MODUL 12
POLA DESAIN SINGLETON

12.1 Tujuan Materi Pembelajaran

1. Memahami konsep dasar design pattern sebagai solusi umum terhadap permasalahan rekayasa perangkat lunak.
2. Menjelaskan pentingnya penggunaan design pattern dalam meningkatkan kualitas desain perangkat lunak, khususnya dari sisi maintainability, scalability, dan reusability.
3. Mengidentifikasi karakteristik utama dari Singleton Pattern sebagai salah satu bentuk creational design pattern.
4. Menguji dan menganalisis hasil implementasi Singleton Pattern untuk memastikan hanya satu instance dari kelas yang dibuat selama siklus hidup program.
5. Membandingkan penerapan Singleton Pattern dengan pendekatan tanpa design pattern untuk memahami manfaatnya.

12.2 Materi Pembelajaran

12.2.1. Pengenalan Design Pattern

Design Pattern (Pola Desain) adalah solusi generik terhadap masalah-masalah berulang dalam desain perangkat lunak. Design pattern bisa dianggap sebagai blueprint siap pakai yang dapat disesuaikan untuk menyelesaikan masalah desain dalam proses pemrograman (Sarcar, 2022). Penting untuk dipahami bahwa design pattern berbeda dengan algoritma. Algoritma adalah serangkaian langkah konkret untuk memecahkan masalah spesifik (misalnya, algoritma sorting). Sedangkan design pattern adalah konsep abstrak yang membantu merancang struktur arsitektur program dengan lebih baik.

Tidak ada penemu tunggal dari design pattern. Pola-pola ini muncul secara alami dari pengalaman para pengembang perangkat lunak dalam mengatasi masalah berulang saat mengerjakan proyek. Ketika solusi ini mulai sering digunakan, orang mulai memberi nama dan mendokumentasikannya. Namun pada tahun 1994, sebuah buku berjudul *Design Patterns: Elements of Reusable Object-Oriented Software*, yang ditulis oleh Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four) menjelaskan dan mengelompokan pola-pola desain yang sering digunkan pada pengembangan sofware. Saat ini design pattern kini juga diterapkan di luar OOP, seperti di pemrograman fungsional dan pengembangan arsitektur microservices.

Mengapa Design Pattern Penting? Ini karena design pattern sudah terbukti efektif dalam mengatasi masalah desain umum. Penggunaan pola desain memberikan terminologi standar antar developer, sehingga diskusi dan dokumentasi menjadi lebih efisien. Dengan memahami pola desain, developer junior dapat membuat keputusan desain sekelas developer senior. Namun hal yang lain yang perlu dipahami adalah design pattern bukan solusi untuk semua masalah pengembangan software (silver bullet).Tidak semua masalah harus dipecahkan menggunakan design pattern. Kadang justru menambah kompleksitas

yang tidak perlu (over engineering). Beberapa pemula menggunakan pola desain secara membabi buta tanpa memahami konteks. Oleh karena itu prinsip yang harus diingat

"Jika Anda memegang palu, semua benda terlihat seperti paku."

Sehingga Anda harus mampu menggunakannya dengan bijak dengan memahami masalahnya dahulu, baru putuskan apakah perlu menggunakan pattern tertentu.

Menurut Gang of Four, design pattern diklasifikasikan menjadi tiga kategori utama:

1. **Creational Patterns** : Mengelola cara pembuatan objek untuk meningkatkan fleksibilitas dan penggunaan ulang kode. Contoh : Singleton, Factory Method, Builder, Prototype, Dependency Injection
2. **Structural Patterns** : Mengatur komposisi kelas dan objek menjadi struktur yang besar dan efisien. Contoh : Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
3. **Behavioral Patterns** : Mengatur interaksi dan komunikasi antar objek serta pengalokasian tanggung jawab. Contoh : Observer, Chain of Responsibility, Command, Iterator, Mediator, State, Strategy, Template Method, Visitor.

12.2.2. Singleton pattern

Singleton Pattern adalah pola desain dari kategori Creational Pattern yang bertujuan untuk membatasi pembuatan objek dari suatu kelas sehingga hanya satu instance saja yang boleh ada selama siklus hidup program. Dengan kata lain, kelas yang menerapkan Singleton Pattern akan membuat constructor private sehingga objek tidak bisa diinstansiasi dari luar kelas. Kemudian class singleton akan menyediakan metode publik statis (biasanya getInstance()) untuk mengakses satu-satunya instance.

Situasi singleton pattern diperlukan saat hanya dibutuhkan satu instance object yang digunakan sebagai pengelolaan sumber daya, misalnya koneksi database, logger, pengelola konfigurasi. Semua bagian program dapat menggunakan instance yang sama tanpa membuat objek baru. Dengan demikian maka pengengelolaan sumber daya aplikasi dilakukan secara terpusat. Struktur umum singleton adalah sebagai berikut :

1. Private static instance: Menyimpan satu-satunya instance kelas.
2. Private constructor: Mencegah pembuatan instance baru dari luar kelas.
3. Public static method: Mengembalikan instance yang sudah ada, atau membuatnya jika belum ada.

Terdapat 4 variasi implementasi Singleton yang dapat diimplementasikan:

1. **Eager Initialization**

   Pada pendekatan ini, instance singleton langsung dibuat saat kelas dimuat, tidak menunggu dipanggil. Kelebihan dari variasi ini adalah implementasinya yang sederhana dan otomatis thread-safe (karena JVM menjamin inisialisasi kelas aman). Sedangkan kekurangannya adalah jika instance memerlukan resource besar dan ternyata tidak digunakan, maka akan ada pemborosan memori. Contoh implementasinya adalah sebagai berikut :

```
public class EagerSingleton {
    private static final EagerSingleton instance = new
EagerSingleton();
```

```
    private EagerSingleton() {
        System.out.println("Instance created at class loading
time.");
    }

    public static EagerSingleton getInstance() {
        return instance;
    }
}
```

2. **Thread-safe Singleton (Lazy Initialization dengan Synchronized)**
   Menggunakan lazy initialization, tapi menambahkan synchronized pada metode getInstance() supaya aman dari masalah thread-race. Kelebihan pendekatan ini ialah hanya membuat instance hanya saat dibutuhkan. Selain itu penggunaan kata kunci Synchronized membuat instance obyek aman di lingkungan multi-threaded. Namun pendekan ini memilki kukurangan yaitu adanya kecendrungan penurunan performa karena kata kunci synchronized membuat getInstance() lambat jika sering dipanggil. Contoh implementasinya adalah sebagai berikut :

```
public class ThreadSafeSingleton {
    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton() {
        System.out.println("Thread-safe Singleton created.");
    }

    public static synchronized ThreadSafeSingleton getInstance() {
        if (instance == null) {
            instance = new ThreadSafeSingleton();
        }
        return instance;
    }
}
```

3. **Double Checked Locking**
   Versi singleton ini mendeklarasikan sebuah instance obyek dengan kata kunci volatile untuk memastikan bahwa beberapa thread mengakses instance obyek dengan benar ketika sedang diinisialisasi ke instance Singleton. Metode ini secara drastis mengurangi overhead pemanggilan method synchronized. Contoh implementasinya adalah sebagai berikut :

```
public class DoubleCheckedLockingSingleton {
    private static volatile DoubleCheckedLockingSingleton instance;

    private DoubleCheckedLockingSingleton() {
        System.out.println("Double Checked Locking Singleton
created.");
```

```
    }

    public static DoubleCheckedLockingSingleton getInstance() {
        if (instance == null) {
            synchronized (DoubleCheckedLockingSingleton.class) {
                if (instance == null) {
                    instance = new DoubleCheckedLockingSingleton();
                }
            }
        }
        return instance;
    }
}
```

4. Bill Pugh Singleton

Versi singleton ini memanfaatkan inner static helper class untuk lazy initialization tanpa perlu synchronized. Kelebihan dari versi ini adalah lebih efisien karena tidak ada penalti kinerja dari synchronized, sehingga instance dibuat hanya saat diperlukan dan Thread-safe secara alami berkat mekanisme pemuatan kelas dari JVM. Contoh implementasinya adalah sebagai berikut :

```
public class BillPughSingleton {
    private BillPughSingleton() {
        System.out.println("Bill Pugh Singleton created.");
    }

    private static class SingletonHelper {
        private static final BillPughSingleton INSTANCE = new
BillPughSingleton();
    }

    public static BillPughSingleton getInstance() {
        return SingletonHelper.INSTANCE;
    }
}
```

12.3 Rangkuman Materi

1. Design Pattern adalah solusi umum untuk masalah desain perangkat lunak berulang.
2. Bukan berupa algoritma konkret, melainkan konsep abstrak untuk membantu merancang arsitektur sistem yang lebih baik.
3. Sejarah Design Pattern dimulai dari pengalaman praktisi OOP yang terdokumentasi dalam buku Design Patterns: Elements of Reusable Object-Oriented Software (1994) oleh Gang of Four.
4. Manfaat Design Pattern:
    a. Mempercepat pengembangan dengan solusi yang telah teruji.
    b. Menjadi bahasa komunikasi standar antar developer.

    c. Membantu pemula membuat desain yang profesional.
5. Kritik terhadap Design Pattern:
    a. Tidak selalu diperlukan dalam semua proyek.
    b. Penggunaan berlebihan bisa menyebabkan overengineering.
    c. Penting untuk memahami konteks sebelum menerapkan pola.
6. Kategori Design Pattern:
    a. Creational (membantu membuat objek, contoh: Singleton).
    b. Structural (mengatur struktur objek, contoh: Adapter).
    c. Behavioral (mengatur interaksi objek, contoh: Observer).
7. Singleton Pattern adalah design pattern yang memastikan hanya satu instance dari suatu kelas yang dapat dibuat, dengan akses global yang konsisten.
8. Variasi Implementasi Singleton:
    a. Eager Initialization: dibuat saat kelas dimuat.
    b. Thread-safe Singleton (Synchronized): dibuat saat diperlukan dengan synchronized.
    c. Bill Pugh Singleton: menggunakan inner static class, lazy dan thread-safe.
    d. Double-Checked Locking: optimisasi synchronized dengan dua kali pengecekan.
9. Bill Pugh Singleton disarankan untuk digunakan di Java modern karena kombinasi simplicity, performance, dan thread-safety.

Implementasi	**Cara Kerja**	**Kelebihan**	**Kekurangan**	**Penggunaan yang cocok**
Eager Initialization	Membuat instance saat kelas diload	Sederhana, aman	Boros memori bila instance tidak dipakai	Instance pasti diperlukan
Tread-safe (Synchronized)	getInstance() disinkronisasi	Aman multi-thread	Lambat karena sinkronisasi	Untuk sistem kecil
Bill Pugh Singleton	Inner static class	Cepat, lazy loading, aman	-	Standar Java modern
Double-checked Locking	Dua kali pengecekan + synchronized	Efisien setelah instance dibuat	Lebih kompleks	Sistem butuh kontrol performa khusus

12.4 Latihan

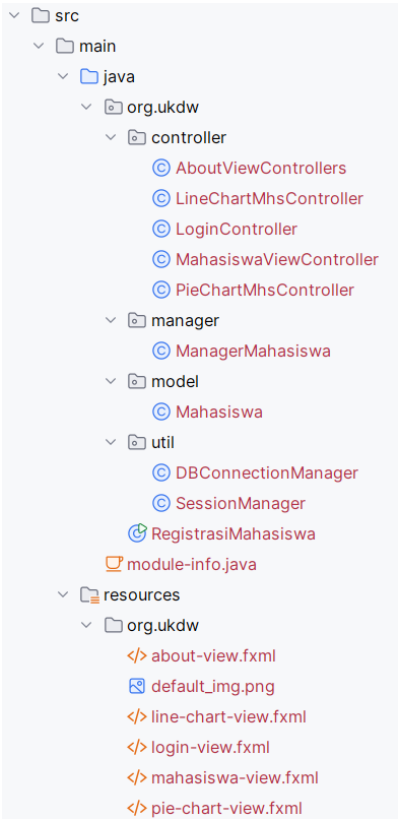
Anda diminta untuk menambahkan beberapa fitur pada aplikasi desktop sederhana menggunakan JavaFX dan SQLite untuk mencatat data mahasiswa yang sudah dibuat pada modul 11. Fitur-fitur tersebut adalah sebagai berikut :

1. Menambahkan fitur login sebagai tampilan pertama saat menggunakan sistem. Untuk melakukan login, menggunakan username “admin” dan password “admin”.

2. Menambahkan session management untuk merekam sesi login user. Session akan disimpan pada sebuah file bernama session.ser, yang akan menyimpan informasi apakah user sudah login apa belum.
3. Menambahkan fitur logout pada tampilan utama sistem, yang jika berhasil akan menampilkan kembali form login.

12.5 Kunci jawaban

Untuk membuat session management, maka dibutuhkah operasi CRUD pada file dengan menggunakan JavaIO. Untuk mengimplementasi fitur-fitur yang diinginkan diatas, maka perlu memodifikasi class MahasiswaViewController untuk menambah fitur logout dan class RegistrasiMahasiswa untuk implementasi session jika user telah melakukan login atau belum. Selain itu perlu menambahkan form baru bernama login-view.fxml lengkap dengan controllernya dan class SessionManager untuk implementasi session management. Struktur kode project adalah sebagai berikut:



Gambar 12.5.1 Struktur project latihan modul 12

Listing kode program login-view.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
```

```
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" onKeyPressed="#onKeyPressEvent"
prefHeight="182.0" prefWidth="316.0"
xmlns="http://javafx.com/javafx/17.0.2-ea"
xmlns:fx="http://javafx.com/fxml/1"
            fx:controller="org.ukdw.controller.LoginController">
   <children>
      <Label layoutX="66.0" layoutY="31.0" text="Welcome, Login Here">
         <font>
            <Font name="System Bold" size="18.0" />
         </font></Label>
      <Label layoutX="34.0" layoutY="73.0" text="Username:" />
      <Label layoutX="34.0" layoutY="111.0" text="Password:" />
      <TextField fx:id="txtUsername" layoutX="119.0" layoutY="69.0"
promptText="Username" />
      <PasswordField fx:id="txtPassword" layoutX="119.0" layoutY="106.0"
promptText="Password" />
      <Button fx:id="btnLogin" layoutX="119.0" layoutY="144.0"
mnemonicParsing="false" onAction="#btnLoginClick" text="Login" />
      <Hyperlink fx:id="lblForgot" layoutX="172.0" layoutY="144.0"
text="Forget Password" />
   </children>
</AnchorPane>
```

Listing kode program mahasiswa-view.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.VBox?>

<HBox xmlns="http://javafx.com/javafx/21"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="org.ukdw.controller.MahasiswaViewController">
   <children>
      <VBox prefWidth="350.0" style="-fx-border-image-width: 2; -fx-
border-style: solid inside;">
         <MenuBar>
           <menus>
             <Menu mnemonicParsing="false" text="File">
               <items>
                 <MenuItem mnemonicParsing="false"
onAction="#onActionLogout" text="Logout" />
               </items>
```

```
            </Menu>
            <Menu mnemonicParsing="false" text="Edit">
              <items>
                <MenuItem mnemonicParsing="false" text="Delete" />
              </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Help">
              <items>
                <MenuItem mnemonicParsing="false"
onAction="#onActionAbout" text="About" />
              </items>
            </Menu>
          </menus>
           <VBox.margin>
              <Insets />
           </VBox.margin>
        </MenuBar>
         <GridPane VBox.vgrow="ALWAYS">
             <columnConstraints>
                 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
                 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" />
             </columnConstraints>
             <rowConstraints>
                 <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                 <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                 <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
              <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
             </rowConstraints>
             <Label text="NIM:" />
             <Label text="Nama:" GridPane.rowIndex="1" />
             <Label text="Nilai:" GridPane.rowIndex="2" />
             <TextField fx:id="txtNim" promptText="NIM"
GridPane.columnIndex="1" />
             <TextField fx:id="txtNama" promptText="Nama"
GridPane.columnIndex="1" GridPane.rowIndex="1" />
             <TextField fx:id="txtNilai" promptText="Nilai"
GridPane.columnIndex="1" GridPane.rowIndex="2" />
            <Label text="Foto" GridPane.rowIndex="3" />
              <VBox alignment="TOP_CENTER" GridPane.columnIndex="1"
GridPane.rowIndex="3" GridPane.vgrow="ALWAYS">
                 <ImageView fx:id="imgFoto" disable="true"
fitHeight="150.0" fitWidth="100.0" pickOnBounds="true"
preserveRatio="true" />
                 <Button fx:id="btnUbahFoto" mnemonicParsing="false"
onAction="#onBtnPilihFotoClick" text="Ubah Foto" />
             </VBox>
            <VBox.margin>
               <Insets bottom="10.0" left="10.0" right="10.0" top="10.0"
/>
            </VBox.margin>
          </GridPane>
          <HBox>
             <Button fx:id="btnAdd" onAction="#onBtnAddClick"
```

```
text="Tambah">
                    <HBox.margin>
                        <Insets right="10.0" />
                    </HBox.margin>
                </Button>
                <Button fx:id="btnSimpan" mnemonicParsing="false"
onAction="#onBtnSimpanClick" text="Simpan">
                    <HBox.margin>
                        <Insets right="10.0" />
                    </HBox.margin>
                </Button>
                <Button fx:id="btnHapus" mnemonicParsing="false"
onAction="#onBtnHapusClick" text="Hapus" />
             <VBox.margin>
                <Insets left="10.0" right="10.0" />
             </VBox.margin>
          </HBox>
          <VBox>
             <children>
                 <Label fx:id="lblInfo" text="Jumlah: " />
                <HBox spacing="2.0">
                   <children>
                      <Button fx:id="btnLineChart" mnemonicParsing="false"
onAction="#onLineChartCLicked" text="Line Chart" />
                      <Button fx:id="btnPieChart" mnemonicParsing="false"
onAction="#onPieChartCLicked" text="Pie Chart" />
                   </children>
                </HBox>
             </children>
             <VBox.margin>
                <Insets bottom="10.0" left="10.0" right="10.0" />
             </VBox.margin>
          </VBox>
       </VBox>
       <VBox>
          <children>
              <GridPane VBox.vgrow="NEVER">
                  <columnConstraints>
                      <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
/>
                      <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
/>
                  </columnConstraints>
                  <rowConstraints>
                      <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
                  </rowConstraints>
                  <VBox.margin>
                      <Insets left="10.0" right="10.0" />
                  </VBox.margin>
                  <Label text="Cari NIM atau Nama">
                     <padding>
                         <Insets right="10.0" />
                     </padding>
                  </Label>
                  <TextField fx:id="txtNamaCari" promptText="NIM atau Nama"
GridPane.columnIndex="1" />
              </GridPane>
```

```
            <TableView fx:id="tblView" VBox.vgrow="ALWAYS">
                <columns>
                    <TableColumn fx:id="colNIM" prefWidth="75.0"
text="NIM" />
                    <TableColumn fx:id="colNama" prefWidth="210"
text="Nama" />
                    <TableColumn fx:id="colNilai" prefWidth="75.0"
text="Nilai" />
                </columns>
            </TableView>
        </children>
    </VBox>
  </children>
</HBox>
```

Listing kode program RegistrasiMahasiswa.java

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;

import java.io.IOException;

import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Modality;
import javafx.stage.Stage;
import org.ukdw.util.SessionManager;

public class RegistrasiMahasiswa extends Application {

    private static Stage primaryStage;

    public RegistrasiMahasiswa() {

    }

    @Override
    public void start(Stage stage) throws IOException {
        primaryStage = stage;
        primaryStage.setTitle("Data Nilai Mahasiswa");
        if (SessionManager.getInstance().isLoggedIn()) {
            primaryStage.setScene(new Scene(loadFXML("mahasiswa-view")));
        } else {
            primaryStage.setScene(new Scene(loadFXML("login-view")));
        }
        primaryStage.show();
    }

    private static Parent loadFXML(String fxml) {
        FXMLLoader fxmlLoader = new FXMLLoader(RegistrasiMahasiswa.class
                .getResource(fxml + ".fxml"));
        try {
            return fxmlLoader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
```

```
    }

    public static void setRoot(String fxml, boolean isResizeable) {
        primaryStage.getScene().setRoot(loadFXML(fxml));
        primaryStage.sizeToScene();
        primaryStage.setResizable(isResizeable);
    }

    public static void openViewWithModal(String fxml, boolean
isResizeable) {
        Stage stage = new Stage();
        stage.setScene(new Scene(loadFXML(fxml)));
        stage.sizeToScene();
        stage.setResizable(isResizeable);
        stage.initOwner(primaryStage);
        stage.initModality(Modality.WINDOW_MODAL);
        stage.showAndWait();
    }

    public static void main(String[] args) throws IOException {
        launch();
    }
}
```

Listing kode program SessionManager.java

```
import java.io.*;

public class SessionManager implements Serializable {
    @Serial
    private static final long serialVersionUID = 1L;
    private static final String SESSION_FILE = "session.ser";

    private static volatile SessionManager instance;
    private boolean isLoggedIn;

    // Private constructor to prevent instantiation from outside
    private SessionManager() {
        isLoggedIn = false;
    }

    // Static method to get the singleton instance
    public static SessionManager getInstance() {
        if (instance == null) {
            synchronized (SessionManager.class) {
                if (instance == null) {
                    instance = new SessionManager();
                    instance.createSessionFile();
                }
            }
        }
        return instance;
    }

    // Method to check if the session file doesn't exist
    public void createSessionFile() {
```

```
        File file = new File(SESSION_FILE);
        if (!file.exists()) {
            saveSession();
        } else {
            loadSession();
        }
    }

    private void loadSession() {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(SESSION_FILE))) {
            SessionManager sessionManager = (SessionManager)
ois.readObject();
            this.isLoggedIn = sessionManager.isLoggedIn;
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error loading session: " +
e.getMessage());
        }
    }

    private void saveSession() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(SESSION_FILE))) {
            oos.writeObject(this);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Method to check if user is logged in
    public boolean isLoggedIn() {
        return isLoggedIn;
    }

    // Method to simulate login
    public void login() {
        isLoggedIn = true;
        saveSession();
    }

    // Method to simulate logout
    public void logout() {
        isLoggedIn = false;
        saveSession();
    }
}
```

Listing kode program MahasiswaViewController.java

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.collections.transformation.FilteredList;
import javafx.collections.transformation.SortedList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.chart.BarChart;
```

```
import javafx.scene.control.*;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.net.URL;
import java.util.Objects;
import java.util.ResourceBundle;
import java.util.function.Predicate;

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.FileChooser;
import org.ukdw.model.Mahasiswa;
import org.ukdw.manager.ManagerMahasiswa;
import org.ukdw.RegistrasiMahasiswa;
import org.ukdw.util.SessionManager;

import javax.imageio.ImageIO;

public class MahasiswaViewController implements Initializable {

    @FXML
    public TableView<Mahasiswa> tblView;

    @FXML
    private TableColumn<Mahasiswa, String> colNIM;
    @FXML
    private TableColumn<Mahasiswa, String> colNama;
    @FXML
    private TableColumn<Mahasiswa, Double> colNilai;

    @FXML
    private TextField txtNim;
    @FXML
    private TextField txtNama;
    @FXML
    private TextField txtNilai;
    @FXML
    public TextField txtNamaCari;

    @FXML
    private Label lblInfo;

    @FXML
    private BarChart<String, Number> barChart;

    @FXML
    private ImageView imgFoto;

    private byte[] selectedMahasiswaFotoBlob;

    private ManagerMahasiswa manager;
    private ObservableList<Mahasiswa> masterData;
```

```
    private FilteredList<Mahasiswa> mahasiswaFilteredList;

    private Mahasiswa selectedMahasiswa;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        this.manager = new ManagerMahasiswa();
        this.masterData = FXCollections.observableArrayList(); // ← list
utama
        this.mahasiswaFilteredList = new FilteredList<>(masterData, p ->
true); // ← filtered list dari master


tblView.getSelectionModel().selectedItemProperty().addListener((obs,
oldVal, newVal) -> {
            if (newVal != null) {
                selectedMahasiswa = newVal;
                txtNim.setText(newVal.getNim());
                txtNama.setText(newVal.getNama());
                txtNilai.setText(String.valueOf(newVal.getNilai()));
                txtNim.setDisable(true); // NIM tidak boleh diubah saat
edit
                selectedMahasiswaFotoBlob = newVal.getFoto();

imgFoto.setImage(convertBytesToImage(selectedMahasiswaFotoBlob));
            }
        });

        SortedList<Mahasiswa> sortedData = new
SortedList<>(mahasiswaFilteredList);

sortedData.comparatorProperty().bind(tblView.comparatorProperty());
        tblView.setItems(sortedData);

        txtNamaCari.textProperty().addListener((
                (obs, oldValue, newValue) ->
                {

mahasiswaFilteredList.setPredicate(createPredicate(newValue));
                }
        ));

        // Display the image in imageView
        Image image = new
Image(Objects.requireNonNull(RegistrasiMahasiswa.class.getResourceAsStream
("default_img.png")));
        imgFoto.setImage(image);

        displayList();
    }

    private Predicate<? super Mahasiswa> createPredicate(String
searchText) {
        return order -> {
            if (searchText == null || searchText.isEmpty()) return true;
            return searchFindsMahasiswa(order, searchText);
        };
```

```
    }

    private boolean searchFindsMahasiswa(Mahasiswa mahasiswa, String
searchText) {
        return
(mahasiswa.getNim().toLowerCase().contains(searchText.toLowerCase())) ||

(mahasiswa.getNama().toLowerCase().contains(searchText.toLowerCase()));
    }

    private void displayList() {
        tblView.setEditable(false);
        colNIM.setCellValueFactory(new PropertyValueFactory<>("nim"));
        colNama.setCellValueFactory(new PropertyValueFactory<>("nama"));
        colNilai.setCellValueFactory(new PropertyValueFactory<>("nilai"));
        masterData.clear();
        masterData.addAll(manager.getAllMahasiswa());
        updateInfo();
    }

    private void updateInfo() {
        double rata = 0;
        for (Mahasiswa m : masterData) {
            rata += m.getNilai();
        }
        rata = !masterData.isEmpty() ? rata / masterData.size() : 0;
        lblInfo.setText("Jumlah data: " + masterData.size() + ", Rata
nilai: " + rata);
    }

    private void bersihkan() {
        txtNim.clear();
        txtNama.clear();
        txtNilai.clear();
        txtNim.setDisable(false);
        // Display the image in imageView
        Image image = new
Image(Objects.requireNonNull(RegistrasiMahasiswa.class.getResourceAsStream
("default_img.png")));
        imgFoto.setImage(image);

        tblView.getSelectionModel().clearSelection();
        selectedMahasiswa = null;
        updateInfo();
    }

    @FXML
    public void onBtnAddClick(ActionEvent actionEvent) {
        Mahasiswa newMahasiswa = new Mahasiswa(txtNim.getText(),
txtNama.getText(),
                Double.parseDouble(txtNilai.getText()),
selectedMahasiswaFotoBlob);
        if (manager.tambahMahasiswa(newMahasiswa)) {
            masterData.add(newMahasiswa);
            Alert alert = new Alert(Alert.AlertType.INFORMATION, "Data
Mahasiswa Ditambahkan!");
            alert.show();
```

```
            bersihkan();
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR, "Data
Mahasiswa gagal Ditambahkan!");
            alert.show();
            bersihkan();
        }
    }

    @FXML
    public void onBtnSimpanClick(ActionEvent actionEvent) {
        if (selectedMahasiswa == null) {
            new Alert(Alert.AlertType.WARNING, "Pilih data dari tabel
untuk diperbarui.").show();
            return;
        }

        Mahasiswa updatedMahasiswa = new Mahasiswa(txtNim.getText(),
txtNama.getText(),
                Double.parseDouble(txtNilai.getText()),
selectedMahasiswaFotoBlob);
        if (manager.updateMahasiswa(updatedMahasiswa)) {
            selectedMahasiswa.setNama(updatedMahasiswa.getNama());
            selectedMahasiswa.setNilai(updatedMahasiswa.getNilai());
            selectedMahasiswa.setFoto(updatedMahasiswa.getFoto());
            tblView.refresh();
            Alert alert = new Alert(Alert.AlertType.INFORMATION, "Data
Mahasiswa Diperbarui!");
            alert.show();
            bersihkan();
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR, "Data
Mahasiswa gagal Diperbarui!");
            alert.show();
            bersihkan();
        }
    }

    @FXML
    public void onBtnHapusClick(ActionEvent actionEvent) {
        if (selectedMahasiswa == null) {
            new Alert(Alert.AlertType.WARNING, "Pilih data dari tabel
untuk dihapus.").show();
            return;
        }

        if (manager.hapusMahasiswa(selectedMahasiswa.getNim())) {
            masterData.remove(selectedMahasiswa);
            new Alert(Alert.AlertType.INFORMATION, "Data Mahasiswa
Dihapus!").show();
            bersihkan();
        } else {
            new Alert(Alert.AlertType.ERROR, "Data Mahasiswa gagal
Dihapus!").show();
        }
    }
```

```
    @FXML
    protected void onBtnPilihFotoClick() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Pilih Foto Mahasiswa");
        fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("Image Files", "*.jpg",
"*.jpeg", "*.png")
        );
        File file =
fileChooser.showOpenDialog(imgFoto.getScene().getWindow());
        if (file != null) {
            selectedMahasiswaFotoBlob = resizeImageToBytes(file, 100,
150);
            imgFoto.setImage(new Image(file.toURI().toString()));
        }
    }

    private byte[] resizeImageToBytes(File file, int width, int height) {
        try {
            BufferedImage originalImage = ImageIO.read(file);
            BufferedImage resizedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
            Graphics2D g2d = resizedImage.createGraphics();
            g2d.drawImage(originalImage.getScaledInstance(width, height,
java.awt.Image.SCALE_SMOOTH), 0, 0, null);
            g2d.dispose();
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ImageIO.write(resizedImage, "png", baos);
            return baos.toByteArray();
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    private Image convertBytesToImage(byte[] bytes) {
        if (bytes == null) return null;
        return new Image(new ByteArrayInputStream(bytes));
    }

    @FXML
    public void onLineChartCLicked() {
        RegistrasiMahasiswa.openViewWithModal("line-chart-view", false);
    }

    @FXML
    public void onPieChartCLicked(ActionEvent actionEvent) {
        RegistrasiMahasiswa.openViewWithModal("pie-chart-view", false);
    }

    public void onActionLogout(ActionEvent actionEvent) {
        // Create a new alert with type Confirmation
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Exit & Logout Confirmation");
        alert.setHeaderText("Are you sure you want to exit?");
        alert.setContentText("Press OK to exit the application.");
```

```
        // Add Yes and No buttons to the alert
        alert.getButtonTypes().setAll(ButtonType.YES, ButtonType.NO);
        // Show the alert and wait for user response
        alert.showAndWait().ifPresent(response -> {
            if (response == ButtonType.YES) {
                // User clicked Yes, exit the application
                SessionManager.getInstance().logout();
                RegistrasiMahasiswa.setRoot("login-view", false);
            }
        });
    }

    public void onActionAbout(ActionEvent actionEvent) {
        RegistrasiMahasiswa.openViewWithModal("about-view",false);
    }
}
```

Listing kode program LoginController.java

```
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import org.ukdw.RegistrasiMahasiswa;
import org.ukdw.util.SessionManager;

import java.io.IOException;

public class LoginController {
    private static final String CORRECT_USERNAME = "admin";
    private static final String CORRECT_PASSWORD = "admin";

    @FXML
    private TextField txtUsername;
    @FXML private PasswordField txtPassword;

    @FXML
    protected void onKeyPressEvent(KeyEvent event) throws IOException {
        if( event.getCode() == KeyCode.ENTER ) {
            btnLoginClick();
        }
    }
    @FXML
    protected void btnLoginClick() {
        Alert alert;
        if (txtUsername.getText().equals(CORRECT_USERNAME) &&
txtPassword.getText().equals(CORRECT_PASSWORD)) {
            alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setHeaderText("Information");
            alert.setContentText("Login success!!");
            SessionManager.getInstance().login();
            alert.showAndWait();
            RegistrasiMahasiswa.setRoot("mahasiswa-view", false);
        } else {
            alert = new Alert(Alert.AlertType.ERROR);
```

```
            alert.setHeaderText("Error");
            alert.setContentText("Login failed!! Please check again.");
            alert.showAndWait();
            txtUsername.requestFocus();
        }
    }
}
```