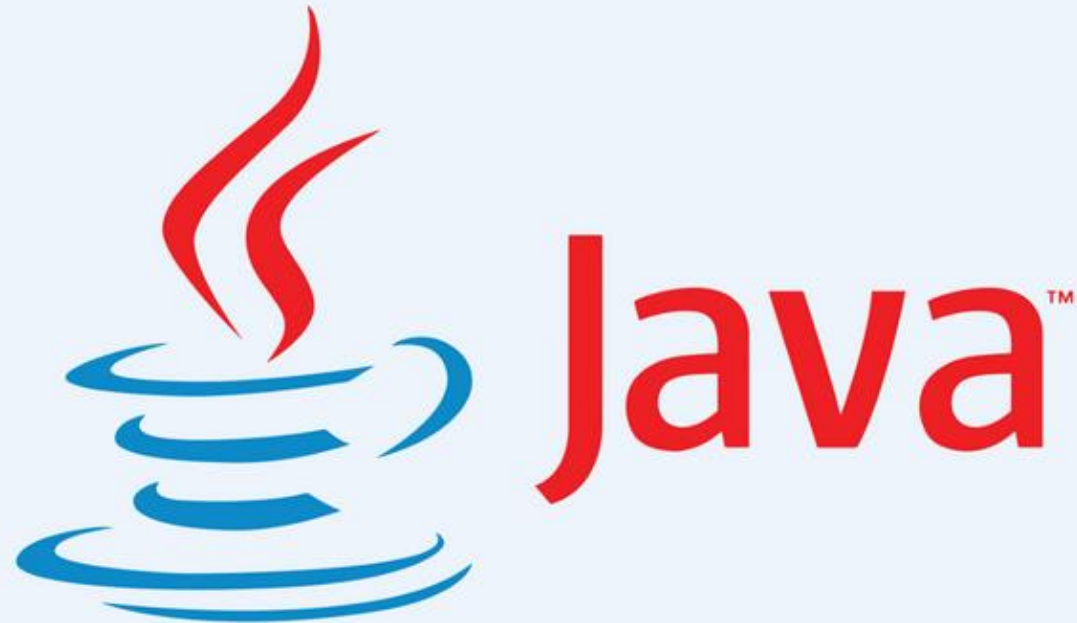




Pemrograman Berorientasi Objek

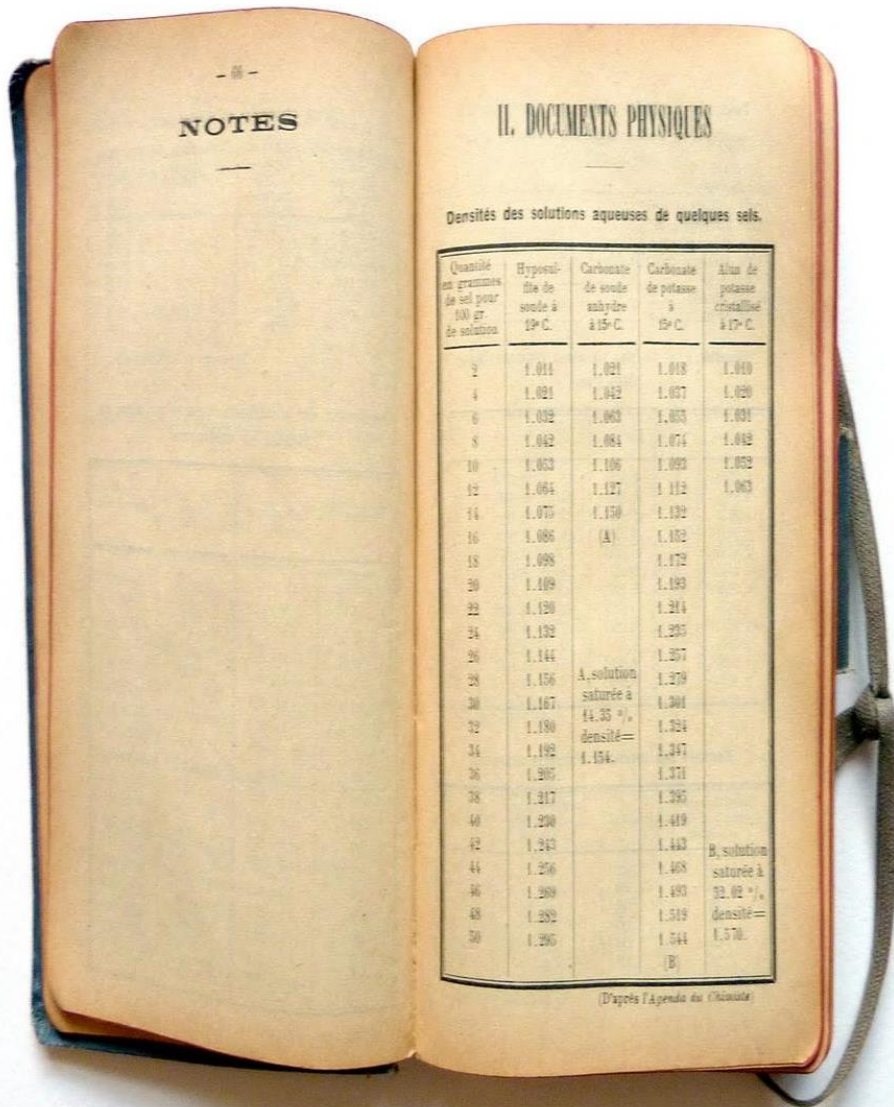


Pertemuan 14

Design Pattern Model - View - Controller

Pemateri : Chrismikha Hardyanto S.Kom., M.Kom.

KONTEN PERKULIAHAN



- Konsep Desain Patern didalam Pemrograman
- Arsitektur Object Model – View - Controller
- Menerapkan MVC didalam pembangunan aplikasi JAVA
- Contoh Aplikasi dengan MVC

Dalam membangun software dengan paradigma PBO, Ada sebuah desain patern yang umum untuk digunakan , yaitu **Arsitektur MVC**

Apa itu Desain Patern?

Merupakan suatu **blueprint** atau **template** yang menunjukkan bagaimana cara **menyelesaikan masalah didalam pembangunan software** yang kemudian dapat digunakan diberbagai situasi yang berbeda-beda saat pengembangan software berlangsung.

Latar belakang (dalam pemrograman) :

- ❑ Proyek pembangunan perangkat lunak merupakan Sesuatu hal yang **besar** dan **kompleks**.
- ❑ programmer membutuhkan suatu **mekanisme** dalam **mengelola** dan **merancang kode** didalam programnya



Desain Patern Didalam PBO

Ada banyak bentuk desain patern yang dapat dimanfaatkan didalam **pembangunan perangkat lunak**. Namun yang kita fokuskan adalah di sisi programming.

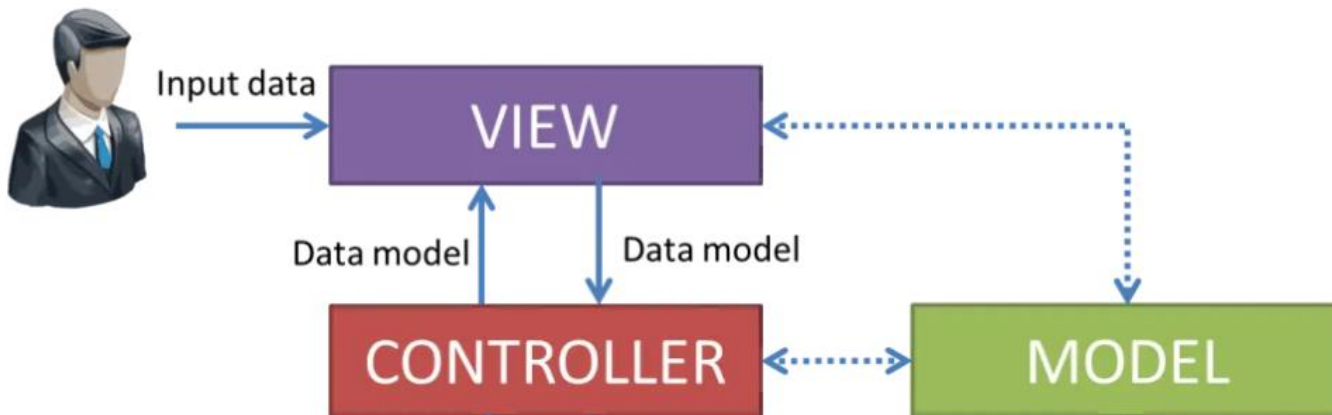
Salat satu desain patern yang umum digunakan didalam pembuatan **program berorientasi Object** adalah :

Arsitektur MVC
[Model – View – Controller]

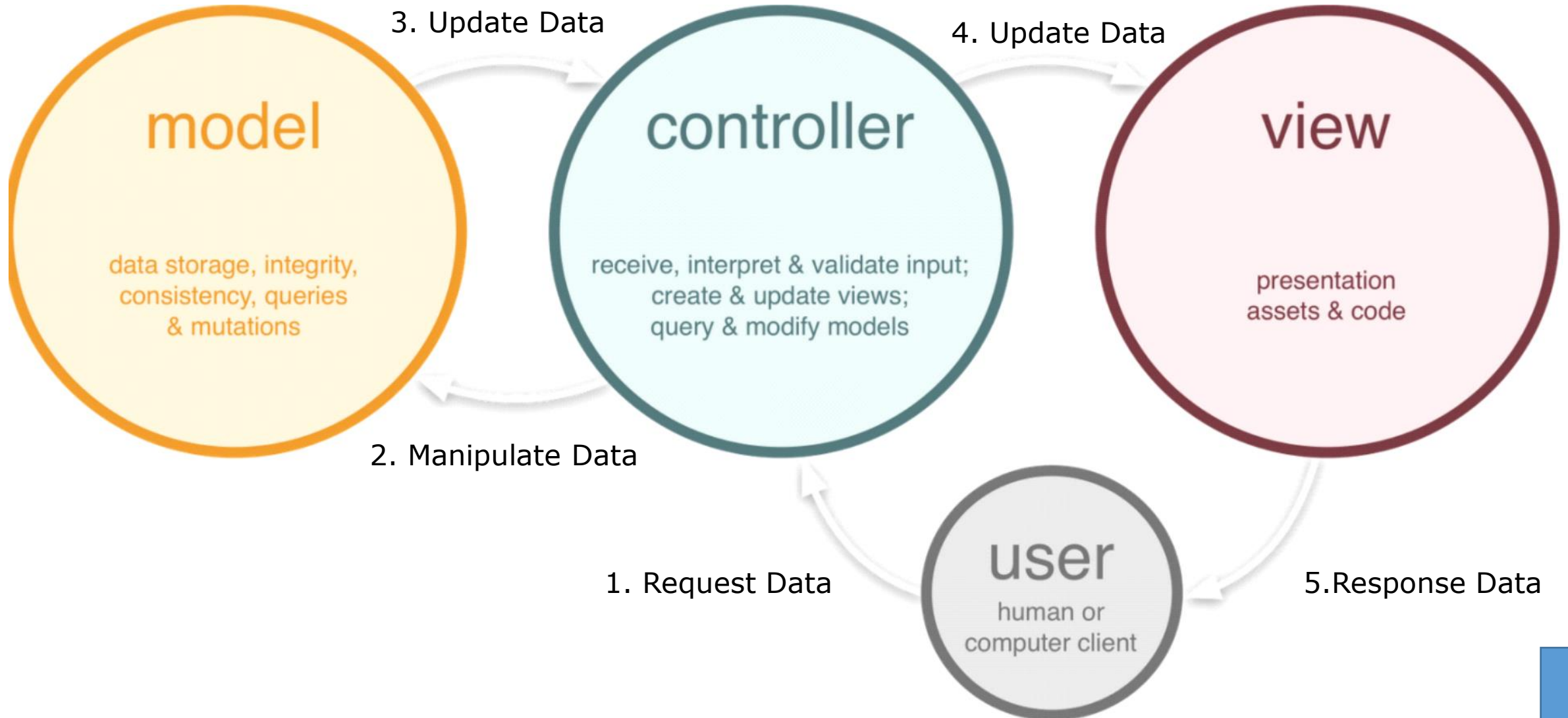


Apa itu Arsitektur Model - View - Controller

- ❑ Konsep **Model - View - Controller (MVC)** merupakan salah satu penerapan compound design pattern dalam pemrograman berorientasi objek.
- ❑ Dimana konsep ini akan **membedakan kelas-kelas** didalam program yang **merepresentasikan data (Model)** dengan kelas-kelas yang **mengelola tampilan program (View)** dan kelas-kelas **pengaturan logik data (Controller)**



Alur data pada Arsitektur MVC



Definisi Model View Controller

❑ Komponen View (Layer View)

Komponen View merupakan kelas-kelas yang **mengimplementasikan user interface** dari program yang dibangun. Bagian inilah yang akan **mengelola tampilan** pada user dan menjadi **media user untuk berinteraksi**. Setiap ada request atau aksi yang diminta oleh user yang berhubungan dengan data akan diteruskan ke bagian controller yang sesuai.

❑ Komponen Controller (Layer Controller)

Komponen controller merupakan kelas-kelas yang akan **mengendalikan alur program secara keseluruhan, mengandung business logic**, dan sebagai **penghubung antara view dengan kelas model**. Kelas ini akan menerima request dan merespon permintaan atau aksi dari view ke kelas Model yang diinginkan.

❑ Komponen Model (Layer Model)

Komponen model merupakan kelas - kelas yang **merepresentasikan POJO (Plain Old Java Object)**, yaitu kelas Java biasa yang lengkap dengan **atribut (Properties) dan method getter-setter** terhadap atribut-atributnya (umumnya private). Biasanya kelas model tidak memiliki method lain selain getter-setter.

Manfaat Arsitektur MVC

Arsitektur MVC memiliki beberapa manfaat dalam pembuatan program antara lain :

- ❑ Proses pengembangan aplikasi menjadi **lebih efisien**
- ❑ Penulisan kode program menjadi **lebih rapi** dan **terstruktur**
- ❑ Dapat melakukan testing program dengan **lebih mudah**
- ❑ Perbaikan bug atau error **lebih cepat untuk diselesaikan**.
- ❑ Mempermudah **pemeliharaan kode program** yang telah complete.
- ❑ **dst**



Mari kita coba praktek...

Setelah memahami konsep dasar yang kita butuhkan tentang MVC. Selanjutnya kita coba untuk membuat program sederhana yang didalamnya terdapat telah menggunakan arsitektur MVC

**Kita akan membuat program JAVA
untuk menyimpan data pelanggan**



Praktikum 1 :

Form Data Pelanggan

Praktikum : Latihan MVC

Untuk latihan praktikum di pertemuan ini, Mari kita buat sebuah aplikasi sederhana yaitu sebuah aplikasi form data pelanggan . Contoh tampilan dari aplikasi adalah sebagai berikut :

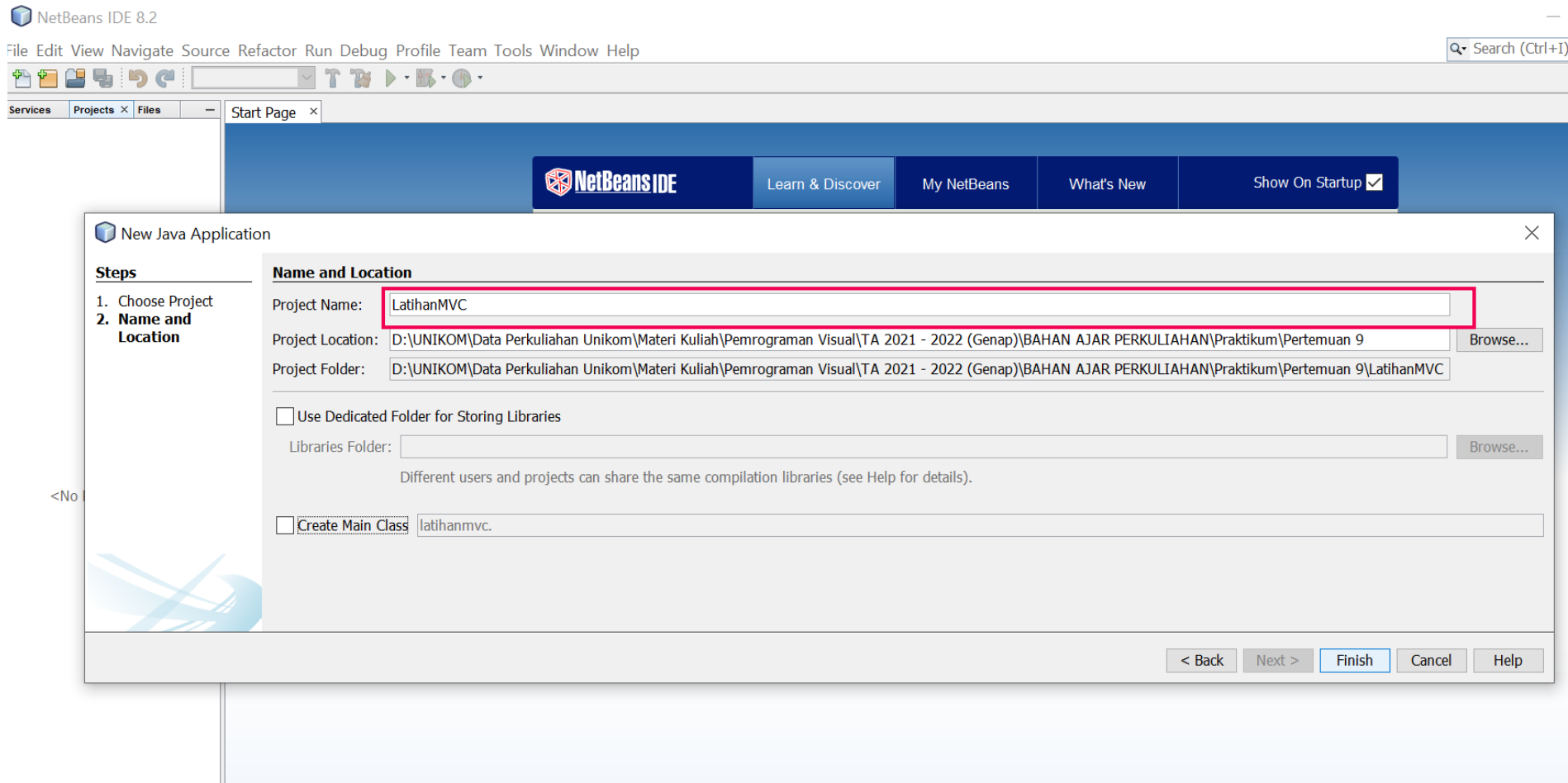


The screenshot shows a web application window with the title bar 'UNIKOM MART APPS'. The main content area has a light blue background and contains the following elements:

- FORM DATA PELANGGAN**
UNIKOM MART
- NAMA :** followed by a text input field.
- EMAIL :** followed by a text input field.
- NO TELEPON :** followed by a text input field.
- ALAMAT :** followed by a larger text input field.
- SIMPAN** button.
- RESET** button.

Praktikum : Latihan MVC

Langkah 1, Buatlah project baru pada netbeans. dengan cara : **File -> New Project-> Java Application -> Tulis nama Project -> unchecklist main class - > finish.** (nama project sesuaikan)

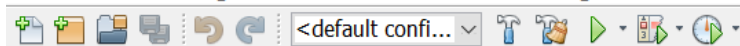


Praktikum : Latihan MVC

Langkah 2, Buatlah **3 buah package** pada project tersebut masing – masing dengan nama :Model, View & Controller. (Gunakan aturan penamaan package seperti biasa)

LatihanMVC - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help



Services Projects Files
LatihanMVC
Source Packages
Libraries

Start Page

New Java Package

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Package Name:

Project:

Location:

Created Folder:

< Back Next > Finish Cancel Help

NetBeans What's New Show On Startup ☒

Tutorials

- Applications
- JavaFX GUI Applications
- Java Web Applications
- Applications
- HTML5 Applications
- Embedded Application...
- Documentation >>

Featured Demo

Error while loading content

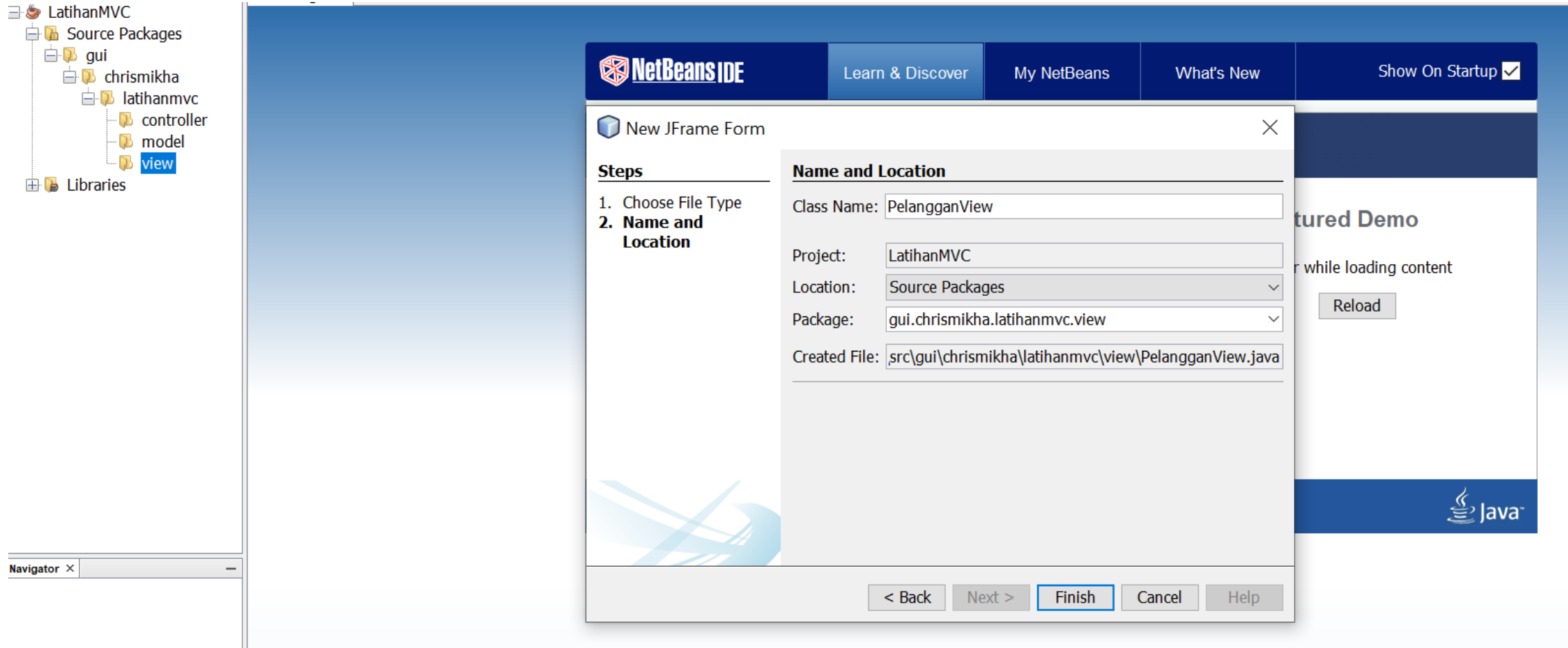
Reload

Java

Navigator

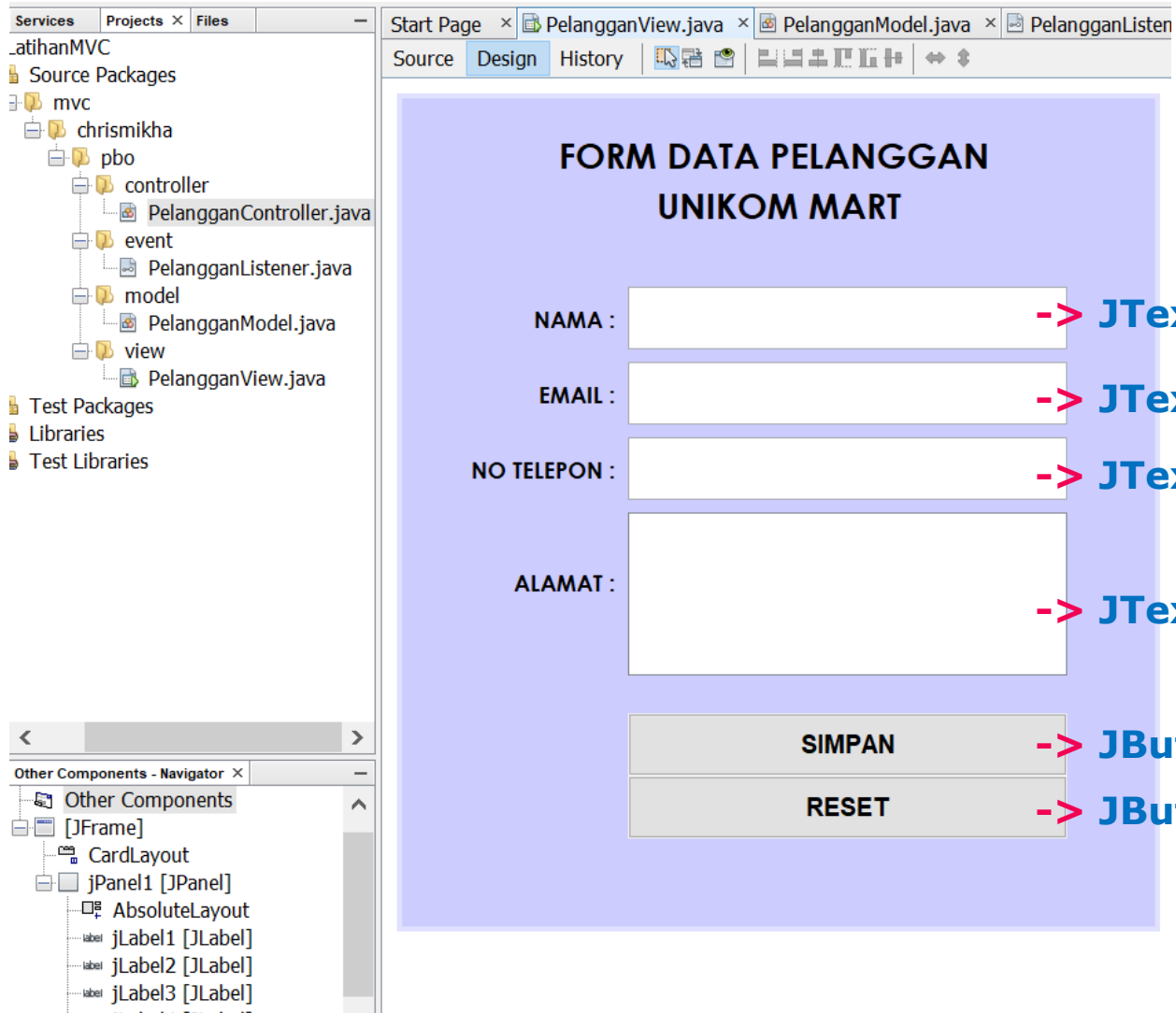
Praktikum : Latihan MVC (Kelas View)

Langkah 3, Untuk langkah pertama mari kita buat sebuah **Class view sebagai tampilan aplikasi**.
Pada package view -> new -> JFrame Form. Beri nama class JFrame sebagai **PelangganView**.



Praktikum : Latihan MVC (Kelas View)

Langkah 4, Aturlah tampilan aplikasi dengan menggunakan component Swing sesuai dengan kebutuhan. Untuk contoh tampilan silakan ikuti contoh berikut :



The screenshot displays an IDE interface for an MVC exercise. The top pane shows the 'Design' view of 'PelangganView.java'. The form is titled 'FORM DATA PELANGGAN UNIKOM MART' and contains the following components:

- NAMA :** -> **TextField, Nama variable : txtNama**
- EMAIL :** -> **TextField, Nama variable : txtEmail**
- NO TELEPON :** -> **TextField, Nama variable : txtNoTelepon**
- ALAMAT :** -> **TextArea, Nama variable :txtAlamat**
- SIMPAN** -> **Button, Nama variable :btnSimpan**
- RESET** -> **Button, Nama variable :btnReset**

The bottom pane shows the 'Other Components - Navigator' with a tree structure of components:

- Other Components
 - JFrame
 - CardLayout
 - jPanel1 [JPanel]
 - AbsoluteLayout
 - jLabel1 [JLabel]
 - jLabel2 [JLabel]
 - jLabel3 [JLabel]

Dimensi Frame, x = 600px , y = 700px
Dimensi Panel, x = 600px , y = 700px

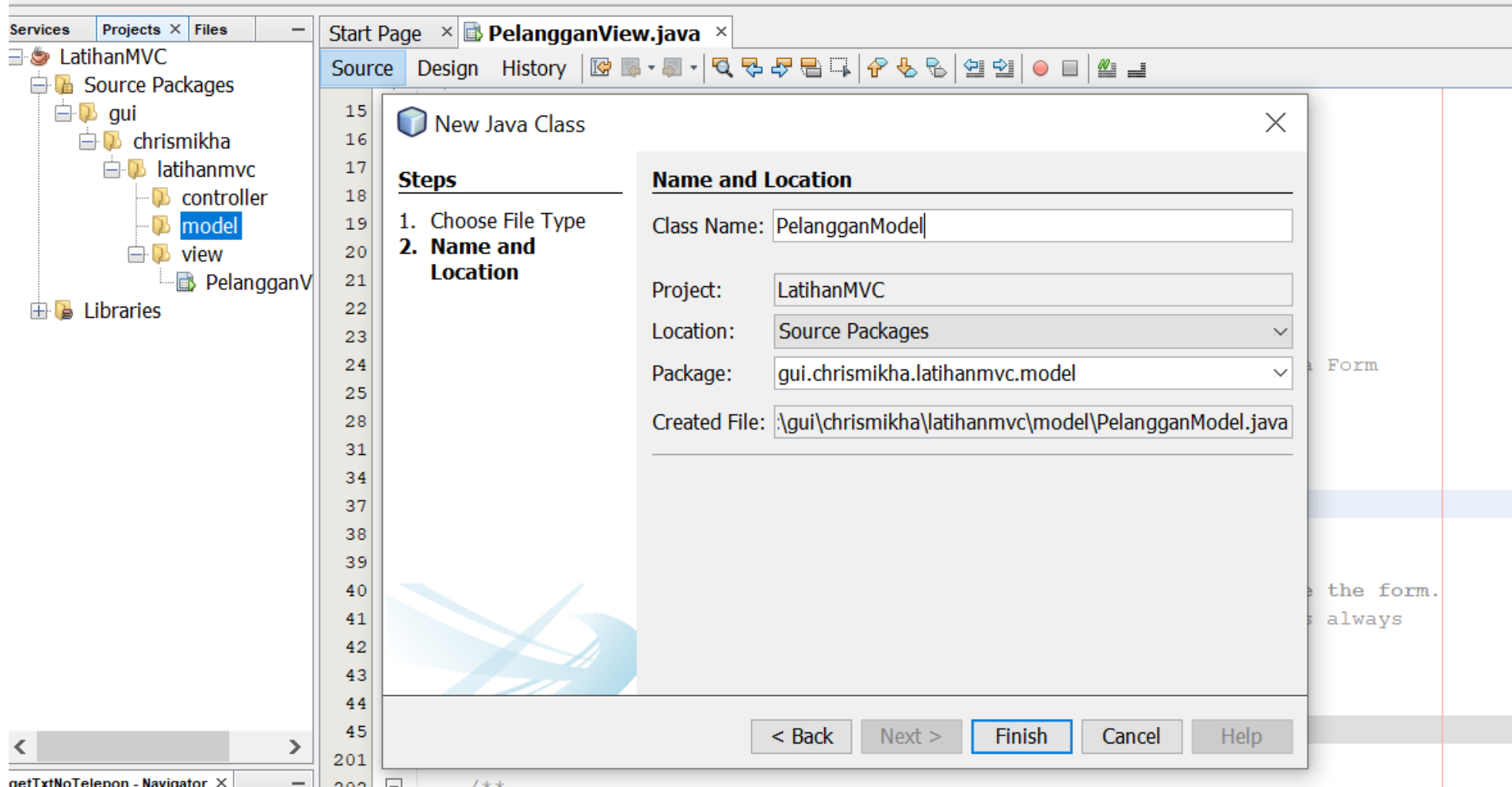
Praktikum : Latihan MVC (Kelas View)

Langkah 5, Masuklah kedalam **mode source**. Untuk setiap field yang nilainya akan disimpan (untuk kasus ini ada 4 object), buatlah **method getter** nya. Method Getter() akan digunakan sebagai mekanisme untuk mengambil nilai dari setiap component input data pada form.

```
15 public class PelangganView extends javax.swing.JFrame {
16
17     /**
18      * Creates new form PelangganView
19      */
20     public PelangganView() {
21         initComponents();
22     }
23
24     //Deklarasi Method Get untuk setiap nilai yang akan digunakan pada Form
25     public JTextArea getTxtAlamat() {...3 lines }
28     public JTextField getTxtEmail() {...3 lines }
31     public JTextField getTxtNama() {...3 lines }
34     public JTextField getTxtNoTelepon() {...3 lines }
37
```

Praktikum : Latihan MVC (Kelas Model)

Langkah 6, Selanjutnya mari kita tambahkan sebuah **Class sebagai model dari form pelanggan**.
Pada package model -> new -> Java Class. Beri nama class sebagai **PelangganModel**



Praktikum : Latihan MVC (Kelas Model)

Langkah 7, Didalam Class PelangganModel, buat atribut untuk menampung nilai yang akan disimpan pada FORM (**ada 4 atribut**). Gunakan aturan penulisan Atribut dengan kaidah PBO yang baik

```
1  package gui.chrismikha.latihanmvc.model;
2
3  public class PelangganModel {
4
5      //Deklarasi Atribut yang digunakan untuk menampung data dari FORM Pelanggan
6      private String nama;
7      private String email;
8      private String noTelepon;
9      private String Alamat;
10
11      //Deklarasi Getter dan Setter dari setiap atribut pada Class PelangganModel
12      public String getNama() {...3 lines }
13
14      public void setNama(String nama) {...3 lines }
15
16
17
18      public String getEmail() {...3 lines }
19
20      public void setEmail(String email) {...3 lines }
21
22
23
24
25      public String getNoTelepon() {...3 lines }
26
27      public void setNoTelepon(String noTelepon) {...3 lines }
28
29
30
31
32      public String getAlamat() {...3 lines }
33
34      public void setAlamat(String Alamat) {...3 lines }
35
36
37
38
39
40 }
41
```

Praktikum : Latihan MVC (Kelas Model)

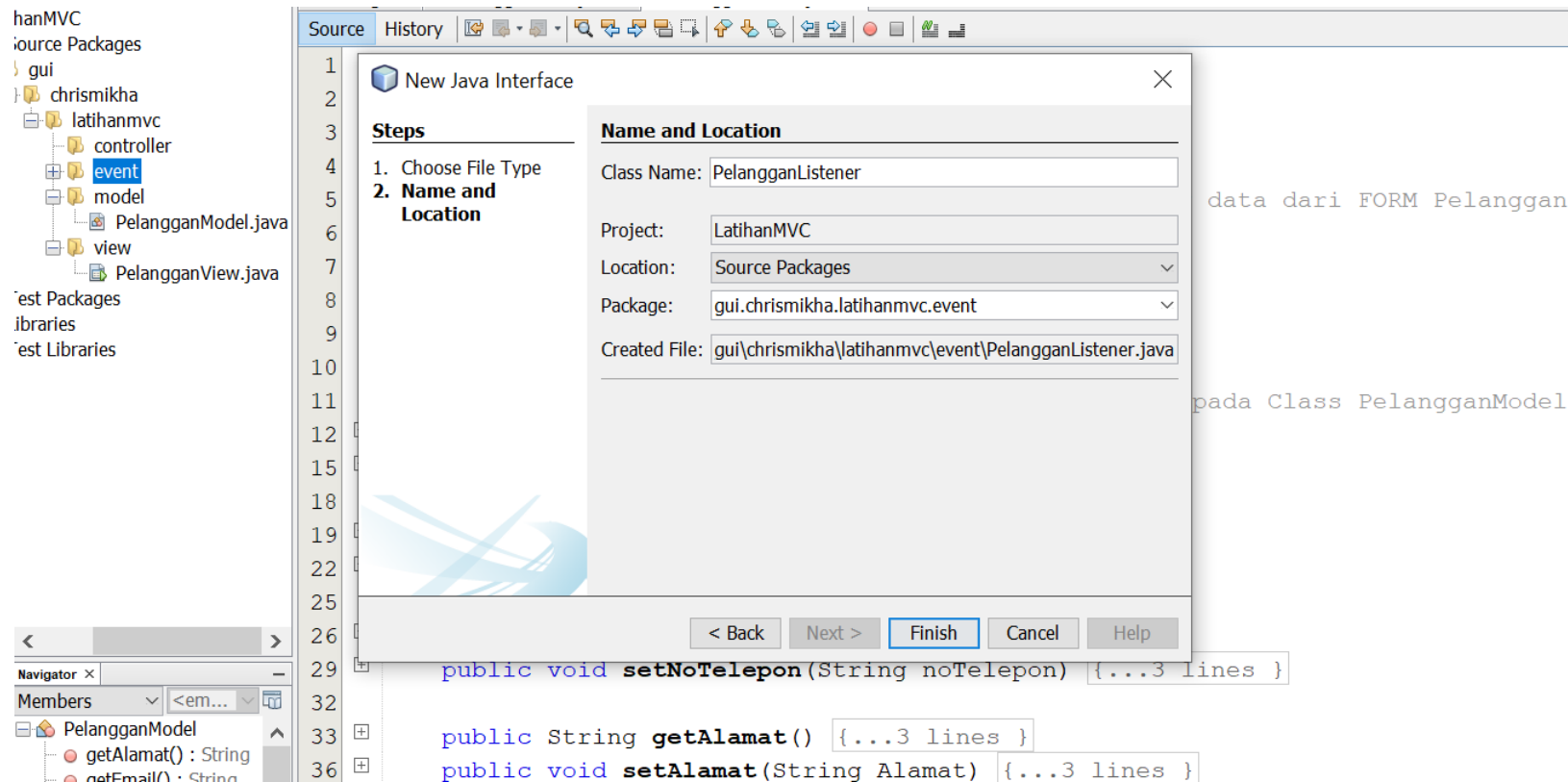
Langkah 8, Didalam Class PelangganModel, buat juga method untuk menangani aksi yang terjadi ketika button simpan dan button reset diclick. Contoh Kodanya sebagai berikut :

```
40 public void simpanForm() {  
41     //Kode untuk membuat aksi ketika button simpan di click  
42 }  
43  
44 public void resetForm() {  
45     //Kode untuk membuat aksi ketika button reset di click  
46 }  
47
```

*Untuk sekarang , kita kosongkan dulu isi dari 2 method diatas. Method diatas akan berisi suatu aksi untuk **melakukan proses pada data didalam model (sesuai dengan kebutuhan program)**

Praktikum : Latihan MVC (Kelas Model)

Langkah 9, Selanjutnya kita perlu menangani apabila terjadi perubahan state pada model (terjadi perubahan data), yang menyebabkan tampilan di view pun akan ikut berubah (terjadi update) . Agar dapat terjadi **komunikasi antara model dan view**, kita perlu membuat **event**.



*Buatlah package baru dengan nama event. Lalu untuk **mengimplementasikan event** didalam sebuah kelas, kita buat sebuah interface dengan nama **PelangganListener**

Praktikum : Latihan MVC (Kelas Model)

Langkah 10, Tambahkan sebuah abstract method dalam interface PelangganModel dengan nama **onChange()** yang berisi parameter **Object Dari PelangganModel**.

Method ini akan digunakan untuk menangani komunikasi ke class view apabila terjadi perubahan data pada model

```
1 package gui.chrismikha.latihanmvc.event;  
2 import gui.chrismikha.latihanmvc.model.PelangganModel;  
3  
4 public interface PelangganListener {  
5  
6     //Method untuk handle apabila terjadi perubahan pada model  
7     void onChange(PelangganModel pelanggan);  
8  
9 }  
10
```

*Tambahkan package baru dengan nama event kedalam project. Untuk **mengimplementasikan event** didalam kelas, buatlah sebuah interface dengan nama **PelangganListener**

Praktikum : Latihan MVC (Kelas Model)

Langkah 11, Selanjutnya tambahkan object PelangganListener pada Class Model sebagai atribut. Definisikan juga method Getter & Setternya (dengan akses private)

```
12 //Deklarasikan Listener yang telah dibuat sebaga atribut pada Class Model
13 private PelangganListener pelangganListener;
14
15 public PelangganListener getPelangganListener() {
16     return pelangganListener;
17 }
18
19 public void setPelangganListener(PelangganListener pelangganListener) {
20     this.pelangganListener = pelangganListener;
21 }
22
```

Praktikum : Latihan MVC (Kelas Model)

Langkah 12, Untuk menembak object event apabila terjadi perubahan (memberitahu jika ada event) Buatlah method dengan nama **fireOnChange()** untuk mengecek apakah object listener kosong (bernilai null) atau tidak.

```
52      /*Method untuk memberikan informasi
53      apabila terjadi perubahan data didalam model */
54      protected void fireOnChange() {
55          //Validasi jika object pelangganListener tidak kosong
56          if (pelangganListener != null) {
57              pelangganListener.onChange(this);
58          }
59      }
```

*Validasi dilakukan juga untuk handle kemungkinan **terjadinya error** didalam program ketika user melakukan komunikasi pertama saat **mengakses interface PelangganListener**. Hal ini disebabkan karena nilai atribut pertama selalu kosong (null)

*Pasang method fireOnChange() **pada seluruh setter** didalam class PelangganModel

Praktikum : Latihan MVC (Kelas Model)

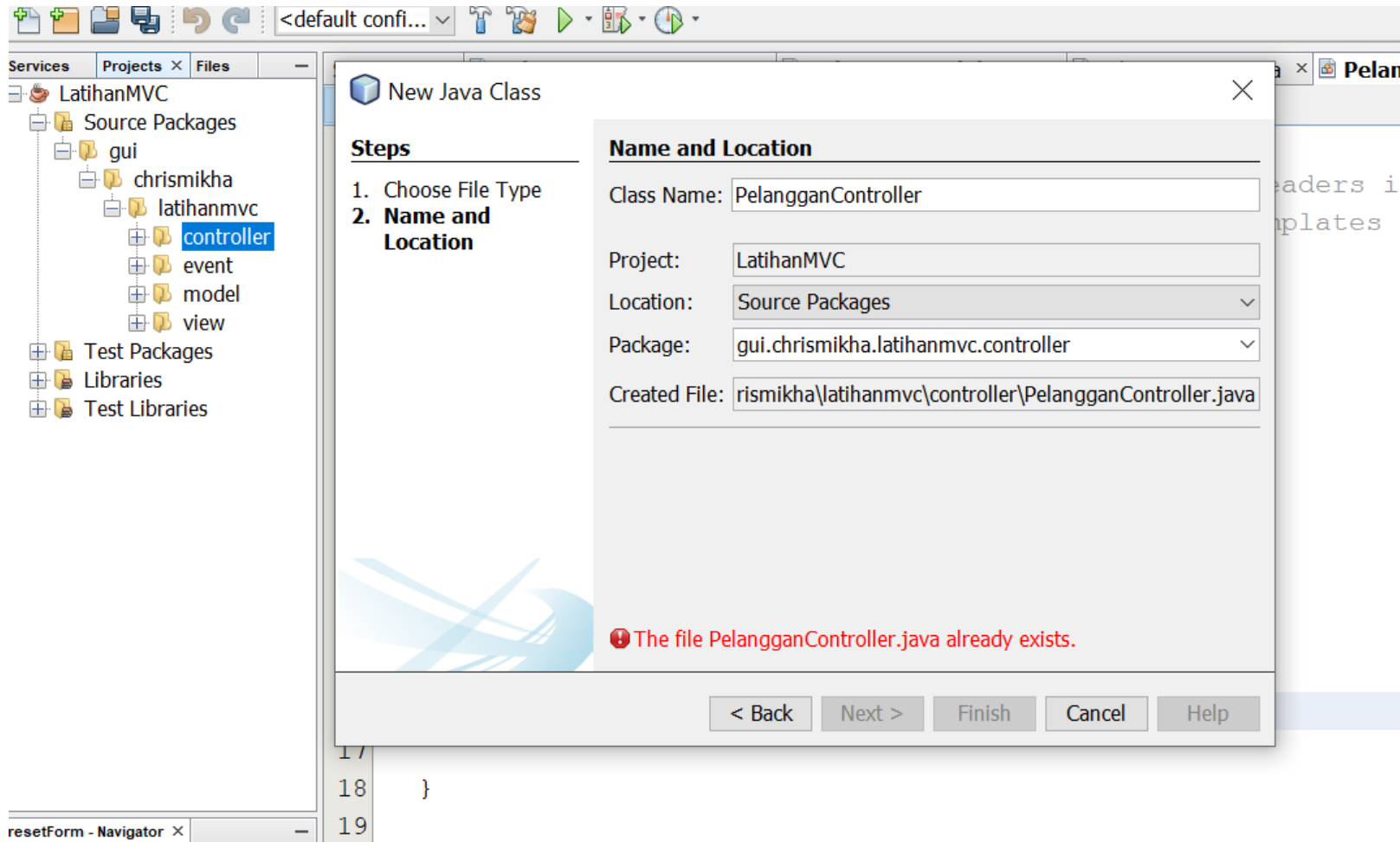
Langkah 13, Isilah **method simpanForm()** dan **resetForm()** pada Class PelangganModel dengan **aksi yang diinginkan**. Misalkan resetForm akan menghapus semua value dan simpanForm menampilkan kotak dialog yang memberikan info data berhasil disimpan.

Contoh kode program sebagai berikut :

```
public void simpanForm() {  
    //Kode untuk membuat aksi ketika button simpan di click  
    JOptionPane.showMessageDialog(null, "Data Pelanggan Berhasil Disimpan");  
}  
  
public void resetForm() {  
    //Kode untuk membuat aksi ketika button reset di click  
    this.setNama("");  
    this.setEmail("");  
    this.setEmail("");  
    this.setAlamat("");  
}
```

Praktikum : Latihan MVC (Kelas Controller)

Langkah 14, langkah selanjutnya tambahkan sebuah **Class sebagai controller dari form pelanggan**. **Pada package model -> new -> Java Class**. Beri nama class sebagai **PelangganController**



Praktikum : Latihan MVC (Kelas Controller)

Langkah 15, Didalam class PelangganController kita akan membuat **method** untuk handle **komunikasi antara view dengan model** sesuai dengan proses/aksi yang ada pada view. Buatlah kode program seperti contoh berikut :

```
1  package gui.chrismikha.latihanmvc.controller;
2  import gui.chrismikha.latihanmvc.model.PelangganModel;
3  import gui.chrismikha.latihanmvc.view.PelangganView;
4
5  public class PelangganController {
6
7      private PelangganModel model;
8
9      public PelangganModel getModel() {...3 lines }
12     public void setModel(PelangganModel model) {...3 lines }
15
16     //Method pada CClass Controller untuk handle komunikasi dari view ke Model
17     public void simpanForm(PelangganView view){
18
19     }
20
21     public void resetForm(PelangganView view){
22
23     }
24 }
```

Praktikum : Latihan MVC (Kelas Controller)

Langkah 16, isilah **method resetForm()** pada controller untuk melakukan aksi **menghapus nilai dari seluruh field** dengan **validasi** aksi tersebut baru dieksekusi **jika seluruh field pada form sudah terisi**. Contoh kode program sebagai berikut :

```
public void resetForm(PelangganView view){
    //Ambil setiap nilai pada field di class view
    String nama = view.getTxtNama().getText();
    String email = view.getTxtEmail().getText();
    String noTelepon = view.getTxtNoTelepon().getText();
    String alamat = view.getTxtAlamat().getText();

    //Buat validasi untuk mengecek apakah seluruh field telah terisi
    if(nama.equals("") && email.equals("") && noTelepon.equals("") && alamat.equals("")){
        //Do Nothing
    } else
    {
        //proses reset dilakukan dengan memanggil method dari model
        model.resetForm();
    }
}
```

Praktikum : Latihan MVC (Kelas Controller)

Langkah 17, lakukan hal yang sama pada method `simpanForm`. Buatlah aksi untuk **menghandle komunikasi data** dari view kedalam model. Berikan juga validasi sesuai kebutuhan program. Contoh kode sebagai berikut :

```
18 public void simpanForm(PelangganView view){
19     //Ambil setiap nilai pada field di class view
20     String nama = view.getTxtNama().getText();
21     String email = view.getTxtEmail().getText();
22     String noTelepon = view.getTxtNoTelepon().getText();
23     String alamat = view.getTxtAlamat().getText();
24
25     //Buat validasi untuk mengecek seluruh field pada form wajib terisi
26     if(nama.trim().equals("")){
27         JOptionPane.showMessageDialog(view, "Nama Masih Kosong");
28     } else if(email.trim().equals("")){
29         JOptionPane.showMessageDialog(view, "Email Masih Kosong");
30     } else if(noTelepon.trim().equals("")){
31         JOptionPane.showMessageDialog(view, "No Telepon Masih Kosong");
32     } else if(alamat.trim().equals("")){
33         JOptionPane.showMessageDialog(view, "Alamat Masih Kosong");
34     } else {
35         model.simpanForm();
36     }
37 }
```

Praktikum : Latihan MVC (Kelas View)

Langkah 18, Langkah terakhir adalah kita membuat mekanisme untuk handle komunikasi yang terjadi antara view dengan controller & model. Kita Deklarasikan class Model dan class controller di class View

```
16 public class PelangganView extends javax.swing.JFrame {
17
18     /**
19      * Creates new form PelangganView
20      */
21     //Deklarasikan Atribut untuk Class Model dan Controller
22     private PelangganModel model;
23     private PelangganController controller;
24
25     public PelangganView() {
26         initComponents();
27     }
28
29     //Deklarasi Method Get untuk setiap nilai yang akan digunakan pada Form
30     public JTextArea getTxtAlamat() {...3 lines }
33     public JTextField getTxtEmail() {...3 lines }
36     public JTextField getTxtNama() {...3 lines }
39     public JTextField getTxtNoTelepon() {
40         return txtNoTelepon;
41     }
```


Praktikum : Latihan MVC (Kelas View)

Langkah 19, implementasikan interface `PelangganListener` di class `PelangganView`. Kemudian **lakukan override pada method `onChange`** sesuai dengan kebutuhan program. Contoh code sebagai berikut :

```
17 public class PelangganView extends javax.swing.JFrame implements PelangganListe
18
19 /** Creates new form PelangganView ...3 lines */
22 //Deklarasikan Atribut untuk Class Model dan Controller
23 private PelangganModel model;
24 private PelangganController controller;
25
26 public PelangganView() {...3 lines }
29
30 //Deklarasi Method Get untuk setiap nilai yang akan digunakan pada Form
31 public JTextArea getTxtAlamat() {...3 lines }
34 public JTextField getTxtEmail() {...3 lines }
37 public JTextField getTxtNama() {...3 lines }
40 public JTextField getTxtNoTelepon() {...3 lines }
43
44 //Method dari interface PelangganListener
45 @Override
46 public void onChange(PelangganModel pelanggan) {
47     txtNama.setText(model.getNama());
48     txtEmail.setText(model.getEmail());
49     txtNoTelepon.setText(model.getNoTelepon());
50     txtAlamat.setText(model.getAlamat());
51 }
```

Praktikum : Latihan MVC (Kelas View)

Langkah 20, Selanjutnya kita lakukan **instansiasi object untuk class model dan controller** didalam view sehingga kode didalam class tersebut dapat dimanfaatkan. Buat juga mekanisme untuk memberikan **object sebagai argument** kedalam listener dan controller

```
26 public PelangganView() {  
27     //Instansiasi Object dari class PelangganModel & PelangganController  
28     model = new PelangganModel();  
29     controller = new PelangganController();  
30  
31     //Set object dari model pada listener dan controller  
32     model.setPelangganListener(this);  
33     controller.setModel(model);  
34  
35     initComponents();  
36 }  
37
```

*Buatlah object didalam **constructor dari class PelangganView** dan letakan kodenya diatas method initComponents()


Praktikum : Latihan MVC (Kelas View)

Langkah 21, tambahkan **event onClick** pada component button untuk simpan dan reset didalam Form. Lalu isi kodenya dengan **memanggil method dari contruktor (PelangganConstruktor)**.

```
228
229 private void btnSimpanMouseClicked(java.awt.event.MouseEvent evt) {
230     // TODO add your handling code here:
231     controller.simpanForm(this);
232 }
233
234 private void btnResetMouseClicked(java.awt.event.MouseEvent evt) {
235     // TODO add your handling code here:
236     controller.resetForm(this);
237 }
238
```

Praktikum : Latihan MVC (Kelas View)

Langkah 22, Jalankan program (run) , dan coba kita uji apakah program dapat berjalan sesuai dengan keinginan kita tau tidak.



UNIKOM MART APPS

FORM DATA PELANGGAN UNIKOM MART

NAMA :

EMAIL :

NO TELEPON :

ALAMAT :


SIMPAN

RESET

Message

i FORM MASIH KOSONG

OK



UNIKOM MART APPS

FORM DATA PELANGGAN UNIKOM MART

NAMA :

EMAIL :

NO TELEPON :

ALAMAT :

SIMPAN

RESET

Message

i DATA PELANGGAN BERHASIL DISIMPAN

OK

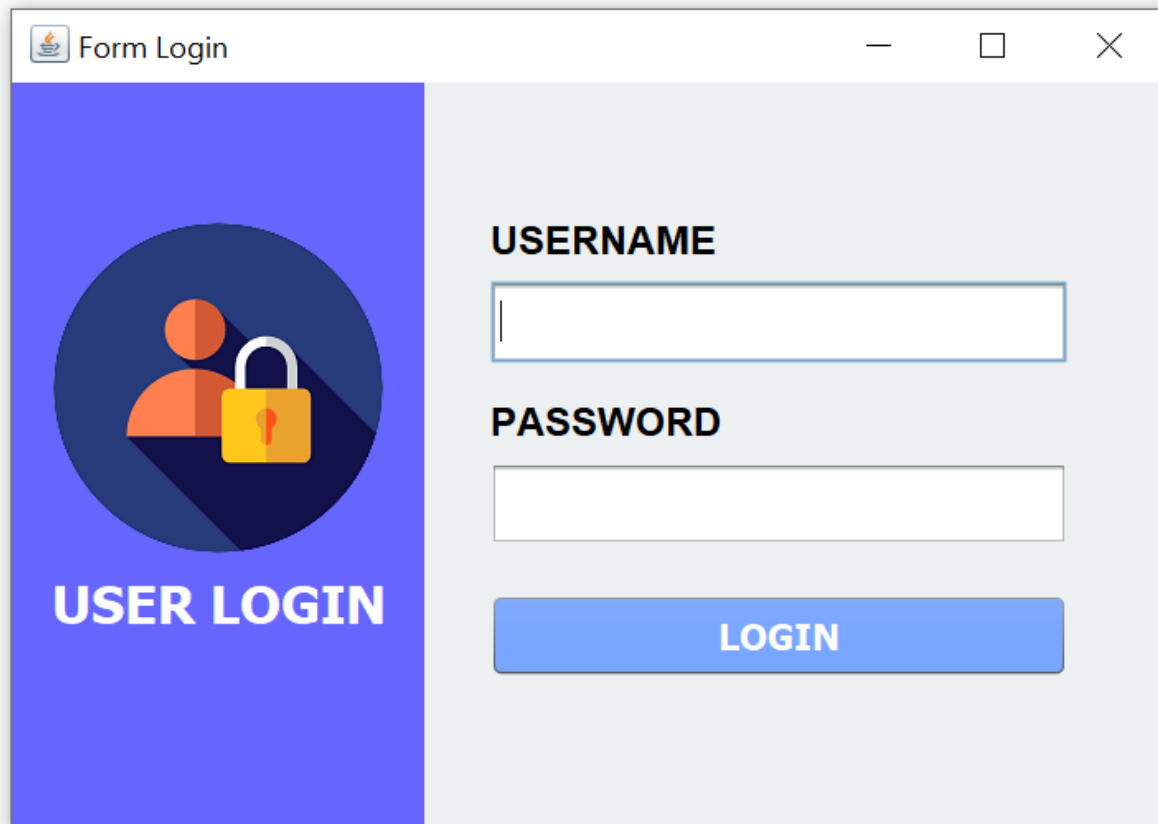
Praktikum 2 :

Membuat Form Login

Latihan Praktikum ini akan dinilai sebagai tugas ke-2 PBO

Praktikum 2: Form Login

Untuk latihan praktikum ke-2 di pertemuan ini, Mari kita buat sebuah aplikasi untuk menangani **proses login dari user.**



The screenshot shows a window titled "Form Login". On the left, there is a blue vertical panel with a circular icon containing an orange person silhouette and a yellow padlock, with the text "USER LOGIN" below it. The main area is light gray and contains two input fields: "USERNAME" and "PASSWORD". Below these fields is a blue button labeled "LOGIN".



The screenshot shows a window titled "Latihan Form Login". The top bar is light gray with the word "DASHBOARD" on the left and "Halo, Admin" with a user icon on the right. The main area is split into a blue vertical sidebar on the left and a large white area on the right.

Praktikum 2: Form Login

Ketentuan Pengerjaan :

- ❑ Buatlah project dengan nama **Tugas2**. Simpan didalam folder Praktikum Anda
- ❑ Buatlah program pada Tugas2 dengan menggunakan **konsep MVC** didalam perancangan coding Anda. (Buatlah class yang sesuai dengan kebutuhan Anda)
- ❑ Tampilan Program dibuat **berbasis GUI**. Anda bebas untuk merancang tampilan dari program yang anda buat . Contoh yg saya berikan dapat digunakan sebagai panduan.
- ❑ **username** dan **password** yang dibutuhkan untuk validasi login ditanam didalam kode program Anda (statis)
- ❑ Apabila Login **berhasil**, tampilkan **kotak dialog** yang menampilkan pesan **“LOGIN BERHASIL”**. Kemudian program akan berganti menjadi **jendela dashboard** (lihat contoh)
- ❑ Apabila Login gagal, tampilkan kotak dialog yang menampilkan pesan **“USERNAME ATAU PASSWORD SALAH”**

TERIMA KASIH