

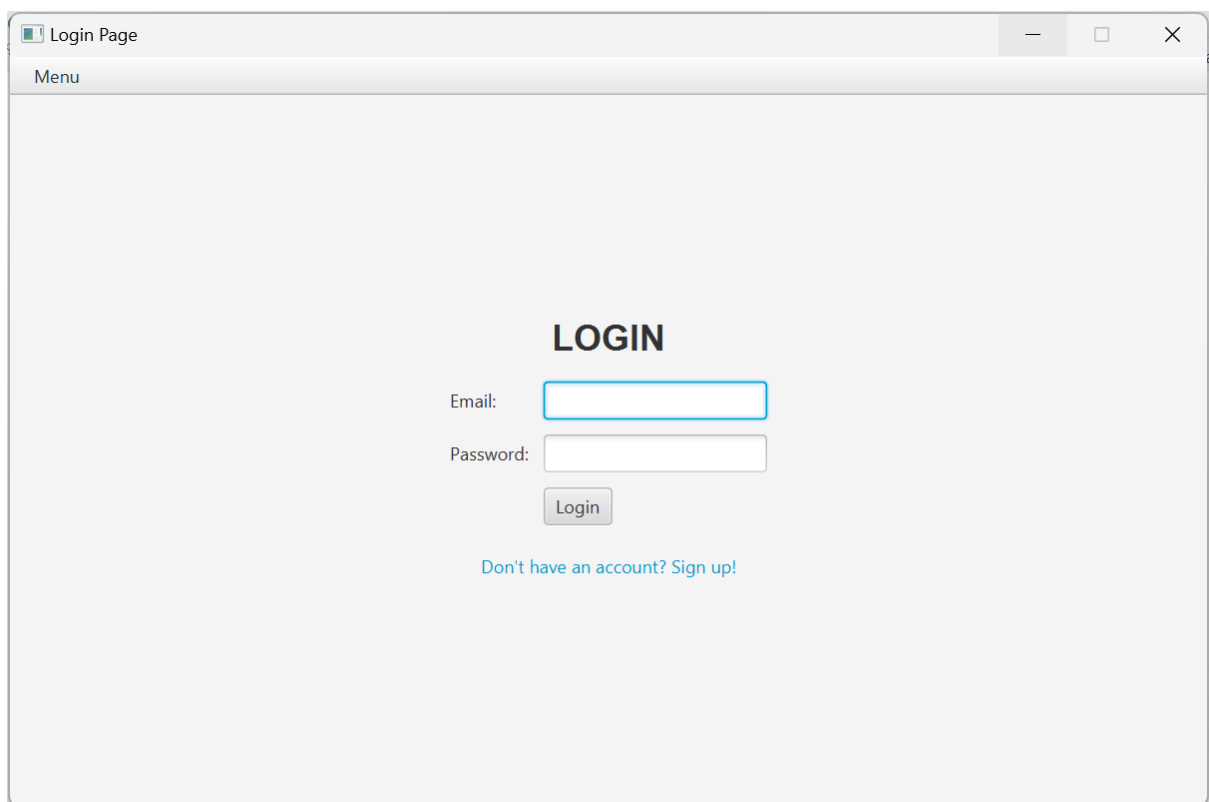
Nama Kelompok:

- **Ester Floria Usboko**
- **Sharon Evalina Jati**
- **Thomas Eurico Siregar**
- **Musa Widodo**

Kelas : BC-80

Documentation

1.Login



Menu

LOGIN

Email:

Password:

Login

[Don't have an account? Sign up!](#)

1. Struktur Utama

Tampilan tersebut menjelaskan sebuah implementasi aplikasi GUI berbasis JavaFX yang menyediakan:

Halaman login dengan validasi input.

Navigasi untuk berpindah ke halaman lain (registrasi, beranda admin, atau beranda pengguna).

Komunikasi dengan database untuk autentikasi pengguna.

a.Import Statements

import javafx.application.Application;

```
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;
import model.User;
import util.Connect;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

- Mengimpor pustaka JavaFX untuk GUI, seperti Button, Label, dan GridPane.
- Mengimpor kelas untuk koneksi database (Connection, PreparedStatement, dan ResultSet).
- Mengimpor User (model pengguna) dan Connect (utilitas koneksi database).

b. Kelas Utama

java

Salin kode

```
public class Login extends Application {
```

Kelas ini memperluas **Application**, yang merupakan kelas dasar untuk membuat aplikasi JavaFX.

c. Metode Utama

i. createLoginLayout

Metode ini membuat layout halaman login:

java

Salin kode

```
private BorderPane createLoginLayout(Stage stage) {  
    // Komponen visual seperti Label, TextField, PasswordField, dan Button.  
}
```

1. Komponen UI:

- **Label:** Menampilkan teks, seperti "LOGIN", "Email:", dan "Password:".
- **TextField:** Untuk input email.
- **PasswordField:** Untuk input kata sandi.
- **Button:** Tombol login.
- **Hyperlink:** Tautan ke halaman registrasi.

2. Tata Letak:

- **GridPane:** Menyusun label dan input dalam bentuk grid.
 - **VBox:** Mengatur elemen secara vertikal.
 - **BorderPane:** Tata letak utama dengan isi di tengah.
-

ii. createMenu

Metode ini membuat menu dengan item untuk login dan registrasi:

java

Salin kode

```
private MenuBar createMenu(Stage stage) {  
    Menu menu = new Menu("Menu");  
    MenuItem loginItem = new MenuItem("Login");  
    MenuItem registerItem = new MenuItem("Register");  
    menu.getItems().addAll(loginItem, registerItem);  
}
```

- **Login Menu Item:** Mengarahkan pengguna ke halaman login (default).
 - **Register Menu Item:** Mengarahkan pengguna ke halaman registrasi.
-

iii. configureActions

Metode ini mengatur aksi yang terjadi saat tombol atau tautan diklik:

java

Salin kode

```
loginButton.setOnAction(event -> {  
    String email = emailField.getText().trim();  
    String password = passField.getText();  
    // Validasi input dan koneksi ke database.  
});
```

1. **Login Button:**

- Mengecek apakah email dan password kosong.
- Memvalidasi user dari database menggunakan metode **validateUser**.
- Mengarahkan ke halaman sesuai peran pengguna (Admin/Customer).

2. **Register Link:**

- Mengarahkan pengguna ke halaman registrasi menggunakan **redirectToRegister**.

iv. **validateUser**

Metode untuk memeriksa kredensial pengguna dari database:

java

Salin kode

```
private User validateUser(String email, String password) throws SQLException {  
    String query = "SELECT UserID, Role FROM msuser WHERE email = ? AND password = ?";  
    try (Connection connection = Connect.getConnection();  
        PreparedStatement preparedStatement = connection.prepareStatement(query)) {  
        preparedStatement.setString(1, email);  
        preparedStatement.setString(2, password);  
        // Eksekusi query dan baca hasil.  
    }  
}
```

- Menjalankan query SQL untuk memeriksa kombinasi email dan password.
 - Jika cocok, mengembalikan objek **User** dengan UserID dan Role.
-

v. **redirectToAdminHome** dan **redirectToUserHome**

Mengarahkan pengguna ke halaman admin atau pengguna biasa berdasarkan peran:

java

Salin kode

```
private void redirectToAdminHome(Stage stage, String userId) {  
    AdminHome adminHome = new AdminHome(userId);  
    adminHome.start(stage);  
}  
  
private void redirectToUserHome(Stage stage, String userId) {  
    CustomerHome userHome = new CustomerHome(userId);  
    userHome.start(stage);  
}
```

- **AdminHome** dan **CustomerHome** adalah kelas lain untuk beranda admin dan pengguna.
-

vi. **clearFields** dan **showAlert**

Metode tambahan untuk:

- Membersihkan input email dan password.
 - Menampilkan pesan pop-up (alert) untuk kesalahan atau informasi.
-

vii. **start**

- Membuat tampilan utama aplikasi dengan menu dan form login.
 - Menampilkan jendela aplikasi (stage) dengan ukuran **800x500**.
-

d. **Main Method**

Memulai aplikasi dengan memanggil metode **launch** dari JavaFX.

2.Registrasi

Menu

Register

Username:

Email:

Password:

Confirm Password:

Age:

Gender: ☐ Male ☐ Female

Country:

Phone Number:

☐ Agree to the terms and conditions

[Already have an account? Sign in!](#)

Kodingan di atas merupakan sebuah aplikasi **JavaFX** yang Dimana membuat sebuah antarmuka registrasi pengguna yang terhubung dengan database. Berikut penjelasan mendetail tentang setiap bagian kode:

1. Import Statement

Mengimpor pustaka yang diperlukan:

- `javafx.*`: Untuk membuat antarmuka pengguna berbasis JavaFX.
- `java.sql.*`: Untuk berinteraksi dengan database.
- `util.Connect`: Kelas utilitas yang menyediakan koneksi ke database.

2. Struktur Kelas

Kelas **Registration** adalah subkelas dari `Application`, yang merupakan bagian dari JavaFX untuk membuat aplikasi GUI.

3. Properti dan Variabel

- **Layout Komponen:**

- **BorderPane:** Layout utama.
 - **GridPane:** Untuk menyusun input form registrasi.
 - **VBox:** Untuk menyusun elemen secara vertikal.
 - **Komponen UI:**
 - **Label, TextField, PasswordField:** Untuk input teks seperti username, email, dll.
 - **Spinner:** Untuk input angka umur.
 - **RadioButton:** Untuk memilih gender.
 - **ComboBox:** Untuk memilih negara.
 - **CheckBox:** Untuk konfirmasi syarat dan ketentuan.
 - **Button, Hyperlink, MenuBar:** Untuk interaksi tambahan.
 - **Properti lainnya:**
 - **Scene:** Untuk menampilkan keseluruhan antarmuka.
 - **ToggleGroup:** Untuk mengelompokkan RadioButton.
-

4. Metode Utama

a. initialize(Stage stage)

Menginisialisasi komponen UI:

- Membuat elemen UI seperti **TextField**, **ComboBox**, dan sebagainya.
- Menambahkan menu navigasi dengan item "Login" dan "Register".

b. setComponent()

- Menambahkan semua komponen UI ke dalam BorderPane dan GridPane.
- Menyusun komponen seperti Label, TextField, dan elemen lainnya ke dalam layout GridPane.

c. arrangeComponent()

Mengatur tata letak dan properti elemen, seperti:

- Alignment (penyelarasan).
- Margin antar elemen.

d. validate()

Metode untuk memvalidasi input pengguna:

- **Validasi form:**
 - Username: 3-15 karakter.

- Email: Harus diakhiri dengan @gmail.com.
 - Password: Harus alfanumerik dan cocok dengan password konfirmasi.
 - Umur: Harus di atas 13 tahun.
 - Nomor Telepon: Maksimal 15 karakter.
 - Checkbox: Harus dicentang.
 - Jika ada kesalahan, menampilkan alert error.
 - Jika validasi berhasil:
 - Membuat ID pengguna baru melalui **generateUserId()**.
 - Menyimpan data ke database dengan **saveToDatabase()**.
-

e. generateUserId()

Membuat ID pengguna baru:

1. Mengeksekusi query SQL untuk mendapatkan nilai maksimal dari kolom UserID.
2. Menambahkan 1 ke nilai maksimal, lalu memformatnya dengan format USxxx (zero-padded).

f. saveToDatabase()

- Menyimpan data pengguna ke tabel msuser di database menggunakan query **INSERT**.
 - Jika berhasil:
 - Menampilkan pesan sukses dengan ID pengguna.
 - Jika gagal:
 - Menampilkan alert error dengan detail kesalahan.
-

g. showLoginPage(Stage stage)

Mengganti tampilan ke halaman login menggunakan kelas **Login**.

5. Metode start(Stage primaryStage)

- Metode utama yang memulai aplikasi.
- Memanggil metode:
 - **initialize()**: Inisialisasi komponen.
 - **setComponent()**: Menyusun komponen.
 - **arrangeComponent()**: Mengatur tata letak.

- **validate()**: Menambahkan logika validasi input.
 - Menyusun **Scene** dan menampilkan aplikasi.
-

6. Main Method

Menjalankan aplikasi JavaFX menggunakan `launch(args)`.

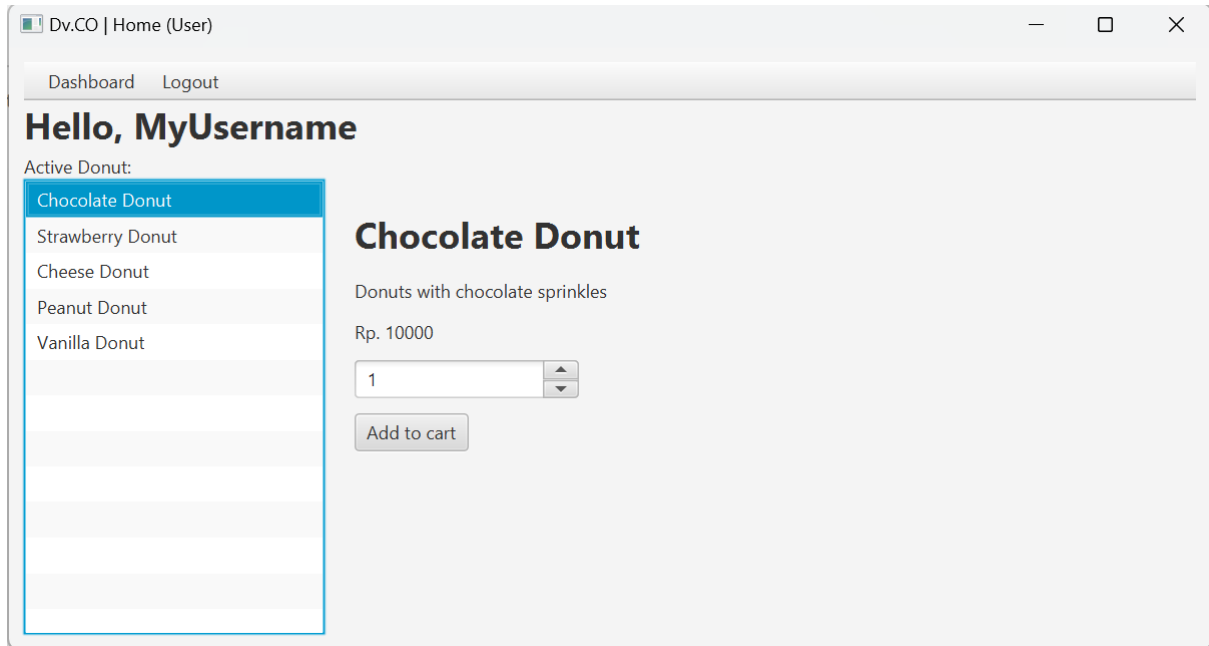
Ringkasan Alur Kerja

1. Aplikasi dibuka, menampilkan form registrasi.
 2. Pengguna mengisi form dan mengklik tombol **Register**.
 3. Validasi input dilakukan:
 - Jika ada kesalahan, ditampilkan alert error.
 - Jika sukses, data disimpan ke database.
 4. Menampilkan pesan sukses beserta ID pengguna.
-

Kebutuhan Eksternal

- **Database**: Aplikasi menggunakan tabel `msuser` dengan kolom seperti `UserID`, `Username`, `Email`, dll.
- **Kelas util.Connect**: Untuk mendapatkan koneksi ke database.

3.Customer Home



package main;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.*;

import javafx.scene.layout.BorderPane;

import javafx.scene.layout.HBox;

import javafx.scene.layout.StackPane;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.scene.text.FontWeight;

import javafx.stage.Stage;

import model.Donut;

import util.Connect;

import java.sql.*;

import java.util.HashMap;

import java.util.Map;

```
public class CustomerHome extends Application {
```

```
    private final Map<String, Integer> cart = new HashMap<>(); // Cart to store donutID  
    (String) and quantity (Integer)
```

```
    private String username = ""; // Username fetched from the database during login
```

```
    private String userId; // Variable to store the logged-in user ID
```

```
    // Constructor to accept the userId
```

```
    public CustomerHome(String userId) {
```

```
        this.userId = userId;
```

```
    }
```

```
    @Override
```

```
    public void start(Stage primaryStage) {
```

```
        username = getLoggedInUsername(userId); // Use the userId passed to the constructor
```

```
        BorderPane root = createHomeLayout(primaryStage);
```

```
        primaryStage.setScene(new Scene(root, 800, 500));
```

```
        primaryStage.setTitle("Dv.CO | Home (User)");
```

```
        primaryStage.setResizable(false);
```

```
        primaryStage.show();
```

```
    }
```

```
    private BorderPane createHomeLayout(Stage primaryStage) {
```

```
        // Top Menu Bar
```

```
        MenuBar menuBar = new MenuBar();
```

```
        Menu menu = new Menu("Dashboard");
```

```
        Menu logout = new Menu("Logout");
```

```
        MenuItem homeItem = new MenuItem("Home");
```

```
        MenuItem cartItem = new MenuItem("Cart");
```

```

MenuItem logoutItem = new MenuItem("Logout");
menu.getItems().addAll(homeItem, cartItem);
logout.getItems().addAll(logoutItem);
menuBar.getMenus().addAll(menu, logout);

homeItem.setDisable(true);

// Event Handlers for MenuItems
cartItem.setOnAction(event -> showCartScene(primaryStage, userId));
logoutItem.setOnAction(event -> logout(primaryStage)); // Placeholder for logout
functionality

// Left Section: List of Donuts
ListView<String> donutListView = new ListView<>();
populateDonutList(donutListView);
donutListView.setPrefWidth(200);

// Right Section: Details
VBox detailsBox = new VBox(10);
detailsBox.setPadding(new Insets(10));

Label donutNameLabel = new Label();
donutNameLabel.setFont(Font.font("Arial", FontWeight.BOLD, 24));
Label donutDescriptionLabel = new Label();
donutDescriptionLabel.setWrapText(true);
Label donutPriceLabel = new Label();

Spinner<Integer> quantitySpinner = new Spinner<>(1, 999, 1);
Button addToCartButton = new Button("Add to cart");
addToCartButton.setVisible(false);

```

```

    detailsBox.getChildren().addAll(donutNameLabel, donutDescriptionLabel,
    donutPriceLabel, quantitySpinner, addToCartButton);

    detailsBox.setVisible(false);

// Center Layout

HBox centerBox = new HBox(10, donutListView, detailsBox);
centerBox.setPadding(new Insets(10));

// Welcome Section

Label welcomeLabel = new Label("Hello, " + username);
welcomeLabel.setFont(Font.font("Arial", FontWeight.BOLD, 20));
VBox topBox = new VBox(10, menuBar, welcomeLabel);
topBox.setPadding(new Insets(10));

// Main Layout

BorderPane root = new BorderPane();
root.setTop(topBox);
root.setCenter(centerBox);

// Event Handlers

donutListView.getSelectionModel().selectedItemProperty().addListener((observable,
oldValue, newValue) -> {
    if (newValue != null) {
        Donut donut = getDonutDetails(newValue);
        if (donut != null) {
            donutNameLabel.setText(donut.getDonutName());
            donutDescriptionLabel.setText(donut.getDonutDescription());
            donutPriceLabel.setText("Price: Rp. " + donut.getDonutPrice());
            detailsBox.setVisible(true);
            addToCartButton.setVisible(true);
        }
    }
}

```

```

    } else {
        detailsBox.setVisible(false);
        addToCartButton.setVisible(false);
    }
});

// Inside the addToCartButton event handler
addToCartButton.setOnAction(event -> {
    String selectedDonut = donutListView.getSelectionModel().getSelectedItem();
    int quantity = quantitySpinner.getValue();

    if (selectedDonut != null) {
        Donut donut = getDonutDetails(selectedDonut);
        if (donut != null) {
            saveToCart(userId, donut.getDonutID(), quantity); // Save to cart in the database

            // Change to use DonutID as the key (String)
            cart.merge(donut.getDonutID(), quantity, Integer::sum); // Merge by summing
the quantities

            showAlert(Alert.AlertType.INFORMATION, "Added to Cart", "You have added
" + quantity + " " + selectedDonut + "(s) to your cart.");
        }
    } else {
        showAlert(Alert.AlertType.WARNING, "Selection Error", "Please select a donut to
add to the cart.");
    }
});

return root;
}

```

```

private void logout(Stage primaryStage) {
    // Clear the logged-in user data

    this.userId = null;

    this.username = null;

    // Clear the current scene by setting a new empty scene
    primaryStage.setScene(new Scene(new StackPane(), 800, 600)); // Empty scene to clear
the UI

    // Redirect to Login scene
    Login loginScene = new Login(); // Assuming you have a Login class
    try {
        loginScene.start(primaryStage);
    } catch (Exception ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to load login screen.");
    }
}

private void populateDonutList(ListView<String> listView) {
    try (Connection connection = Connect.getConnection());
        Statement statement = connection.createStatement();

        ResultSet resultSet = statement.executeQuery("SELECT DonutName FROM
mddonut") {

        while (resultSet.next()) {
            listView.getItems().add(resultSet.getString("DonutName"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```
}
```

```
private Donut getDonutDetails(String name) {
```

```
    String query = "SELECT DonutID, DonutName, DonutDescription, DonutPrice FROM  
mddonut WHERE DonutName = ?";
```

```
    try (Connection connection = Connect.getConnection());
```

```
        PreparedStatement preparedStatement = connection.prepareStatement(query) {
```

```
        preparedStatement.setString(1, name);
```

```
        try (ResultSet resultSet = preparedStatement.executeQuery()) {
```

```
            if (resultSet.next()) {
```

```
                return new Donut(
```

```
                    resultSet.getString("DonutID"),
```

```
                    resultSet.getString("DonutName"),
```

```
                    resultSet.getString("DonutDescription"),
```

```
                    resultSet.getDouble("DonutPrice")
```

```
                );
```

```
            }
```

```
        }
```

```
    } catch (SQLException e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    return null;
```

```
}
```

```
private void saveToCart(String userId, String donutId, int quantity) {
```

```
    String query = "INSERT INTO cart (UserID, DonutID, Quantity) VALUES (?, ?, ?) ON  
DUPLICATE KEY UPDATE Quantity = Quantity + ?";
```

```

try (Connection connection = Connect.getConnection());
    PreparedStatement preparedStatement = connection.prepareStatement(query)) {

    preparedStatement.setString(1, userId);
    preparedStatement.setString(2, donutId);
    preparedStatement.setInt(3, quantity);
    preparedStatement.setInt(4, quantity);
    preparedStatement.executeUpdate();

} catch (SQLException e) {
    e.printStackTrace();
}
}

```

```

private String getLoggedInUsername(String userId) {
    String query = "SELECT Username FROM msuser WHERE UserID = ?";

    try (Connection connection = Connect.getConnection());
        PreparedStatement preparedStatement = connection.prepareStatement(query)) {

        preparedStatement.setString(1, userId);
        try (ResultSet resultSet = preparedStatement.executeQuery()) {
            if (resultSet.next()) {
                return resultSet.getString("Username");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```

        return "Guest"; // Default username if not found
    }

    private void showCartScene(Stage stage, String userId) {
        CartScene userHome = new CartScene(userId); // Pass the userId to UserHome
        try {
            userHome.start(stage);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

private void showAlert(Alert.AlertType type, String title, String content) {
    Alert alert = new Alert(type);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(content);
    alert.showAndWait();
}

```

```

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

```

Kodingan tersebut adalah aplikasi berbasis GUI (Graphical User Interface) menggunakan Java dan JavaFX. Aplikasi ini merupakan bagian dari sistem penjualan donat (Donut Store System) yang menyediakan fitur seperti menampilkan daftar donat, melihat detail donat, menambah donat ke keranjang (cart), dan logout. Berikut adalah penjelasan detail dari kodingan tersebut:

1. Struktur Program

Program ini terdiri dari beberapa bagian utama:

1. Variabel Utama

- `Map<String, Integer> cart`: Untuk menyimpan daftar donat yang ditambahkan ke keranjang, berisi pasangan DonutID dan jumlah.
- `username`: Menyimpan username pengguna yang sedang login.
- `userId`: Menyimpan ID pengguna yang sedang login, digunakan untuk mengidentifikasi pengguna di database.

2. Konstruktor

- `public CustomerHome(String userId)`: Konstruktor menerima parameter `userId` dari pengguna yang login.

3. Metode `start(Stage primaryStage)`

- Entry point dari aplikasi. Digunakan untuk mengatur layout utama aplikasi.

4. Layout

- `createHomeLayout(Stage primaryStage)`: Membuat layout utama aplikasi dengan komponen seperti menu bar, daftar donat, dan detail donat.

2. Komponen-Komponen Utama

a. Menu Bar

- Terdapat menu **Dashboard** dan **Logout**.
 - **Dashboard**: Berisi menu item Home (disable) dan Cart (menampilkan keranjang pengguna).
 - **Logout**: Berisi menu item Logout (keluar dari aplikasi).

b. ListView (Daftar Donat)

- **ListView donutListView**: Menampilkan daftar donat dari database menggunakan metode `populateDonutList`.

c. Detail Donat

- `VBox detailsBox`: Menampilkan detail donat seperti:
 - Nama donat (`Label donutNameLabel`).
 - Deskripsi donat (`Label donutDescriptionLabel`).
 - Harga donat (`Label donutPriceLabel`).
 - Spinner untuk memilih jumlah donat yang ingin ditambahkan ke keranjang.
 - Tombol **Add to Cart** untuk menambahkan donat ke keranjang.

d. Keranjang Belanja

- Data keranjang belanja dikelola di Map<String, Integer> cart dan disimpan di database menggunakan saveToCart.
-

3. Fitur Utama

a. Menampilkan Daftar Donat

- Metode populateDonutList mengambil data nama donat dari tabel mddonut di database.

b. Menampilkan Detail Donat

- Ketika pengguna memilih donat dari daftar, metode getDonutDetails mengambil detail donat dari database (nama, deskripsi, harga, dll.).

c. Menambah Donat ke Keranjang

- Tombol **Add to Cart** akan:
 1. Memastikan donat dipilih.
 2. Menambahkan donat ke keranjang lokal (cart) dengan merge untuk menjumlahkan jumlah.
 3. Memperbarui data keranjang di database menggunakan saveToCart.

d. Logout

- Metode logout akan:
 1. Menghapus data pengguna (userId dan username).
 2. Mengarahkan pengguna kembali ke halaman login menggunakan kelas Login.
-

4. Database Operasi

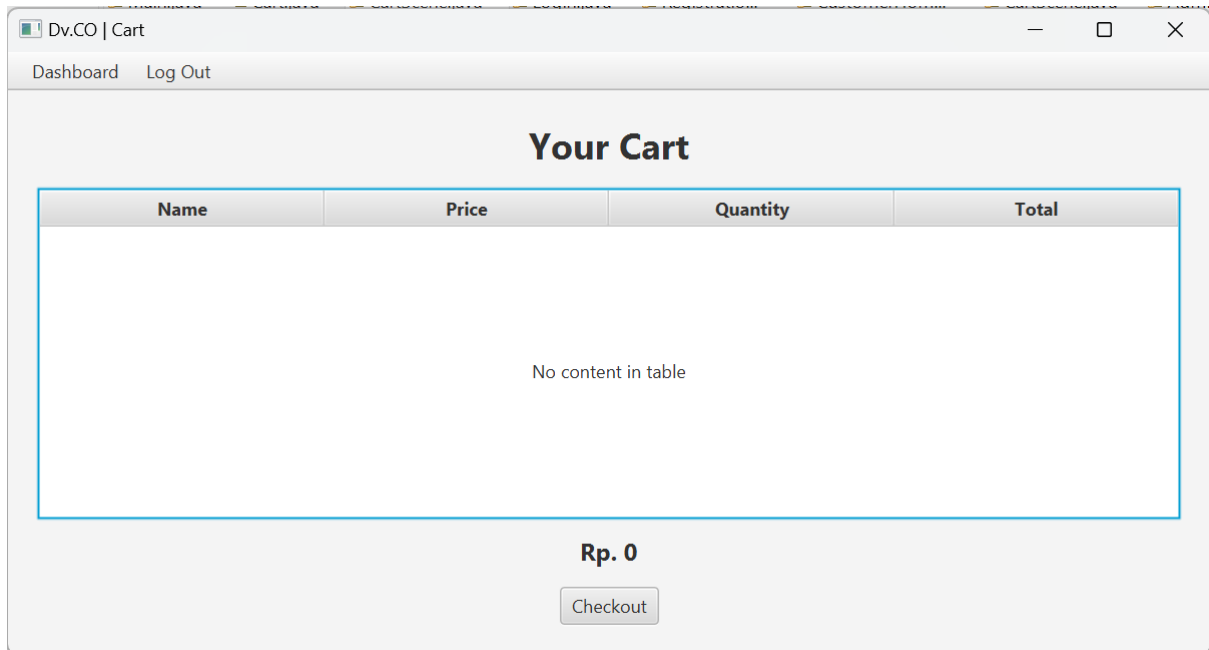
- **getLoggedInUsername(String userId)**
 - Mengambil username pengguna dari tabel msuser berdasarkan UserID.
 - **saveToCart(String userId, String donutId, int quantity)**
 - Menyimpan atau memperbarui jumlah donat ke tabel cart. Jika donat sudah ada, jumlahnya diperbarui dengan ON DUPLICATE KEY UPDATE.
 - **getDonutDetails(String name)**
 - Mengambil detail donat (DonutID, nama, deskripsi, harga) dari tabel mddonut berdasarkan nama donat.
-

5. Fungsi Utility

- **showAlert(Alert.AlertType type, String title, String content)**

- Menampilkan kotak dialog (alert) dengan pesan tertentu, misalnya pemberitahuan sukses atau error.
- **showCartScene(Stage stage, String userId)**
 - Berpindah ke scene keranjang belanja (CartScene) dengan melewati userId.

4.Cart Scene



package main;

import java.sql.*;

import javafx.application.Application;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

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.cell.PropertyValueFactory;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import model.Cart;
import util.Connect;

public class CartScene extends Application {

    Scene scene;

    BorderPane borderPaneForm;

    MenuBar menuBar;
    Menu dashboardMenu, logOutMenu;
    MenuItem homeBar, cartBar;
    MenuItem logoutBar;

    Label yourCartTitle, totalPriceLbl;
    TableView<Cart> tableView;
    Button checkOutBtn;

    private ObservableList<Cart> cartItems;
    private String userId; // Variable to store the logged-in user ID
    private String username;

    // Constructor to accept the userId

```

```
public CartScene(String userId) {  
    this.userId = userId;  
}  
  
void initialize() {  
    borderPaneForm = new BorderPane();  
  
    menuBar = new MenuBar();  
    dashboardMenu = new Menu("Dashboard");  
    logOutMenu = new Menu("Log Out");  
  
    homeBar = new MenuItem("Home");  
    cartBar = new MenuItem("Cart");  
    logoutBar = new MenuItem("Logout");  
    cartBar.setDisable(true);  
  
    cartItems = FXCollections.observableArrayList();  
    loadCartData(); // Load data from the database  
  
    yourCartTitle = new Label("Your Cart");  
    yourCartTitle.setStyle("-fx-font-size: 24px; -fx-font-weight: bold;");  
  
    tableView = new TableView<>();  
    tableView.setPlaceholder(new Label("No content in table"));  
    setTableView();  
  
    totalPriceLbl = new Label("Rp. " + calculateTotalPrice());  
    totalPriceLbl.setStyle("-fx-font-size: 16px; -fx-font-weight: bold;");  
  
    checkOutBtn = new Button("Checkout");
```

```
dashboardMenu.getItems().addAll(homeBar, cartBar);  
logoutMenu.getItems().addAll(logoutBar);  
menuBar.getMenus().addAll(dashboardMenu, logoutMenu);  
}
```

```
void setComponent(Stage stage) {  
    VBox vbox = new VBox(10, yourCartTitle, tableView, totalPriceLbl, checkoutBtn);  
    vbox.setPadding(new Insets(20));  
    vbox.setAlignment(Pos.CENTER);  
  
    if (cartItems.isEmpty()) {  
        totalPriceLbl.setText("Rp. 0");  
    } else {  
        totalPriceLbl.setText("Rp. " + calculateTotalPrice());  
    }  
    borderPaneForm.setTop(menuBar);  
    borderPaneForm.setCenter(vbox);  
  
    homeBar.setOnAction(e -> redirectToUserHome(stage, userId ));  
    logoutBar.setOnAction(e -> logout(stage));  
    checkoutBtn.setOnAction(e -> handleCheckout());  
  
}
```

```
private void logout(Stage primaryStage) {  
    // Clear the logged-in user data  
    this.userId = null;  
    this.username = null;
```

```
// Clear the current scene by setting a new empty scene

primaryStage.setScene(new Scene(new StackPane(), 800, 600)); // Empty scene to clear
the UI
```

```
// Redirect to Login scene

Login loginScene = new Login(); // Assuming you have a Login class

try {
    loginScene.start(primaryStage);
} catch (Exception ex) {
    ex.printStackTrace();
    showAlert(Alert.AlertType.ERROR, "Error", "Failed to load login screen.");
}
}
```

```
private void redirectToUserHome(Stage stage, String userId) {

    CustomerHome userHome = new CustomerHome(userId); // Pass the userId to
UserHome

    try {
        userHome.start(stage);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
```

```
private void setTableView() {

    TableColumn<Cart, String> nameColumn = new TableColumn<>("Name");
    nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));
    nameColumn.setPrefWidth(200);

    TableColumn<Cart, Integer> priceColumn = new TableColumn<>("Price");
```

```
priceColumn.setCellValueFactory(new PropertyValueFactory<>("price"));
priceColumn.setPrefWidth(100);
```

```
TableColumn<Cart, Integer> quantityColumn = new TableColumn<>("Quantity");
quantityColumn.setCellValueFactory(new PropertyValueFactory<>("quantity"));
quantityColumn.setPrefWidth(100);
```

```
TableColumn<Cart, Integer> totalColumn = new TableColumn<>("Total");
totalColumn.setCellValueFactory(new PropertyValueFactory<>("total"));
totalColumn.setPrefWidth(120);
```

```
tableView.setItems(cartItems); // Bind the cartItems list
tableView.getColumns().addAll(nameColumn, priceColumn, quantityColumn,
totalColumn);
}
```

```
private int calculateTotalPrice() {
    return cartItems.stream().mapToInt(Cart::getTotal).sum();
}
```

```
private void handleCheckout() {
    Cart selectedItem = tableView.getSelectionModel().getSelectedItem();

    // Validation: Check if an item is selected
    if (selectedItem == null) {
        showAlert(Alert.AlertType.ERROR, "Error", "No item selected for checkout.");
        return;
    }
```

```
// Generate Transaction ID
String transactionId = generateTransactionId();
```

```

try (Connection connection = Connect.getConnection()) {
    connection.setAutoCommit(false);

    // Insert into Transaction Header

    String insertHeaderQuery = "INSERT INTO transactionheader (TransactionID,
UserID) VALUES (?, ?)";

    try (PreparedStatement headerStmt =
connection.prepareStatement(insertHeaderQuery)) {
        headerStmt.setString(1, transactionId);
        headerStmt.setString(2, userId);
        headerStmt.executeUpdate();
    }

    // Get DonutID from selected item

    String donutIdQuery = "SELECT DonutID FROM mddonut WHERE DonutName =
?";

    String donutId;

    try (PreparedStatement donutStmt = connection.prepareStatement(donutIdQuery)) {
        donutStmt.setString(1, selectedItem.getName());
        try (ResultSet rs = donutStmt.executeQuery()) {
            if (rs.next()) {
                donutId = rs.getString("DonutID");
            } else {
                throw new SQLException("DonutID not found for selected item.");
            }
        }
    }

    // Insert into Transaction Detail

    String insertDetailQuery = "INSERT INTO transactiondetail (TransactionID,
DonutID, Quantity) VALUES (?, ?, ?)";

```

```

        try (PreparedStatement detailStmt =
connection.prepareStatement(insertDetailQuery)) {
            detailStmt.setString(1, transactionId);
            detailStmt.setString(2, donutId);
            detailStmt.setInt(3, selectedItem.getQuantity());
            detailStmt.executeUpdate();
        }

        // Remove item from Cart table
        String deleteCartQuery = "DELETE FROM cart WHERE UserID = ? AND DonutID
= ?";

        try (PreparedStatement deleteCartStmt =
connection.prepareStatement(deleteCartQuery)) {
            deleteCartStmt.setString(1, userId);
            deleteCartStmt.setString(2, donutId);
            deleteCartStmt.executeUpdate();
        }

        connection.commit();

        // Remove the item from the TableView and update the total price
        cartItems.remove(selectedItem);
        tableView.refresh();
        totalPriceLbl.setText("Rp. " + calculateTotalPrice());

        // Show success alert
        showAlert(Alert.AlertType.INFORMATION, "Checkout Success", "Transaction ID: "
+ transactionId);

    } catch (SQLException e) {
        e.printStackTrace();

        showAlert(Alert.AlertType.ERROR, "Checkout Error", "Failed to complete checkout.
Please try again.");
    }

```

```
}  
}
```

```
private String generateTransactionId() {
```

```
    String query = "SELECT COUNT(*) AS TransactionCount FROM transactionheader";
```

```
    try (Connection connection = Connect.getConnection();
```

```
        PreparedStatement stmt = connection.prepareStatement(query);
```

```
        ResultSet rs = stmt.executeQuery()) {
```

```
        int transactionIndex = 1; // Default to 1 if no transactions exist
```

```
        if (rs.next()) {
```

```
            transactionIndex = rs.getInt("TransactionCount") + 1;
```

```
        }
```

```
        return String.format("TR%03d", transactionIndex);
```

```
    } catch (SQLException e) {
```

```
        e.printStackTrace();
```

```
        throw new RuntimeException("Failed to generate transaction ID.");
```

```
    }
```

```
}
```

```
private void showAlert(Alert.AlertType alertType, String title, String content) {
```

```
    Alert alert = new Alert(alertType);
```

```
    alert.setTitle(title);
```

```
    alert.setContentText(content);
```

```
    alert.showAndWait();
```

```
}
```

```
private void loadCartData() {
```

```
    String query = "SELECT mddonut.DonutName, mddonut.DonutPrice, cart.Quantity "
```

+

```

"FROM cart " +
"JOIN mddonut ON cart.DonutID = mddonut.DonutID " +
"WHERE cart.UserID = ?";;

```

```

try (Connection connection = Connect.getConnection();
    PreparedStatement preparedStatement = connection.prepareStatement(query)) {

    preparedStatement.setString(1, userId);

    try (ResultSet resultSet = preparedStatement.executeQuery()) {
        cartItems.clear();
        while (resultSet.next()) {
            String name = resultSet.getString("DonutName");
            int price = resultSet.getInt("DonutPrice");
            int quantity = resultSet.getInt("Quantity");
            cartItems.add(new Cart(name, price, quantity));
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
    showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to load cart data.");
}
}

```

```

@Override
public void start(Stage primaryStage) throws Exception {
    initialize();
    setComponent(primaryStage);

    scene = new Scene(borderPaneForm, 800, 400);
}

```

```
primaryStage.setTitle("Dv.CO | Cart");  
primaryStage.setScene(scene);  
primaryStage.show();  
}
```

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

Kode di atas adalah sebuah aplikasi berbasis Java yang menggunakan **JavaFX** untuk antarmuka pengguna (UI) dan **JDBC** untuk menghubungkan aplikasi dengan basis data. Program ini merupakan implementasi dari sebuah halaman "Cart" (keranjang belanja) yang memungkinkan pengguna untuk melihat item di keranjang mereka, menghitung total harga, serta melakukan proses checkout. Berikut adalah penjelasan terperinci:

1. Struktur Kode

Kode ini terdiri dari beberapa bagian utama:

- **Variabel dan Properti Utama** Menyimpan elemen UI, data, dan properti aplikasi seperti:
 - `userId & username`: Menyimpan ID dan nama pengguna yang sedang login.
 - `cartItems`: Koleksi data item di keranjang yang ditampilkan pada tabel.
 - Komponen antarmuka seperti `menuBar`, `tableView`, `checkoutBtn`, dll.
- **Metode Utama**
 - **`initialize()`**: Menginisialisasi elemen UI, memuat data keranjang dari database, dan membuat menu navigasi.
 - **`setComponent(Stage stage)`**: Menempatkan elemen UI di dalam layout aplikasi dan menambahkan logika navigasi pada menu.
 - **`handleCheckout()`**: Menangani proses checkout termasuk validasi, penyimpanan transaksi ke database, dan penghapusan item dari keranjang.
 - **`loadCartData()`**: Memuat data keranjang dari database.
 - **`calculateTotalPrice()`**: Menghitung total harga item di keranjang.

- **logout():** Menangani proses logout dengan mengosongkan data pengguna dan mengarahkan ke halaman login.
 - **Main Method dan start(Stage primaryStage)**
 - Memulai aplikasi dengan memanggil metode initialize() dan menampilkan scene utama.
-

2. Fungsi dan Proses Utama

a. Tampilan dan Interaksi

- **Menu Navigasi**
 - Terdapat dua menu utama:
 - **Dashboard** (berisi menu ke "Home" dan "Cart").
 - **Log Out** (berisi opsi logout).
 - Menu cartBar dinonaktifkan karena pengguna sudah berada di halaman "Cart".
- **Tabel (TableView)**
 - Tabel menampilkan daftar item keranjang dengan kolom **Name, Price, Quantity,** dan **Total**.
 - Data tabel diambil dari objek cartItems yang berisi data dari database.
- **Label dan Tombol**
 - Label yourCartTitle menunjukkan judul halaman.
 - Label totalPriceLbl menampilkan total harga semua item di keranjang.
 - Tombol checkoutBtn digunakan untuk melakukan proses checkout.

b. Proses Checkout

Proses checkout melibatkan:

1. **Validasi**
 - Memastikan ada item yang dipilih sebelum memproses checkout.
2. **Penyimpanan Transaksi ke Database**
 - Membuat ID transaksi menggunakan metode generateTransactionId().
 - Menyimpan data transaksi ke tabel transactionheader dan transactiondetail.
 - Menghapus item yang telah di-checkout dari tabel cart.
3. **Pembaruan UI**
 - Menghapus item dari tabel UI dan memperbarui total harga.
4. **Pemberitahuan**

- Menampilkan dialog sukses atau error menggunakan showAlert().

c. Pengambilan dan Penyimpanan Data ke Database

- **Metode loadCartData()**
 - Mengambil data dari tabel cart dan tabel mddonut menggunakan query SQL.
 - Data dimasukkan ke dalam cartItems untuk ditampilkan di tabel.
- **Metode generateTransactionId()**
 - Menghitung jumlah transaksi yang ada di database untuk menghasilkan ID transaksi unik.

d. Logout dan Navigasi

- **Logout**
 - Menghapus data pengguna dari variabel dan mengganti scene ke halaman login.
 - **Navigasi**
 - Menu "Home" mengarahkan pengguna ke halaman utama menggunakan kelas CustomerHome.
-

3. Kelas dan Objek Lain yang Terkait

- **Cart**
 - Representasi data item keranjang, termasuk nama, harga, jumlah, dan total.
 - **Connect**
 - Utility untuk membuat koneksi ke database.
 - **CustomerHome dan Login**
 - Kelas lain dalam aplikasi ini yang bertanggung jawab untuk halaman utama pengguna dan halaman login.
-

4. Fitur Penting

- **Antarmuka Responsif**
 - Menggunakan layout seperti BorderPane dan VBox untuk mengatur elemen UI.
- **Integrasi Database**
 - Menggunakan JDBC untuk membaca, menulis, dan menghapus data di database.
- **Validasi**
 - Memastikan item dipilih sebelum checkout.

- **Kemudahan Navigasi**
 - Menggunakan menu dan tombol untuk memudahkan perpindahan antar halaman.
- **Dialog Pemberitahuan**
 - Memberikan feedback kepada pengguna saat terjadi kesalahan atau proses berhasil.

5. Potensi Pengembangan

- Menambahkan fitur seperti:
 - Memperbarui jumlah item di keranjang.
 - Menampilkan riwayat transaksi.
 - Dukungan untuk lebih banyak metode pembayaran.

Kode ini cukup modular dan mudah dikembangkan lebih lanjut sesuai kebutuhan.

5.Admin Home

Donut ID	Donut Name	Donut Description	Donut Price
DN001	Chocolate Donut	Donuts with chocolate sprinkles	10000
DN002	Strawberry Donut	Donuts with strawberry jam	12000
DN003	Cheese Donut	Donuts with cheese topping	13000
DN004	Peanut Donut	Donuts sprinkled with nuts	13000
DN005	Vanilla Donut	Donuts with vanilla flavor	11000

Donut Name:

Donut Description:

Donut Price:

package main;

```
import javafx.application.Application;
```

```
import javafx.beans.property.SimpleStringProperty;
```

```

import javafx.beans.property.StringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.sql.*;

import util.Connect;

public class AdminHome extends Application {

    private TableView<Donut> tableView;
    private TextField nameField, priceField;
    private TextArea descriptionArea;
    private String username; // Fetched from the database
    private ObservableList<Donut> donutData;
    private String userId; // Variable to store the logged-in user ID

    // Database connection setup
    private Connection connection;

    // Constructor to accept the userId
    public AdminHome(String userId) throws SQLException {
        this.userId = userId;
        connection = Connect.getConnection();
    }

```

@Override

```
public void start(Stage primaryStage) {  
    fetchUsername();  
    donutData = getDonutData();  
  
    // Menu bar setup  
    MenuBar menuBar = new MenuBar();  
    Menu menu = new Menu("Logout");  
    MenuItem logoutMenuItem = new MenuItem("Logout");  
    menu.getItems().add(logoutMenuItem);  
    menuBar.getMenus().add(menu);  
  
    // Logout event handler  
    logoutMenuItem.setOnAction(e -> {  
        logout(primaryStage);  
    });  
  
    // Labels  
    Label welcomeLabel = new Label("Hello, " + username);  
    welcomeLabel.setStyle("-fx-font-size: 24px; -fx-font-weight : bold;");  
    Label activeDonutLabel = new Label("Active Donut:");  
    Label nameLabel = new Label("Donut Name:");  
    Label descriptionLabel = new Label("Donut Description:");  
    Label priceLabel = new Label("Donut Price:");  
  
    // TableView  
    tableView = new TableView<>();  
    TableColumn<Donut, String> idColumn = new TableColumn<>("Donut ID");  
    idColumn.setCellValueFactory(data -> data.getValue().idProperty());
```

```
TableColumn<Donut, String> nameColumn = new TableColumn<>("Donut Name");  
nameColumn.setCellValueFactory(data -> data.getValue().nameProperty());
```

```
TableColumn<Donut, String> descriptionColumn = new TableColumn<>("Donut  
Description");  
descriptionColumn.setCellValueFactory(data -> data.getValue().descriptionProperty());
```

```
TableColumn<Donut, String> priceColumn = new TableColumn<>("Donut Price");  
priceColumn.setCellValueFactory(data -> data.getValue().priceProperty());
```

```
tableView.getColumns().addAll(idColumn, nameColumn, descriptionColumn,  
priceColumn);
```

```
tableView.setItems(donutData);
```

```
// TextFields and TextArea
```

```
nameField = new TextField();
```

```
priceField = new TextField();
```

```
descriptionArea = new TextArea();
```

```
descriptionArea.setPrefHeight(60);
```

```
// Event: Populate fields when a row is selected
```

```
tableView.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection,  
newSelection) -> {
```

```
    if (newSelection != null) {
```

```
        nameField.setText(newSelection.getName());
```

```
        descriptionArea.setText(newSelection.getDescription());
```

```
        priceField.setText(newSelection.getPrice());
```

```
    }
```

```
});
```

```
// Buttons
```

```
Button addButton = new Button("Add Donut");  
addButton.setOnAction(e -> handleAddDonut());
```

```
Button updateButton = new Button("Update Donut");  
updateButton.setOnAction(e -> handleUpdateDonut());
```

```
Button deleteButton = new Button("Delete Donut");  
deleteButton.setOnAction(e -> handleDeleteDonut());
```

```
// Layout
```

```
GridPane formLayout = new GridPane();  
formLayout.setPadding(new Insets(10));  
formLayout.setHgap(10);  
formLayout.setVgap(10);
```

```
formLayout.add(nameLabel, 0, 0);  
formLayout.add(nameField, 1, 0);  
formLayout.add(descriptionLabel, 0, 1);  
formLayout.add(descriptionArea, 1, 1);  
formLayout.add(priceLabel, 0, 2);  
formLayout.add(priceField, 1, 2);
```

```
HBox buttonLayout = new HBox(10, addButton, updateButton, deleteButton);  
formLayout.add(buttonLayout, 1, 3);
```

```
VBox layout = new VBox(10, menuBar, welcomeLabel, activeDonutLabel, tableView,  
formLayout);
```

```
layout.setPadding(new Insets(10));
```

```
// Scene
```

```

Scene scene = new Scene(layout, 800, 600);
primaryStage.setScene(scene);
primaryStage.setTitle("Home Scene (Admin)");
primaryStage.show();
}

```

```

private void logout(Stage primaryStage) {

```

```

    // Clear the logged-in user data

```

```

    this.userId = null;

```

```

    this.username = null;

```

```

    // Clear the current scene by setting a new empty scene

```

```

    primaryStage.setScene(new Scene(new StackPane(), 800, 600)); // Empty scene to clear
the UI

```

```

    // Redirect to Login scene

```

```

    Login loginScene = new Login(); // Assuming you have a Login class

```

```

    try {

```

```

        loginScene.start(primaryStage);

```

```

    } catch (Exception ex) {

```

```

        ex.printStackTrace();

```

```

        showAlert(Alert.AlertType.ERROR, "Error", "Failed to load login screen.");

```

```

    }

```

```

}

```

```

private void fetchUsername() {

```

```

    String query = "SELECT username FROM msuser WHERE UserID = ?";

```

```

    try (PreparedStatement stmt = connection.prepareStatement(query)) {

```

```

        stmt.setString(1, userId);

```

```

        ResultSet rs = stmt.executeQuery();

```

```

        if (rs.next()) {

```

```

        username = rs.getString("username");
    }
} catch (SQLException e) {
    e.printStackTrace();
    showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to fetch username.");
}
}

```

```

private ObservableList<Donut> getDonutData() {
    ObservableList<Donut> data = FXCollections.observableArrayList();
    String query = "SELECT * FROM mddonut";

    try (Statement stmt = connection.createStatement(); ResultSet rs =
stmt.executeQuery(query)) {
        while (rs.next()) {
            data.add(new Donut(
                rs.getString("DonutID"),
                rs.getString("DonutName"),
                rs.getString("DonutDescription"),
                rs.getString("DonutPrice")
            ));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to fetch donuts
data.");
    }
    return data;
}

```

```

private void handleAddDonut() {
    String name = nameField.getText().trim();

```

```
String description = descriptionArea.getText().trim();
```

```
String price = priceField.getText().trim();
```

```
if (name.isEmpty() || description.isEmpty() || price.isEmpty()) {  
    showAlert(Alert.AlertType.ERROR, "Validation Error", "All fields must be filled.");  
    return;  
}
```

```
String newId = generateDonutId();
```

```
String query = "INSERT INTO mddonut (DonutID, DonutName, DonutDescription,  
DonutPrice) VALUES (?, ?, ?, ?)";
```

```
try (PreparedStatement stmt = connection.prepareStatement(query)) {  
    stmt.setString(1, newId);  
    stmt.setString(2, name);  
    stmt.setString(3, description);  
    stmt.setString(4, price);  
    stmt.executeUpdate();  
    donutData.add(new Donut(newId, name, description, price));  
    showAlert(Alert.AlertType.INFORMATION, "Success", "Donut added  
successfully!");  
    nameField.clear();  
    descriptionArea.clear();  
    priceField.clear();  
} catch (SQLException e) {  
    e.printStackTrace();  
    showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to add donut.");  
}  
}
```

```
private void handleUpdateDonut() {
```

```
    Donut selectedDonut = tableView.getSelectionModel().getSelectedItem();
```

```

    if (selectedDonut == null) {
        showAlert(Alert.AlertType.ERROR, "Validation Error", "No donut selected for
update.");
        return;
    }

    String name = nameField.getText().trim();
    String description = descriptionArea.getText().trim();
    String price = priceField.getText().trim();

    if (name.isEmpty() || description.isEmpty() || price.isEmpty()) {
        showAlert(Alert.AlertType.ERROR, "Validation Error", "All fields must be filled.");
        return;
    }

    String query = "UPDATE mddonut SET DonutName = ?, DonutDescription = ?,
DonutPrice = ? WHERE DonutID = ?";

    try (PreparedStatement stmt = connection.prepareStatement(query)) {
        stmt.setString(1, name);
        stmt.setString(2, description);
        stmt.setString(3, price);
        stmt.setString(4, selectedDonut.getId());
        stmt.executeUpdate();

        selectedDonut.setName(name);
        selectedDonut.setDescription(description);
        selectedDonut.setPrice(price);

        tableView.refresh();

        showAlert(Alert.AlertType.INFORMATION, "Success", "Donut updated
successfully!");
    }

```

```

    } catch (SQLException e) {
        e.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to update donut.");
    }
}

```

```

private void handleDeleteDonut() {
    Donut selectedDonut = tableView.getSelectionModel().getSelectedItem();
    if (selectedDonut == null) {
        showAlert(Alert.AlertType.ERROR, "Validation Error", "No donut selected for deletion.");
        return;
    }
}

```

```

String query = "DELETE FROM mddonut WHERE DonutID = ?";
try (PreparedStatement stmt = connection.prepareStatement(query)) {
    stmt.setString(1, selectedDonut.getId());
    stmt.executeUpdate();
    donutData.remove(selectedDonut);
    showAlert(Alert.AlertType.INFORMATION, "Success", "Donut deleted successfully!");
} catch (SQLException e) {
    e.printStackTrace();
    showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to delete donut.");
}
}

```

```

private String generateDonutId() {
    // Query to get the maximum DonutID from the database
    String query = "SELECT MAX(CAST(SUBSTRING(DonutID, 3) AS UNSIGNED)) AS maxId FROM mddonut";
}

```

```

    try (Statement stmt = connection.createStatement(); ResultSet rs =
stmt.executeQuery(query)) {
        if (rs.next()) {
            int nextId = rs.getInt("maxId") + 1; // Get the next ID number
            return String.format("DN%03d", nextId); // Format it to be like DN001, DN002,
etc.
        } else {
            // If no rows in the table, start with DN001
            return "DN001";
        }
    } catch (SQLException e) {
        e.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to generate Donut
ID.");
        return "DN001"; // Fallback ID in case of an error
    }
}

```

```

private void showAlert(Alert.AlertType alertType, String title, String message) {
    Alert alert = new Alert(alertType);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

```

//Donut class (model)

```

public static class Donut {
    private final SimpleStringProperty id;
    private final SimpleStringProperty name;
    private final SimpleStringProperty description;
}

```

```
private final SimpleStringProperty price;
```

```
public Donut(String id, String name, String description, String price) {  
    this.id = new SimpleStringProperty(id);  
    this.name = new SimpleStringProperty(name);  
    this.description = new SimpleStringProperty(description);  
    this.price = new SimpleStringProperty(price);  
}
```

```
public StringProperty idProperty() { return id; }  
public StringProperty nameProperty() { return name; }  
public StringProperty descriptionProperty() { return description; }  
public StringProperty priceProperty() { return price; }
```

```
public String getId() { return id.get(); }  
public String getName() { return name.get(); }  
public String getDescription() { return description.get(); }  
public String getPrice() { return price.get(); }
```

```
public void setName(String name) { this.name.set(name); }  
public void setDescription(String description) { this.description.set(description); }  
public void setPrice(String price) { this.price.set(price); }  
}
```

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

```
}
```

Kodingan ini merupakan implementasi aplikasi desktop menggunakan JavaFX untuk mengelola data **donat** yang disimpan dalam database. Berikut adalah penjelasan elemen penting dari kode:

1. Fungsi Utama

Aplikasi ini dirancang untuk **Admin Home** di mana seorang admin dapat:

- Melihat daftar donat (dari database).
- Menambah donat baru.
- Memperbarui data donat yang ada.
- Menghapus donat.
- Logout dari aplikasi.

2. Struktur Utama

- **Kelas AdminHome:** Kelas utama aplikasi JavaFX.
- **Konstruktor:**
 - Menerima `userId` sebagai parameter untuk mengidentifikasi admin yang sedang login.
 - Menginisialisasi koneksi ke database menggunakan utilitas `Connect`.
- **Metode start:**
 - Mengatur antarmuka pengguna (UI) aplikasi menggunakan JavaFX.

3. Komponen Antarmuka

a. Menu Bar

- Menu "Logout" untuk keluar dari aplikasi dan kembali ke halaman login.

b. TableView

- Menampilkan data donat dalam bentuk tabel dengan kolom:
 - Donut ID
 - Donut Name
 - Donut Description
 - Donut Price
- Data diambil dari database menggunakan metode `getDonutData`.

c. Form Input

- **TextField** dan **TextArea** untuk input nama, deskripsi, dan harga donat.
- Ketika baris di tabel dipilih, field input akan diisi dengan data dari baris tersebut.

d. Tombol Aksi

- **Add Donut:** Menambah data donat baru ke database.
- **Update Donut:** Memperbarui data donat yang dipilih.
- **Delete Donut:** Menghapus data donat yang dipilih.

e. Layout

- **VBox:** Mengatur tata letak secara vertikal.
- **GridPane:** Mengatur tata letak form input.
- **HBox:** Mengatur tata letak tombol secara horizontal.

4. Interaksi dengan Database

a. Pengambilan Data Donat

Metode `getDonutData` menjalankan query SQL:

```
SELECT * FROM mddonut;
```

Untuk mengambil semua data donat dari tabel `mddonut`.

b. Tambah Donat

Metode `handleAddDonut`:

1. Memvalidasi input.
2. Menggunakan `PreparedStatement` untuk menambah data ke database:

```
INSERT INTO mddonut (DonutID, DonutName, DonutDescription, DonutPrice) VALUES (?, ?, ?, ?);
```

c. Update Donat

Metode `handleUpdateDonut`:

1. Memvalidasi input.
2. Menggunakan `PreparedStatement` untuk memperbarui data donat yang dipilih:

```
UPDATE mddonut SET DonutName = ?, DonutDescription = ?, DonutPrice = ? WHERE DonutID = ?;
```

d. Hapus Donat

Metode `handleDeleteDonut`:

1. Menghapus data donat yang dipilih menggunakan query:

```
DELETE FROM mddonut WHERE DonutID = ?;
```

e. Mengambil Nama Admin

Metode `fetchUsername` mengambil nama pengguna admin berdasarkan `userId`:

```
SELECT username FROM msuser WHERE UserID = ?;
```

f. Menghasilkan ID Donat

Metode generateDonutId:

1. Mengambil ID maksimum dari tabel:

```
SELECT MAX(CAST(SUBSTRING(DonutID, 3) AS UNSIGNED)) AS maxId FROM mddonut;
```

2. Menghasilkan ID berikutnya dalam format DN001, DN002, dll.
-

5. Model Donut

Model ini digunakan untuk merepresentasikan data donat:

- Properti: id, name, description, price.
 - Metode properti digunakan untuk binding data antara database dan TableView.
-

6. Logout dan Navigasi

- **Logout:**
 - Membersihkan data pengguna yang sedang login.
 - Mengatur ulang scene aplikasi ke halaman login.
-

7. Alert

Metode showAlert digunakan untuk menampilkan pesan kesalahan atau keberhasilan pada pengguna menggunakan JavaFX Alert.

6.Cart

```
1 package model;
2
3 public class Cart {
4     private String name;
5     private Integer price;
6     private Integer quantity;
7     private Integer total;
8
9     public Cart(String name, Integer price, Integer quantity) {
10         this.name = name;
11         this.price = price;
12         this.quantity = quantity;
13         this.total = price * quantity;
14     }
15
16     public String getName() {
17         return name;
18     }
19
20     public Integer getPrice() {
21         return price;
22     }
23
24     public Integer getQuantity() {
25         return quantity;
26     }
27
28     public Integer getTotal() {
29         return total;
30     }
31 }
32
```

Kode di atas mendefinisikan sebuah class Java bernama Cart, yang dapat digunakan untuk merepresentasikan data terkait sebuah item dalam keranjang belanja. Berikut adalah penjelasan dari setiap bagian kode:

Paket

package model;

- Menunjukkan bahwa class ini berada dalam paket model. Paket digunakan untuk mengelompokkan kelas-kelas yang berkaitan secara logis dan mempermudah pengelolaan kode.

Deklarasi Atribut

private String name;

private Integer price;

private Integer quantity;

private Integer total;

- **name:** Menyimpan nama item dalam keranjang belanja.
- **price:** Menyimpan harga satuan dari item.

- **quantity:** Menyimpan jumlah item yang dibeli.
- **total:** Menyimpan total harga item yang dihitung berdasarkan $\text{price} * \text{quantity}$.

Semua atribut bersifat **private**, yang berarti hanya dapat diakses langsung dari dalam class Cart. Untuk mengaksesnya dari luar, getter disediakan.

Constructor

```
public Cart(String name, Integer price, Integer quantity) {
    this.name = name;
    this.price = price;
    this.quantity = quantity;
    this.total = price * quantity;
}
```

- Constructor adalah metode khusus yang digunakan untuk menginisialisasi objek baru dari class Cart.
- Parameter:
 - **name:** Nama item.
 - **price:** Harga satuan item.
 - **quantity:** Jumlah item.
- Constructor mengatur nilai atribut name, price, dan quantity menggunakan nilai parameter.
- Atribut total dihitung secara otomatis menggunakan operasi perkalian: $\text{price} * \text{quantity}$.

Getter Methods

Getter digunakan untuk mengambil nilai atribut yang bersifat private.

1. **getName():**
2. `public String getName() {`
3. `return name;`
4. `}`

Mengembalikan nama item.

5. **getPrice():**
6. `public Integer getPrice() {`
7. `return price;`
8. `}`

Mengembalikan harga satuan item.

9. **getQuantity():**

```
10. public Integer getQuantity() {  
11.     return quantity;  
12. }
```

Mengembalikan jumlah item.

13. **getTotal():**

```
14. public Integer getTotal() {  
15.     return total;  
16. }
```

Mengembalikan total harga item (price * quantity).

Fungsi Class

Class Cart ini berguna untuk:

1. Menyimpan informasi tentang item dalam keranjang belanja.
2. Mengelola data dengan keamanan yang lebih baik melalui penggunaan atribut private dan getter.
3. Menghitung total harga item secara otomatis saat objek dibuat.

Contoh Penggunaan

```
Cart cartItem = new Cart("Laptop", 10000000, 2);  
System.out.println("Item: " + cartItem.getName());  
System.out.println("Price: " + cartItem.getPrice());  
System.out.println("Quantity: " + cartItem.getQuantity());  
System.out.println("Total: " + cartItem.getTotal());
```

Output:

Item: Laptop

Price: 10000000

Quantity: 2

Total: 20000000

Kode ini cocok untuk digunakan dalam aplikasi seperti sistem e-commerce untuk menangani data keranjang belanja.

7.Donut

```

1 package model;
2
3 public class Donut {
4     private final String donutID;
5     private final String donutName;
6     private final String donutDescription;
7     private final double donutPrice;
8
9     public Donut(String donutID, String donutName, String donutDescription, double donutPrice) {
10         this.donutID = donutID;
11         this.donutName = donutName;
12         this.donutDescription = donutDescription;
13         this.donutPrice = donutPrice;
14     }
15
16     public String getDonutID() {
17         return donutID;
18     }
19
20     public String getDonutName() {
21         return donutName;
22     }
23
24     public String getDonutDescription() {
25         return donutDescription;
26     }
27
28     public double getDonutPrice() {
29         return donutPrice;
30     }
31 }
32

```

Kode yang Anda berikan adalah implementasi dari sebuah kelas Donut dalam bahasa pemrograman Java. Kelas ini mendefinisikan objek donat dengan beberapa atribut dan menyediakan metode untuk mengakses nilai-nilai atribut tersebut. Berikut adalah penjelasan rinci mengenai kode tersebut:

1. Deklarasi Kelas

```
public class Donut {
```

- Kelas ini bernama Donut. Kata kunci `public` menandakan bahwa kelas ini dapat diakses dari kelas lain dalam paket yang berbeda.

2. Atribut (Field)

```
private final String donutID;
```

```
private final String donutName;
```

```
private final String donutDescription;
```

```
private final double donutPrice;
```

- **donutID:** Menyimpan ID unik untuk setiap donat. Tipe datanya adalah `String`.
- **donutName:** Menyimpan nama donat. Tipe datanya adalah `String`.
- **donutDescription:** Menyimpan deskripsi donat. Tipe datanya adalah `String`.
- **donutPrice:** Menyimpan harga donat. Tipe datanya adalah `double`.

Keempat atribut ini dideklarasikan dengan kata kunci `private` yang berarti hanya bisa diakses dalam kelas Donut dan tidak dapat diakses langsung dari luar kelas. Kata kunci `final` berarti nilai-nilai atribut tersebut tidak dapat diubah setelah objek Donut dibuat.

3. Konstruktor

```

public Donut(String donutID, String donutName, String donutDescription, double donutPrice) {
    this.donutID = donutID;
    this.donutName = donutName;
    this.donutDescription = donutDescription;
    this.donutPrice = donutPrice;
}

```

- Konstruktor ini digunakan untuk membuat objek Donut baru. Konstruktor ini menerima empat parameter: donutID, donutName, donutDescription, dan donutPrice.
- Kata kunci this digunakan untuk membedakan antara parameter konstruktor dan atribut kelas yang memiliki nama yang sama. Misalnya, this.donutID merujuk pada atribut kelas donutID, sedangkan donutID merujuk pada parameter yang diterima oleh konstruktor.

4. Getter Methods

```

public String getDonutID() {
    return donutID;
}

```

```

public String getDonutName() {
    return donutName;
}

```

```

public String getDonutDescription() {
    return donutDescription;
}

```

```

public double getDonutPrice() {
    return donutPrice;
}

```

- Keempat metode ini adalah metode "getter" yang digunakan untuk mengakses nilai atribut dari objek Donut.
- Setiap metode mengembalikan nilai atribut terkait. Misalnya, getDonutID() mengembalikan nilai dari atribut donutID.

8.User

```
1 package model;
2
3 public class User {
4     private final String userId;
5     private final String role;
6
7     public User(String userId, String role) {
8         this.userId = userId;
9         this.role = role;
10    }
11
12    public String getUserId() {
13        return userId;
14    }
15
16    public String getRole() {
17        return role;
18    }
19 }
```

Kode yang Anda berikan adalah definisi kelas Java bernama User yang digunakan untuk merepresentasikan seorang pengguna dengan dua atribut utama: `userId` dan `role`. Berikut adalah penjelasan bagian per bagian:

1. Deklarasi Kelas dan Package

`package model;`

- Kode ini menunjukkan bahwa kelas User berada dalam package model. Package digunakan untuk mengelompokkan kelas-kelas yang memiliki fungsionalitas serupa atau terkait.

2. Deklarasi Kelas

`public class User {`

- Kelas User dideklarasikan sebagai `public`, artinya kelas ini dapat diakses dari luar package-nya. Ini berarti objek dari kelas User bisa dibuat di kelas lain yang berada di package berbeda.

3. Atribut Private

`private final String userId;`

`private final String role;`

- Ada dua atribut dalam kelas ini, yaitu `userId` dan `role`, keduanya bertipe `String`.
- Kata kunci `private` menunjukkan bahwa atribut ini hanya bisa diakses di dalam kelas User itu sendiri, tidak dapat diakses langsung dari kelas lain.

- Kata kunci final berarti bahwa nilai dari atribut ini tidak dapat diubah setelah objek User dibuat (nilai atribut bersifat tetap).

4. Konstruktor

```
public User(String userId, String role) {
    this.userId = userId;
    this.role = role;
}
```

- Konstruktor ini digunakan untuk membuat objek User dengan memberikan dua parameter: userId dan role.
- this.userId = userId; dan this.role = role; digunakan untuk menginisialisasi atribut kelas dengan nilai yang diterima dari parameter konstruktor.
- Kata kunci this digunakan untuk membedakan antara parameter konstruktor dan atribut kelas yang memiliki nama yang sama.

5. Getter Method

```
public String getUserId() {
    return userId;
}
```

```
public String getRole() {
    return role;
}
```

- Kedua metode ini adalah getter untuk atribut userId dan role. Mereka mengembalikan nilai dari atribut yang sesuai.
- getUserId() mengembalikan nilai userId, dan getRole() mengembalikan nilai role.

9.Connect

```
package util;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
```

```
public final class Connect {
```

```
    private static final String DB_URL = "jdbc:mysql://localhost:3306/dvco"; // Ganti  
sesuai dengan database Anda
```

```
    private static final String DB_USER = "root"; // Ganti sesuai username database
```

```
    private static final String DB_PASSWORD = ""; // Ganti sesuai password database
```

```
    private Connect() {
```

```
        // Prevent instantiation
```

```
    }
```

```
    /**
```

```
        * Get a new database connection.
```

```
        *
```

```
        * @return Connection object to the database
```

```
        * @throws SQLException if there is a problem with the connection
```

```
    */
```

```
    public static Connection getConnection() throws SQLException {
```

```
        try {
```

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

```
        } catch (ClassNotFoundException e) {
```

```
            throw new SQLException("JDBC Driver not found!", e);
```

```
        }
```

```
        return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
```

```
    }
```

```
    /**
```

```
        * Close a Connection safely.
```

```
        *
```

```
        * @param connection the Connection object to close
```

```

*/

public static void close(Connection connection) {
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

/**
 * Close a PreparedStatement safely.
 *
 * @param preparedStatement the PreparedStatement object to close
 */
public static void close(PreparedStatement preparedStatement) {
    if (preparedStatement != null) {
        try {
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

/**
 * Close a ResultSet safely.
 *
 * @param resultSet the ResultSet object to close

```

```

*/

public static void close(ResultSet resultSet) {
    if (resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}

```

Kode yang diberikan adalah sebuah kelas Connect yang bertujuan untuk mengelola koneksi ke database MySQL di dalam aplikasi Java. Kelas ini menggunakan JDBC (Java Database Connectivity) untuk menghubungkan dan mengelola sumber daya database secara efisien. Berikut adalah penjelasan detail tentang setiap bagian kode:

1. Deklarasi Konstanta Database:

```

private static final String DB_URL = "jdbc:mysql://localhost:3306/dvco";

private static final String DB_USER = "root";

private static final String DB_PASSWORD = "";

```

- **DB_URL:** Merupakan URL untuk menghubungkan ke database MySQL. Pada contoh ini, database yang digunakan bernama dvco, dan berada di server lokal (localhost) pada port default MySQL yaitu 3306.
- **DB_USER:** Username yang digunakan untuk masuk ke database. Di sini, username yang digunakan adalah root, yang biasanya adalah user default di MySQL.
- **DB_PASSWORD:** Password untuk user yang digunakan. Dalam hal ini, password dikosongkan.

2. Konstruktor Private:

```

private Connect() {
    // Prevent instantiation
}

```

- Konstruktor ini bersifat **private** untuk mencegah pembuatan instance (objek) dari kelas Connect. Ini membuat kelas ini hanya dapat digunakan melalui metode statis saja, untuk mencegah adanya objek dari kelas ini yang dibuat (karena tujuan kelas ini adalah sebagai utility untuk manajemen koneksi database).

3. Metode getConnection()

```
public static Connection getConnection() throws SQLException {  
    try {  
        Class.forName("com.mysql.cj.jdbc.Driver");  
    } catch (ClassNotFoundException e) {  
        throw new SQLException("JDBC Driver not found!", e);  
    }  
    return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);  
}
```

- **getConnection()** adalah metode statis yang digunakan untuk mendapatkan koneksi ke database.
- **Class.forName("com.mysql.cj.jdbc.Driver")**: Memuat driver JDBC MySQL secara dinamis. Hal ini penting agar JDBC dapat memahami cara berkomunikasi dengan MySQL.
- Jika driver tidak ditemukan, maka **ClassNotFoundException** akan ditangkap, dan akan dilemparkan sebagai **SQLException**.
- Setelah itu, menggunakan `DriverManager.getConnection()`, metode ini mengembalikan objek `Connection` yang dapat digunakan untuk berinteraksi dengan database MySQL.

4. Metode close() untuk Menutup Koneksi:

```
public static void close(Connection connection) {  
    if (connection != null) {  
        try {  
            connection.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

- **close(Connection connection)**: Menutup objek `Connection` yang diberikan jika koneksi tidak null. Ini untuk melepaskan sumber daya yang digunakan oleh koneksi tersebut.
- Jika terjadi kesalahan saat menutup, kesalahan tersebut akan dicetak ke console (dengan `e.printStackTrace()`).

5. Metode close() untuk Menutup PreparedStatement:

```

public static void close(PreparedStatement preparedStatement) {
    if (preparedStatement != null) {
        try {
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

- **close(PreparedStatement preparedStatement):** Menutup objek PreparedStatement jika tidak null. Seperti halnya koneksi, objek ini perlu ditutup setelah digunakan untuk melepaskan sumber daya.

6. Metode close() untuk Menutup ResultSet:

```

public static void close(ResultSet resultSet) {
    if (resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

- **close(ResultSet resultSet):** Menutup objek ResultSet jika tidak null. ResultSet berisi hasil dari query yang dieksekusi di database, dan perlu ditutup setelah tidak digunakan lagi.

Ringkasan Fungsi Kelas Connect:

Kelas ini bertanggung jawab untuk:

- Mengelola koneksi ke database MySQL.
- Menghindari pembuatan objek dari kelas ini dengan konstruktor yang private.
- Menyediakan metode statis untuk membuka koneksi (getConnection()) dan menutup sumber daya seperti koneksi, PreparedStatement, dan ResultSet setelah selesai digunakan.

Tujuan dari desain ini adalah untuk mempermudah pengelolaan koneksi database, menghindari kebocoran sumber daya, dan membuat kode lebih bersih dan mudah dipelihara.