi diatas lakukan langkah-langkah berikut :

1. Buat sebuah class JFrame, beri nama CustomersList.java
2. Tambahkan komponen JList ke dalam CustomersList, beri nama listCustomers.
3. Bukalah Jendela Code dan tambahkan baris kode ini ke dalam class constructor class CustomersList di bagian bawah dari baris pemanggilan method initComponents :

```
try {
    EventList<Customer> list =
        DBAccess.getCustomers();
    EventListModel model = new EventListModel(list);
    listCustomers.setModel(model);
} catch (SQLException ex) {
    ex.printStackTrace();
}
```

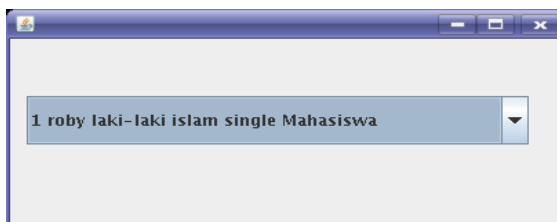
4. Compile dan jalankan program diatas.

Bisa kita lihat, kode diatas sangat sederhana sekali. Setelah kita mempunyai class yang bertugas untuk melakukan mapping antara table CUSTOMER dan class Customer, GlazedLists akan mempermudah pekerjaan kita menampilkannya ke dalam JList.

EventComboBoxModel

Seperti juga JList, JComboBox mempunyai karakteristik untuk menampilkan koleksi data. Sama juga seperti JList, JComboBox menggunakan method toString dari sebuah class sebagai representasi object tersebut dalam item JComboBox.

Berikut ini contoh aplikasi sederhana untuk menampilkan isi table CUSTOMER dalam JComboBox. Menampilkan data Customer dalam JComboBox :



Aplikasi sederhana menampilkan data customer di JComboBox

Untuk membuat aplikasi diatas lakukan langkah-langkah berikut ini:

1. Buat sebuah class JFrame dan beri nama CustomersCombo
2. Letakkan sebuah JComboBox ke dalam class JFrame tersebut, beri nama cmbCustomer.
3. Tuliskan kode berikut ini di dalam konstruktor dari class CustomersCombo di bagian bawah dari baris pemanggilan method initComponents.

```
try {  
    EventList<Customer> customers =  
        DBAccess.getCustomers();  
    EventComboBoxModel model =  
        new EventComboBoxModel(customers);  
    cmbCustomers.setModel(model);  
    cmbCustomers.setSelectedIndex(0);  
} catch (SQLException ex) {  
    ex.printStackTrace();  
}
```

4. Compile dan jalankan programnya.

Kode diatas sangat mirip dengan kode untuk menampilkan EventList ke dalam komponen JList, hal ini karena karakteristik JList dan JComboBox yang mirip, yang berbeda hanyalah behaviour dari keduanya. JComboBox hanya dapat memilih satu pilihan sedangkan JList dapat memilih lebih dari satu pilihan.

Menampilkan EventList ke dalam JTable berbeda dengan JList dan JComboBox, karena JTable memiliki struktur data tabular dimana setiap baris dibagi-bagi dalam kolom.

EventTableModel

JTable memiliki struktur data tabular dua dimensi, sedangkan EventList adalah list object satu dimensi. Memasukkan EventList dalam EventTableModel memerlukan class yang mengimplement

interface `TableFormat` dari `GlazedLists`. Interface `TableFormat` digunakan untuk mendefinisikan beberapa hal yang berhubungan dengan kolom dari `JTable`, antara lain: jumlah kolom, nama kolom, dan nilai kolom. Dengan class `TableFormat` ini `EventList` yang hanya sebuah list dapat dipandang menjadi sebuah data tabular dua dimensi.

Untuk lebih jelasnya, perhatikan class `CustomerTableFormat` berikut ini :

```
public class CustomerTableFormat implements TableFormat{
    public CustomerTableFormat() { }
    public int getColumnCount() { return 6; }
    public String getColumnName(int column) {
        switch(column){
            case 0: return "Id";
            case 1: return "Nama";
            case 2: return "Jenis Kelamin";
            case 3: return "Agama";
            case 4: return "Status";
            case 5: return "Pekerjaan";
            default: return "";
        }
    }
    public Object getColumnValue(Object baseObject,
        int column) {
        Customer customer = (Customer) baseObject;
        switch(column){
            case 0: return customer.getId();
            case 1: return customer.getNama();
        }
    }
}
```

```

        case 2: return customer.getJenisKelamin();
        case 3: return customer.getAgama();
        case 4: return customer.getStatus();
        case 5: return customer.getPekerjaan();
        default: return "";
    }
}
}

```

Method `getColumnCount` mengembalikan jumlah kolom, kemudian method `getColumnName` mengembalikan nama kolom, dan terakhir method `getColumnValue` mengembalikan nilai kolom dari `baseObject`. `baseObject` ini sebenarnya adalah 'baris' atau dalam hal ini adalah object `Customer`.

Kita akan membuat sebuah aplikasi kecil untuk menampilkan `Customer` ke dalam suatu table dengan bantuan class `CustomerTableFormat` yang telah kita buat sebelumnya.

The screenshot shows a window with a JTable containing the following data:

Id	Nama	Jenis Kelamin	Agama	Status	Pekerjaan
4	agil	laki-laki	islam	single	Mahasiswa
3	bayu	laki-laki	islam	single	Mahasiswa
14	dani	laki-laki	islam	single	Programmer
11	doni	laki-laki	islam	single	Tester
10	dwi	laki-laki	islam	single	Programmer
2	juned	laki-laki	islam	single	Mahasiswa
12	pandu	laki-laki	islam	single	Technical W...
1	robby	laki-laki	islam	single	Mahasiswa
5	robby	laki-laki	islam	single	Mahasiswa
6	robby	laki-laki	islam	single	Mahasiswa

Data Customer ditampilkan dalam JTable

Untuk membuat aplikasi seperti di atas lakukan langkah-langkah berikut ini :

1. Buat class `JFrame` baru, beri nama `CustomersTable`
2. Letakkan sebuah `JTable` dalam class `CustomersTable`, beri nama `tblCustomers`
3. Ketik kode berikut ini di dalam constructor class

CustomersTable di bagian bawah dari baris pemanggilan method initComponents.

```
try{
    EventList<Customer> list =
        DBAccess.getCustomers();
    EventTableModel model =
        new EventTableModel(sortedList,
            new CustomerTableFormat());
    tblCustomers.setModel(model);
} catch(SQLException e){
    e.printStackTrace();
}
```

4. Compile dan jalankan programnya

Bisa kita lihat bahwa proses menampilkan data dari database menggunakan bantuan GlazedLists sangat mudah dan mengurangi banyak sekali code yang tidak perlu. Kita juga tidak perlu membuat class yang mengimplement TableModel, karena GlazedLists sudah menyediakan EventTableModel. Tetapi tetap saja kita memerlukan sebuah class yang mengimplement TableFormat untuk mendefinisikan kolom dari tabel, untungnya mengimplement interface TableFormat lebih mudah dan sederhana dibandingkan harus mengimplement TableModel.

Satu EventList untuk Semua Model

Menampilkan isi database ke dalam JList, JComboBox maupun JTable sangat sederhana dan yang paling penting adalah ketiganya dapat menerima EventList dalam membentuk Model untuk masing-masing komponen. Fleksibilitas ini sangat berguna jika kita harus menampilkan sebuah data dalam beberapa format, tidak ada lagi langkah-langkah untuk melakukan konvert dari ListModel ke ComboModel atau ke TableModel, semuanya disederhanakan oleh

GlazedLists dengan hanya menggunakan EventList.

Cermati contoh-contoh kode di atas, maka anda akan melihat bahwa perbedaan kodenya terletak di dalam jenis Model yang akan digunakan oleh setiap kontrol.

Di bagian-bagian selanjutnya kita akan membahas bagaimana GlazedLists membantu kita menyederhanakan kode dan memudahkan kita dengan menyediakan berbagai class untuk memanipulasi EventList.

Pengurutan Tabel

SortedList adalah class turunan dari TransformedList yang digunakan untuk menampilkan EventList dalam keadaan terurut. Setiap TransformedList termasuk juga SortedList, akan mendengarkan event dari EventList. Ketika EventList berubah maka TransformedList juga berubah menyesuaikan dengan EventList.

Untuk membuat SortedList, diperlukan class yang mengimplement interface Comparator. Class Comparator ini digunakan SortedList untuk menentukan urutan dari class yang disimpan dalam EventList, dalam hal ini class Customer. Jadi kita harus membuat class baru yang mengimplement interface Comparator untuk membandingkan antar object customer. Dalam contoh ini kita menggunakan field nama sebagai parameter perbandingan.

Berikut ini adalah class CustomerComparator yang digunakan untuk membandingkan satu object customer dengan dengan object customer lainnya :

```
public class CustomerComparator implements Comparator{
    public CustomerComparator() { }
    public int compare(Object a, Object b) {
        Customer custA = (Customer) a;
        Customer custB = (Customer) b;
        return custA.getNama().compareTo(custB.getNama());
    }
}
```

Sekarang dengan menggunakan class ini kita bisa membandingkan apakah suatu object customer 'lebih besar' dari object customer yang lain. Perhitungannya didasarkan pada nilai dari field lama dalam object customer. Misalnya ada object customer yang nilai field nama-nya adalah "robi" dan ada object customer yang lain dimana nilai field nama-nya adalah "ilham". Maka object customer dengan nama "robi" 'lebih besar' dibandingkan object customer dengan nama 'ilham'.

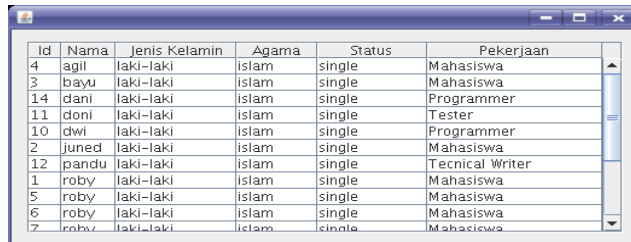
Dengan bantuan class CustomerComparator kita bisa dengan mudah mengurutkan EventList dengan menggunakan SortedList. Berikut ini contoh penggunaannya:

```
EventList customers = DBAccess.getCustomers();
SortedList sortedList =
    new SortedList(customers,
        new CustomerComparator())
```

Manual Sorting

Setelah EventList diurutkan dalam SortedList, kita bisa membuat EventTableModel yang didalamnya dimasukkan SortedList kemudian menggunakan EventTableModel tersebut sebagai TableModel dari suatu JTable. Secara otomatis JTable tersebut akan terurut barisnya berdasarkan namanya, lho kok bisa? hal ini dikarenakan kita menggunakan CustomerComparator yang

membandingkan satu customer dengan customer lainnya hanya berdasarkan pada field nama.



Id	Nama	Jenis Kelamin	Agama	Status	Pekerjaan
4	agil	laki-laki	islam	single	Mahasiswa
3	bayu	laki-laki	islam	single	Mahasiswa
14	dani	laki-laki	islam	single	Programmer
11	doni	laki-laki	islam	single	Tester
10	dwi	laki-laki	islam	single	Programmer
2	juned	laki-laki	islam	single	Mahasiswa
12	pandu	laki-laki	islam	single	Technical Writer
1	roby	laki-laki	islam	single	Mahasiswa
5	roby	laki-laki	islam	single	Mahasiswa
6	roby	laki-laki	islam	single	Mahasiswa
7	roby	laki-laki	islam	single	Mahasiswa

Data Customer diurutkan dengan SortedList

Program diatas menampilkan jTable yang telah terurut berdasarkan namanya. Ikuti langkah-langkah berikut ini untuk membuat aplikasi diatas :

1. Buat sebuah class JFrame, beri nama CustomerTableSorted.
2. Masukkan satu buah jTable ke dalam CustomerTableSorted dan beri nama tblCustomers.
3. Buka Jendela Code dan ubah method constructor dari class CustomerTableSorted menjadi seperti di bawah ini :

```
public CustomerTableSorted() {
    initComponents();
    EventList<Customer> customers;
    try {
        customers = DBAccess.getCustomers();
        SortedList sortedList =
            new SortedList(customers,
                new CustomerComparator());
        EventTableModel model =
            new EventTableModel(sortedList,
                new CustomerTableFormat());
    }
}
```

```
tblCustomer.setModel(model);
    }catch(SQLException ex){ex.printStackTrace();}
}
```

4. Compile dan jalankan program diatas.

Automatic Sorting

Sekarang pertanyaanya, kalau misalnya kita ingin melakukan sorting table berdasarkan idnya, apakah harus membuat class CustomerComparator lainnya yang membandingkan satu customer dengan customer lainnya berdasarkan id? jawabanya adalah tidak perlu.

GlazedLists menyediakan class TableComparatorChooser yang akan menambahkan komponen sorter yang diletakkan di bagian header dari kolom jTable. Jadi ketika kita melakukan double klik pada header kolom tersebut, jTable akan otomatis terurut berdasarkan kolom tersebut. Perhatikan gambar di bawah ini :

Id	Nama	Jenis Kelamin	Agama	Status	Pekerjaan ▲
1	robby	laki-laki	islam	single	Mahasiswa
2	juned	laki-laki	islam	single	Mahasiswa
3	bayu	laki-laki	islam	single	Mahasiswa
4	agil	laki-laki	islam	single	Mahasiswa
5	robby	laki-laki	islam	single	Mahasiswa
6	robby	laki-laki	islam	single	Mahasiswa
7	robby	laki-laki	islam	single	Mahasiswa
8	robby	laki-laki	islam	single	Mahasiswa
9	ryan	laki-laki	islam	single	Mahasiswa
15	ryan	laki-laki	islam	single	Mahasiswa

TableComparatorChooser mengurutkan data secara otomatis

Sekarang table diatar terurut berdasarkan pekerjaan, karena header dari kolom pekerjaan di klik dan sekarang terlihat ada tanda segitiga kecil menghadap keatas yang artinya baris table terurut ascendant (menaik) berdasarkan kolom pekerjaan.

Sekarang perhatikan lagi gambar di bawah ini :

Id	Nama	Jenis Kelamin	Agama	Status	Pekerjaan
4	agil	laki-laki	islam	single	Mahasiswa
3	bayu	laki-laki	islam	single	Mahasiswa
2	juned	laki-laki	islam	single	Mahasiswa
1	robby	laki-laki	islam	single	Mahasiswa
5	robby	laki-laki	islam	single	Mahasiswa
6	robby	laki-laki	islam	single	Mahasiswa
7	robby	laki-laki	islam	single	Mahasiswa
8	robby	laki-laki	islam	single	Mahasiswa
9	ryan	laki-laki	islam	single	Mahasiswa
15	ryan	laki-laki	islam	single	Mahasiswa

Cascade sorting menggunakan TableComparatorChooser

Terlihat juga ada tanda segitiga kecil di header kolom nama, artinya bahwa table diatas diurutkan terlebih dahulu berdasarkan pekerjaan kemudian diurutkan berdasarkan nama. Hal ini terjadi karena tadi setelah header kolom pekerjaan diklik, kemudian header nama diklik secara berurutan. Fasilitas ini disediakan oleh class TableComparatorChooser.

Sekarang ikuti langkah-langkah berikut ini untuk membuat program seperti diaatas :

1. Buat class JFrame baru, beri nama CustomersTableAutomaticSorted.
2. Tambahkan sebuah jTable.
3. Modifikasi method constructornya menjadi seperti berikut ini :

```
public CustomerTableAutomaticSorted() {
    initComponents();
    EventList<Customer> customers;
    try {
        customers = DBAccess.getCustomers();
        SortedList sortedList =
            new SortedList(customers,
                new CustomerComparator());
    }
}
```

```

EventTableModel model =
    new EventTableModel(sortedList,
        new CustomerTableFormat());
tblCustomer.setModel(model);
TableComparatorChooser chooser =
    new TableComparatorChooser(
        tblCustomer, sortedList, true);
} catch (SQLException ex) {ex.printStackTrace();}
}

```

4. Compile dan jalankan programnya.

Jika kita perhatikan kode diatas, hanya ada perbedaan kecil dengan program sebelumnya. Perbedaanya terletak pada adanya class TableComparatorChooser.

Menyaring Text dalam Tabel

Fasilitas filtering data didukung penuh oleh GlazedLists. Menggunakan fasilitas filtering di GlazedLists sama mudahnya dengan fasilitas sorting. Text filtering didukung penuh oleh GlazedLists, sedangkan filtering berdasarkan suatu kriteria tertentu harus dibuat sendiri.

TextFilterator

Sebelum melakukan filtering kita perlu memberitahu GlazedLists field apa saja dalam class Customer yang digunakan sebagai basis filtering. Misalnya kita menggunakan semua field sebagai basis filtering kecuali field id. Untuk menentukan aturan tersebut harus dibuat sebuah class baru yang mengimplement TextFilterator.

Berikut ini class CustomerFilter yang mengimplement TextFilterator:

```

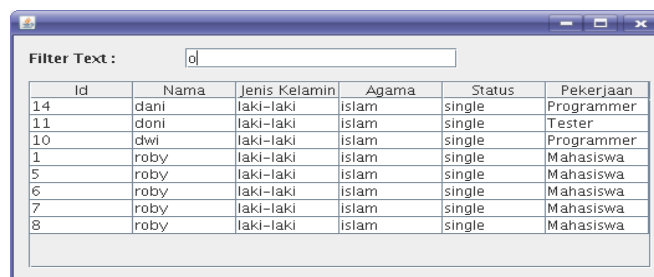
public class CustomerFilter implements TextFilterator{
    public CustomerFilter() {    }
    public void getFilterStrings(List baseList,
        Object element) {
        Customer customer = (Customer) element;
        baseList.add(customer.getNama());
        baseList.add(customer.getJenisKelamin());
        baseList.add(customer.getAgama());
        baseList.add(customer.getStatus());
        baseList.add(customer.getPekerjaan());
    }
}

```

Method `getFilterStrings` digunakan untuk mendefinisikan field apa saja yang akan digunakan sebagai basis filtering. Dalam hal ini semua field, kecuali field id digunakan sebagai basis filtering.

Memfilter Data Customer

Perhatikan aplikasi sederhana di bawah ini, terdapat sebuah `JTable` yang menampung data `Customer` dan sebuah `JTextArea` yang digunakan untuk mengisikan string filter. Setiap kali user mengetikkan huruf dalam `JTextField`, `JTable` di bawahnya akan terfilter berdasarkan text yang dimasukkan dalam `JTextField`.



Id	Nama	Jenis Kelamin	Agama	Status	Pekerjaan
14	dani	laki-laki	islam	single	Programmer
11	doni	laki-laki	islam	single	Tester
10	dwi	laki-laki	islam	single	Programmer
1	roby	laki-laki	islam	single	Mahasiswa
5	roby	laki-laki	islam	single	Mahasiswa
6	roby	laki-laki	islam	single	Mahasiswa
7	roby	laki-laki	islam	single	Mahasiswa
8	roby	laki-laki	islam	single	Mahasiswa

Memfilter isi table menggunakan `TextFilterator`

Untuk membuat aplikasi seperti di atas lakukan langkah-langkah

berikut ini :

1. Buat class JFrame baru, beri nama CustomersTableFilter.
2. Masukkan sebuah jTable dengan nama tblCustomer, JTextField dengan nama txtFilter dan JLabel dengan nama lblFilter.
3. Modifikasi method constructor dari class CustomersTableFilter menjadi seperti di bawah ini:

```
public CustomersTableFilter() {
    initComponents();
    EventList<Customer> list = DBAccess.getCustomers();
    SortedList<Customer> sortedList =
        new SortedList<Customer>(list,
            new CustomerComparator());
    FilterList filterList =
        new FilterList(sortedList,
            new TextComponentMatcherEditor(txtFilter,
                new CustomerFilter()));
    EventTableModel model =
        new EventTableModel(filterList,
            new CustomerTableFormat());
    tblCustomers.setModel(model);
}
```

4. Compile dan jalankan aplikasinya.

Dalam kode diatas kita ada satu class lagi yang berperan untuk menyaring text dari table yaitu TextComponentMatcherEditor. Class ini memerlukan class CustomerFilter yang telah kita buat sebelumnya. TextComponentMatcherEditor memerlukan method getFilterStrings dari class CustomerFilter untuk menentukan apakah suatu baris terfilter atau tidak. Caranya yaitu dengan mencari pola String dari txtFilter dalam class Customer, method getFilterStrings mendaftarkan semua field yang digunakan sebagai

basis penyaringan.

Sebagai contohnya, dalam program kita di atas `CustomerFilter` mendefinisikan bahwa semua field kecuali field `id` akan digunakan sebagai basis pencarian. Class `TextComponentMatcherEditor` akan mengambil `String` yang dimasukkan dalam `txtFilter`, yaitu huruf 'o'. Kemudian melihat baris-baris dalam `tblCustomer` dan melakukan filtering terhadap baris-baris tersebut, yang tersisa adalah baris-baris (atau object-object `customer`) yang dalam field `nama`, `jeniskelamin`, `pekerjaan` dan `agama` terdapat huruf 'o'.

Contoh lainnya, jika kita memasukkan angka 1, maka semua baris dalam `tblCustomer` akan terfilter, tidak tersisa satu baris pun. Lho kok bisa? bukanya ada kolom `id` yang mengandung angka 1? Nah, kita harus melihat lagi dalam class `CustomerFilter`, apakah field `id` dimasukkan dalam basis penyaringan? ternyata tidak. Sehingga `TextComponentMatcherEditor` tidak melakukan pencarian angka '1' dari field `id`. Hasilnya jelas terlihat bahwa semua baris akan terfilter.

`FilterList` sangat berguna jika kita banyak membuat jendela pencarian. Baik full-text search ke semua kolom, atau pencarian berdasarkan kriteria tertentu. `FilterList` dan `TextComponentMatcherEditor` mengurangi kode secara signifikan. Bayangkan anda harus membuat program seperti di atas tanpa library `GlazedLists`, anda harus handle event penekanan tombol pada `txtFilter`, kemudian menyaring baris-baris `tblCustomer` dan merefresh view `tblCustomer`. Anda akan memerlukan puluhan baris kode. Dengan `GlazedLists` anda hanya memerlukan lima baris kode saja, fantastis!.

Sampai dengan disini kita sudah tahu mengenai bagaimana melakukan sorting dan filtering `EventList` menggunakan class-class yang meng-extends `TransformedList`, masih ada beberapa lagi

class-class yang dapat kita gunakan memanipulasi `EventList` dengan metode-metode yang berbeda.

TransformedList dan UniqueList

`TransformedList` pada umumnya digunakan untuk melakukan manipulasi terhadap `EventList`, baik mensort atau memfilter. Di bagian sebelumnya kita sudah mengenal `SortedList` dan `FilterList`, kedua List ini adalah turunan dari `TransformedList`. Di bagian ini kita akan membahas tentang `UniqueList`, turunan lain dari `TransformedList`. `UniqueList` mirip dengan `FilterList`, yaitu menyaring `EventList` dan menyisakan item-item yang memenuhi kriteria tertentu.

`UniqueList` akan menyaring `EventList` dan hanya menyisakan item-item yang 'unique'. `UniqueList` ini gunanya semisal : mendaftar semua jenis pekerjaan, mengambil daftar nama mahasiswa, mendapatkan daftar agama. Semua kebutuhan tersebut memerlukan nilai unique dari masing-masing kolom, jika ada dua nilai yang sama dalam satu kolom, hanya perlu ditampilkan satu item saja. Inilah konsep 'unique' dari `UniqueList`.

Pertanyaanya adalah kriteria 'unique' ini berdasarkan apa? berdasarkan id? berdasarkan nama? atau berdasarkan kriteria yang lain?. Untuk menjawab pertanyaan tersebut kita memerlukan class yang mentransform `EventList` yang berisi `Customer` menjadi hanya berisi salah satu field `Customer` saja. Semisal kita akan membutuhkan daftar pekerjaan yang ada dalam table `CUSTOMER`, yang harus kita lakukan pertama-tama adalah membuat class `TransformedList` yang mengubah `EventList` yang berisi object-object `customer` seolah-olah berisi hanya field pekerjaan saja. Berikut ini kodenya :

```

public class CustomerJobTransformator
    extends TransformedList {
    public CustomerJobTransformator(EventList source) {
        super(source);
        source.addListEventListener(this);
    }
    public void listChanged(ListEvent listChanges) {
        updates.forwardEvent(listChanges);
    }
    public Object get(int index) {
        return ((Customer)source.get(index))
            .getPekerjaan();
    }
}

```

Perhatikan bagian kode yang dicetak miring, method `get` digunakan untuk mengambil isi dari `EventList` (`source`) pada index ke-x. Class `CustomerJobTransformator` meng-override method `get` agar mengembalikan nilai field pekerjaan bukannya object customer itu sendiri. Hal ini menyebabkan ketika diambil semua item dari class `CustomerJobTransformator`, yang dikeluarkan bukannya object-object customer tetapi nilai field-field pekerjaan dari setiap customer. Dapat dikatakan bahwa, class `CustomerJobTransformator` mentransform `List Customer` menjadi `List pekerjaan`.

Setelah kita bisa mendapatkan semua nilai field pekerjaan dengan menggunakan class `CustomerJobTransformator`, kita tinggal menyaring pekerjaan-pekerjaan tersebut agar mempunyai nilai unique. Untuk lebih jelasnya perhatikan kode berikut ini :

```

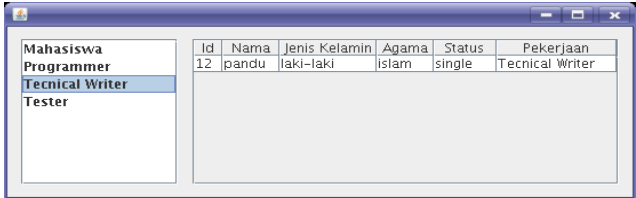
EventList<Customer> list = DBAccess.getCustomers();
CustomerJobTransformator customerJobs =
    new CustomerJobTransformator(list);
UniqueList uniqueCustomerJobs =
    new UniqueList(customerJobs);
    
```

Bisa kita lihat diatas hanya ada tiga baris kode. Baris pertama mengambil semua object customer dari table CUSTOMER. Baris kedua mentransform daftar customer menjadi daftar pekerjaan. Dan baris ketiga menyaring daftar pekerjaan agar menjadi daftar pekerjaan yang unique.

UniqueList ini salah satu kegunaanya adalah untuk membuat daftar Master-Detail. Gambaran singkat Master-Detail ini adalah kita mempunyai suatu daftar Master, misalnya pekerjaan, kemudian kita juga mempunyai daftar data Detail, misalnya customer. Kemudian ketika kita memilih salah satu daftar Master, misalnya pekerjaan = 'Mahasiswa' maka secara otomatis daftar Detail akan menampilkan customer dengan pekerjaan Mahasiswa saja.

Model Master-Detail

Model Master-Detail ini lazim terdapat dalam aplikasi-aplikasi pengolahan database. Untuk lebih jelasnya perthatikan gambar berikut ini :



Model Master-Detail pekerjaan dan Customer

Gambar ini menampilkan sebuah class JFrame yang mempunyai

sebuah JList di sisi kiri dan jTable di sisi kanan. JList mempunyai daftar unique pekerjaan dari table CUSTOMER dan jTable memegang isi dari table CUSTOMER. Dalam hal ini JList adalah Master dan jTable adalah Detail.

Ketika user memilih satu atau lebih item pekerjaan pada JList, maka jTable akan difilter berdasarkan pekerjaan dari setiap customer. Misalnya kita memilih pekerjaan 'Technical Writer', maka hanya customer-customer yang mempunyai pekerjaan 'Technical Writer' saja yang akan ditampilkan, sisanya difilter.

FilterList, Matcher dan MatcherEditor

Pada bagian sebelumnya kita sudah membahas bagaimana melakukan filtering sebuah table menggunakan TextFilterator dan TextComponentMatcherEditor. Model Master-Detail pada dasarnya mirip dengan text filtering, perbedaanya ada pada kategori filtering dan basis filtering.

Text filtering pada bagian sebelumnya, menggunakan sebuah JTextField, txtFilter, sebagai kategori filtering dan TextFilterator sebagai basis filtering. Cara kerjanya, GlazedLists akan mencocokkan string dari txtFilter di dalam field-field customer yang didefinisikan sebagai basis filtering oleh TextFilterator.

Model Master-Detail mengambil kategori filtering dari JList dan basis filteringnya sesuai dengan kategori filtering yang ditampilkan oleh JList. Seperti pada contoh gambar sebelumnya, JList menampilkan daftar unique pekerjaan customer, maka basis filteringnya adalah field pekerjaan dari customer.

Matcher adalah interface yang digunakan GlazedLists untuk menentukan apakah sebuah item dalam EventList cocok dengan kriteria tertentu atau tidak. Jika cocok maka item tersebut akan dimasukkan ke dalam FilterList. Namun Matcher ini bersifat immutable, artinya kriteria pencocokanya tidak dapat dirubah.

Misalnya `Matcher` mempunyai kriteria pekerjaan 'Mahasiswa', maka kriteria tersebut tidak dapat dirubah menjadi 'Technical Writer'. Hal ini dilakukan untuk tujuan keamanan menggunakan `Matcher` dalam lingkungan multithreading, kita tidak akan membahas lebih lanjut mengenai multithreading.

`MatcherEditor` digunakan jika kriteria filtering berubah secara dinamis. Misalnya sekarang kriterianya 'Mahasiswa' kemudian nanti kriterianya berubah menjadi 'Technical Writer' dan seterusnya. Sehingga model Master-Detail yang mempunyai kriteria filtering berubah-ubah harus dibuatkan `MatcherEditor`-nya.

Berikut ini class yang mengimplement `Matcher` dan `MatcherEditor` yang akan digunakan dalam model Master-Detail.

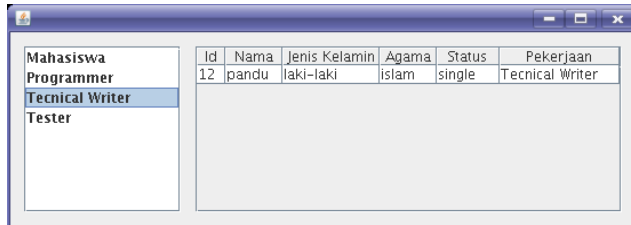
```
public class CustomerJobMatcher implements Matcher{
    private Set customers = new HashSet();
    public CustomerJobMatcher(
        Collection customerSelected) {
        customers.addAll(customerSelected);
    }
    public boolean matches(Object item) {
        if(item == null)
            return false;
        else if (customers.isEmpty())
            return false;
        Customer cust = (Customer) item;
        return customers.contains(cust.getPekerjaan());
    }
}
```

`CustomerJobMatcherEditor` digunakan untuk melakukan matching baris-baris table secara otomatis dan memfilter table berdasarkan pekerjaan.

```
public class CustomerJobMatcherEditor
    extends AbstractMatcherEditor
    implements ListSelectionListener{
private EventList selectedCustomerJob;
public CustomerJobMatcherEditor(JList jList,
    UniqueList uniqueCustomerJob) {
    EventListModel model =
        new EventListModel(uniqueCustomerJob);
    jList.setModel(model);
    EventSelectionModel selectionModel =
        new EventSelectionModel(uniqueCustomerJob);
    jList.setSelectionModel(selectionModel);
    selectedCustomerJob =
        selectionModel.getSelected();
    jList.addListSelectionListener(this);
    }
public void valueChanged(
    ListSelectionEvent listSelectionEvent) {
    Matcher matcher =
        new CustomerJobMatcher(selectedCustomerJob);
    fireChanged(matcher);
    }
}
```

Membuat Model Master-Detail

Mari kita lihat sekali lagi gambar contoh aplikasi Master-Detail :



Implementasi Master-Detail

Untuk membuat aplikasi seperti diatas lakukan langkah-langkah berikut ini :

1. Buat class JFrame baru, beri nama CustomersJobFilter.
2. Tambahkan sebuah JTable dengan nama tblCustomer dan sebuah JList dengan nama lstPekerjaan.
3. Buka Jendela Code dan modifikasi method constructor dari class CustomersJobFilter menjadi seperti berikut ini :

```
public CustomersJobFilter() {
    initComponents();
    EventList<Customer> customers = null;
    try {
        customers = DBAccess.getCustomers();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    SortedList sortedList = new SortedList(customers,
        new CustomerComparator());
    CustomerJobTransformator jobList =
        new CustomerJobTransformator(sortedList);
    UniqueList uniqueList = new UniqueList(jobList);
}
```

```
FilterList filterList =
    new FilterList(sortedList,
        new CustomerJobMatcherEditor(
            lstPekerjaan,uniqueList));
EventTableModel model =
    new EventTableModel(filterList,
        new CustomerTableFormat());
tblCustomer.setModel(model);
TableComparatorChooser comparatorChooser =
    new TableComparatorChooser(tblCustomer,
        sortedList,true);
}
```

4. Compile dan jalankan programnya.

GlazedLists sangat mudah digunakan dan menyederhanakan banyak sekali pekerjaan kita. Terutama pada penanganan event dari komponen swing, dari awal sampai akhir bab ini tidak ada satu pun baris kode untuk menangani event dari komponen swing, secara otomatis GlazedLists yang menangani event tersebut.

Penutup

Dengan ilmu yang sudah diperoleh dalam pelatihan ini, anda sudah bisa mulai untuk membuat program Java desktop sederhana. Pada awalnya pasti terasa sulit, sikapi pantang menyerah dan selalu cari cara yang lebih baik dalam membuat aplikasi.

Langkah selanjutnya anda bisa mulai aktif bertanya atau menjawab hal-hal yang berhubungan dengan Java. Media yang bisa digunakan banyak sekali, bisa forum mulis atau diskusi dengan teman. Ini cara terbaik untuk mengetes apakah pemahaman anda mengenai Java sudah cukup lengkap atau anda masih perlu belajar lebih banyak lagi.

Setelah yakin dengan kemampuan anda, berfikirilah untuk mengambil sertifikasi profesional Java. Pelatihan untuk persiapan sertifikasi Java banyak tersedia di lembaga pelatihan. Kalau anda merasa terlalu berat mengambil kursus persiapan sertifikasi, berlatihlah sendiri menggunakan materi yang banyak tersedia di internet.

Cara belajar Java yang paling efektif adalah dengan melibatkan diri dalam project berbasis Java. Jika di perusahaan anda tidak memungkinkan, di luar sana banyak sekali project opensource yang memerlukan bantuan anda. Berkunjunglah ke website-website open source project hosting seperti sourceforge.net atau dev.java.net

Learn, Try dan Teach adalah formula untuk meningkatkan pengetahuan anda tentang Java. Jika sudah belajar dan sukses mencoba, tularkan ilmu anda pada orang disekeliling anda, dijamin ilmunya bakalan bertambah berlipat-lipat.

Referensi dan Bacaan Lebih Lanjut

1. Java Tutorial : <http://java.sun.com/docs/books/tutorial/>
2. Java Documentation : <http://java.sun.com/javase/6/docs/>
3. Netbeans Indonesia : <http://groups.yahoo.com/group/Netbeans-indonesia/>
4. Java User Group Indonesia : <http://groups.yahoo.com/group/jug-indonesia/>
5. GlazedLists : <http://publicobject.com/glazedlists/>
6. Java Design Pattern : <http://www.javacamp.org/designPattern/>
7. Java Desktop : <http://www.javadesktop.org>
8. JGoodies : <http://www.jgoodies.com>
9. SwingX : <http://www.swinglabs.org>