Laporan Praktikum Week 4 Solid


Nama:Ahmad firnanda setiawan

Nim:1201230001

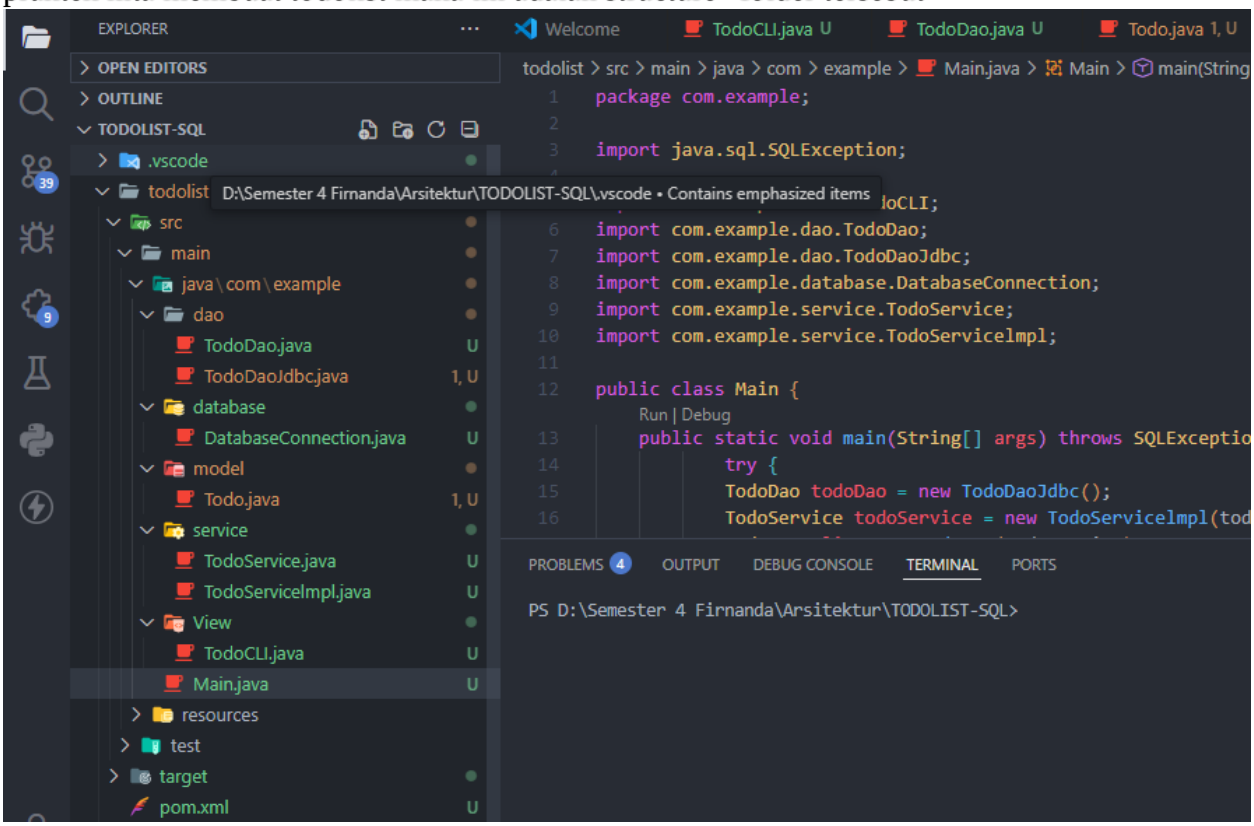
Kelas:SE-06

DOSEN PENGAMPU

ACHMAD MUZAKKI, S.Kom., M.Kom.

Jadi dalam week kemarin kita mempelajari SOLID yak langsung aja dimulai.

1.baik yang pertama kita setup terlebih dahulu membuat beberapa folder dan kemarin waktu praktek kita membuat todolist maka ini adalah structure" folder tersebut




Yak dalam beberapa folder itu terdapat beberapa file yang berbeda beda untuk fungsi"nnya itu sendiri kita pertama bahas pada folder database aja yang mana dalam folder database itu terdapat fole database connection yang mana dalam file tersebut berisikan source code untuk menyambungkan data yang akan kita inputkan akan masuk kedalam mysql yang dimana mysql tersebut yang akan menyimpan data kita dan utnuk source code koneksinnya sendiri itu seperti ini.

```java
package com.example.database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/todolist-sol";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "";

    private static Connection connection = null;
```

```
   public static Connection getConnection() throws SQLException{
    try{
        Class.forName("com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
    }catch(ClassNotFoundException e) {
        throw new SQLException("My sql jdbc tidak ditemukan", e);
    }
    return connection;
   }


   public static void closeConnection() throws SQLException {
    if (connection != null && !connection.isClosed()) {
        connection.close();
    }
   }
}
```

Yak itu adalah source code untuk koneksinnya sendiri untuk menyambungkannya itu kita tinggal sesuaikan nama database yang ada pada database yang kita buat contohnnya punya saya adalah databasennya Bernama todolist-sol untuk nama atau password sesuaikan dengan punya kalian ya teman" .oh iyaa teman" jangan lupa kita mengubah file pom xmlnnya untuk source codennya sendiri itu seperti ini ya teman".

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>todolist</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>
</dependencies>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
```

```
    </properties>

</project>
```

2.yak setelah kita membahas untuk koneksi dengan databasennya kita langsung ke folder modelnnya ya untuk dimodel sendiri ini sebenarnnya simple teman" karena dalam folder model ini value atau table yang ada ditabasennya contoh source code saya sendiri itu seperti ini.

```
package com.example.model;

public class Todo {
    private int id;
    private String title;
    private String description;
    private boolean completed;

    public Todo () {}

    public Todo(String title, String description) {
        this.title = title;
        this.description = description;
        this.completed = completed;
    }

    //Getter dan setter
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }
    public void   setTitle(String title) {
        this.title= title;
    }

    public String getDescription() {
        return description;
    }
    public void   setDescription(String description) {
        this.description= description;
    }
```

```
    public boolean isCompleted() {
        return completed;
    }
    public void   setCompleted(boolean completed) {
        this.completed= completed;
    }

    @Override
    public String toString(){
    return id + "_" + title + (completed ? "[V]" : "[]") +
    "\n" + description;
    }
}
```

Dan yaps itu adalah contoh sourccecode saya untuk melkukan set dan get yang nantinnya akan dipanggil ke kelas lain.

3.yak setelah kita melakukan eksekusi pada folder model akan mengeksuksi pada folder dao yang mana dalam folder dao ini ada 2 file yang fungsinnya berbeda yang satu untuk interfacennya yang satu untuk memanggil fungsi dari kelas" lain contoh source codennya sendiri yang untuk memanggil kelas kelas lain adalah itu seperti ini.

```
package com.example.dao;

import java.util.ArrayList;
import java.util.List;
import java.sql.Array;

import com.example.database.DatabaseConnection;
import com.example.model.Todo;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class TodoDaoJdbc implements TodoDao {

    @Override
   // CREATE
     public void create(Todo todo) throws Exception {
      String sql = "INSERT INTO todos (title, description, completed) VALUES (?,
?, ?)";

      try (Connection conn = DatabaseConnection.getConnection();
```

```
            PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

         stmt.setString(1, todo.getTitle());
         stmt.setString(2, todo.getDescription());
         stmt.setBoolean(3, todo.isCompleted());

         int affectedRows = stmt.executeUpdate();

         if (affectedRows == 0) {
            throw new Exception("Gagal membuat todo, tidak ada baris yang
terpengaruh.");
         }

         try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
               todo.setId(generatedKeys.getInt(1));
            } else {
               throw new Exception("Gagal membuat todo, tidak ada ID yang
diperoleh.");
            }
         }
      }
   }
    @Override
    public void update(Todo todo) throws Exception {
        String sql = "UPDATE todos SET title = ?, description = ?, completed = ?
WHERE id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
              PreparedStatement stmt = conn.prepareStatement(sql)) {

           stmt.setString(1, todo.getTitle());
           stmt.setString(2, todo.getDescription());
           stmt.setBoolean(3, todo.isCompleted());
           stmt.setInt(4, todo.getId());

           int affectedRows = stmt.executeUpdate();

           if (affectedRows == 0) {
              throw new Exception("Gagal mengupdate todo, tidak ada baris yang
terpengaruh.");
           }
        }
     } {
```

```
    }
    @Override
    public void delete (int id) throws Exception {
        String sql = "DELETE FROM todos WHERE id = ?";

      try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {

         stmt.setInt(1, id);

         int affectedRows = stmt.executeUpdate();

         if (affectedRows == 0) {
            throw new Exception("Gagal menghapus todo, tidak ada baris yang
terpengaruh.");
         }
      }
    }

    @Override
    public Todo findByid (int id) throws Exception {
        String sql = "SELECT * FROM todos WHERE id = ?";

      try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {

         stmt.setInt(1, id);

         try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
               return extraceTodoFromResultset(rs);
            } else {
               return null;
            }
         }
      }
    }

    public List <Todo> findAll () throws Exception {
        String sql = "SELECT * FROM todos";
      List<Todo> todos = new ArrayList<>();

      try (Connection conn = DatabaseConnection.getConnection();
            Statement stmt = conn.createStatement();
```

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

         while (rs.next()) {
            todos.add(extraceTodoFromResultset(rs));
         }
      }

      return todos;
   }

   private Todo extraceTodoFromResultset(ResultSet set) throws Exception {
      Todo todo = new Todo();
      todo.setId(set.getInt("id"));
      todo.setTitle(set.getString("title"));
      todo.setDescription(set.getString("description"));
      todo.setCompleted(set.getBoolean("completed"));

      return todo;
   }
}
```

Yak setelah melakukan pembuataun fungsi dalam folder tododaojdbc tersebut langsung kita panggil untuk memunculkan interfacennya pada file TodoDao yang mana dalam file Tododao itu kita tinggal manggil fungsinnya saja dan itu sudah menerapkan prinsip ocp principle dan utnuk source codennya sendiri itu seperti ini.

```
package com.example.dao;

import java.util.List;
import com.example.model.Todo;


//Menerapkan ocp prinsiple
//Interface ini memungkinkan untuk estensi tanpa modifikasi
public interface TodoDao {
void create(Todo todo) throws Exception;
void update(Todo todo) throws Exception;
void delete(int id) throws Exception;

Todo findByid(int id) throws Exception;

List<Todo> findAll () throws Exception;

}
```

Dan yaps cukup simple bukan karena itu tinggal kita panggil aja dari folder model yang berisikan nama fiennya yaitu Todo.

4.dan yak setelah kita sudah melakukan beberapa tahapan ini kita melakukan servicenya yaitu pada folder service yang mana dalam folder service sendiri ini berisi terkait fungsi" yang akan dipanggil dalam file tododaojdbc dan dalam file service sendiri ini ada 2 filejuga yang berbeda yang mana ada todoservice untuk interface dan ada todoservicelmpl dan dalam todoservicemlpl ini itu berisakan yang ada pada model tadi kita tadi kita ekseuksi disini untuk contoh source codennya seperti ini.

```
package com.example.service;
import java.util.List;
import com.example.model.Todo;
import com.example.dao.TodoDao;


public class TodoServicelmpl implements TodoService {
    private final TodoDao todoDao;

    public TodoServicelmpl(TodoDao todoDao) {
        this.todoDao = todoDao ;
    }

    @Override
    public void addTodo(String title, String description) throws Exception {
        Todo todo = new Todo(title, description);
        todoDao.create(todo);
    }
    @Override
    public void updateTodo(Todo todo) throws Exception {
        todoDao.update(todo);
    }
    @Override
    public void deleteTodo(int id)  throws Exception {
        todoDao.delete(id);
    }
    @Override
    public Todo getTodoByid(int id) throws Exception {
        return todoDao.findByid(id);
    }
    @Override
    public List <Todo> getallTodos( ) throws Exception {
        return todoDao.findAll();
    }
    @Override
```

```
    public void markAscompleted(int id) throws Exception {
        Todo todo = todoDao.findByid(id);

        if (todo != null) {
            todo.setCompleted(true);
            todoDao.update(todo);
        }else{
            throw new Exception("todo dengan id" + id + "tidak ditemukan");
        }
    }
}
```

Ya begitulah kurang lebihnnya.untuk todoservice untuk interfacennya itu source codennya seperti ini

```
package com.example.service;

import java.util.List;
import com.example.model.Todo;

//Service Interface - Menerapkan Ocp
public interface TodoService {

        void addTodo(String title, String description) throws Exception;
        void updateTodo(Todo todo) throws Exception;
        void deleteTodo(int id) throws Exception;
        Todo getTodoByid(int id) throws Exception;

        List<Todo> getallTodos () throws Exception;

        void markAscompleted(int id) throws Exception;


}
```

5.setelah kita membahas folder” diatas ini adalah folder view yang mana dalam folder view ini yang akan memunculkan tampilan yang ada pada terminal untuk source codennya sendiri itu seperti ini yang berada pada file TodoCli.

```
// VIEW CLI
package com.example.View;

import com.example.model.Todo;
import com.example.service.TodoService;
```

```java
import java.util.List;
import java.util.Scanner;

// 7. CLI Interface - Menerapkan SRP
public class TodoCLI {
   private final TodoService todoService;
   private final Scanner scanner;

   public TodoCLI(TodoService todoService) {
      this.todoService = todoService;
      this.scanner = new Scanner(System.in);
   }

   public void start() {
      boolean running = true;
      while (running) {
         displayMenu();
         int choice = getUserChoice();

         try {
            switch (choice) {
               case 1:
                  addTodo();
                  break;
               case 2:
                  listTodos();
                  break;
               case 3:
                  markTodoAsCompleted();
                  break;
               case 4:
                  updateTodo();
                  break;
               case 5:
                  deleteTodo();
                  break;
               case 6:
                  running = false;
                  System.out.println("Terima kasih telah menggunakan TodoList
CLI!");
                  break;
               default:
                  System.out.println("Pilihan tidak valid. Silakan coba lagi.");
            }
         } catch (Exception e) {
```

```
            System.out.println("Error: " + e.getMessage());
        }
    }
    scanner.close();
}

private void displayMenu() {
    System.out.println("\n===== TodoList CLI =====");
    System.out.println("1. Tambah Todo");
    System.out.println("2. Lihat Semua Todo");
    System.out.println("3. Tandai Todo Selesai");
    System.out.println("4. Update Todo");
    System.out.println("5. Hapus Todo");
    System.out.println("6. Keluar");
    System.out.print("Pilih menu: ");
}

private int getUserChoice() {
    try {
        return Integer.parseInt(scanner.nextLine());
    } catch (NumberFormatException e) {
        return -1;
    }
}

private void addTodo() throws Exception {
    System.out.print("Masukkan judul todo: ");
    String title = scanner.nextLine();

    System.out.print("Masukkan deskripsi todo: ");
    String description = scanner.nextLine();

    todoService.addTodo(title, description);
    System.out.println("Todo berhasil ditambahkan!");
}

private void listTodos() throws Exception {
    List<Todo> todos = todoService.getallTodos();

    if (todos.isEmpty()) {
        System.out.println("Tidak ada todo.");
        return;
    }

    System.out.println("\n===== Daftar Todo =====");
```

```
        for (Todo todo : todos) {
            System.out.println(todo);
        }
    }

    private void markTodoAsCompleted() throws Exception {
        System.out.print("Masukkan ID todo yang akan ditandai selesai: ");
        int id = Integer.parseInt(scanner.nextLine());

        todoService.markAscompleted(id);
        System.out.println("Todo berhasil ditandai selesai!");
    }

    private void updateTodo() throws Exception {
        System.out.print("Masukkan ID todo yang akan diupdate: ");
        int id = Integer.parseInt(scanner.nextLine());

        Todo todo = todoService.getTodoByid(id);
        if (todo == null) {
            System.out.println("Todo dengan ID " + id + " tidak ditemukan.");
            return;
        }

        System.out.print("Masukkan judul baru (kosongkan jika tidak ingin
mengubah): ");
        String title = scanner.nextLine();
        if (!title.isEmpty()) {
            todo.setTitle(title);
        }

        System.out.print("Masukkan deskripsi baru (kosongkan jika tidak ingin
mengubah): ");
        String description = scanner.nextLine();
        if (!description.isEmpty()) {
            todo.setDescription(description);
        }

        todoService.updateTodo(todo);
        System.out.println("Todo berhasil diupdate!");
    }

    private void deleteTodo() throws Exception {
        System.out.print("Masukkan ID todo yang akan dihapus: ");
        int id = Integer.parseInt(scanner.nextLine());
```

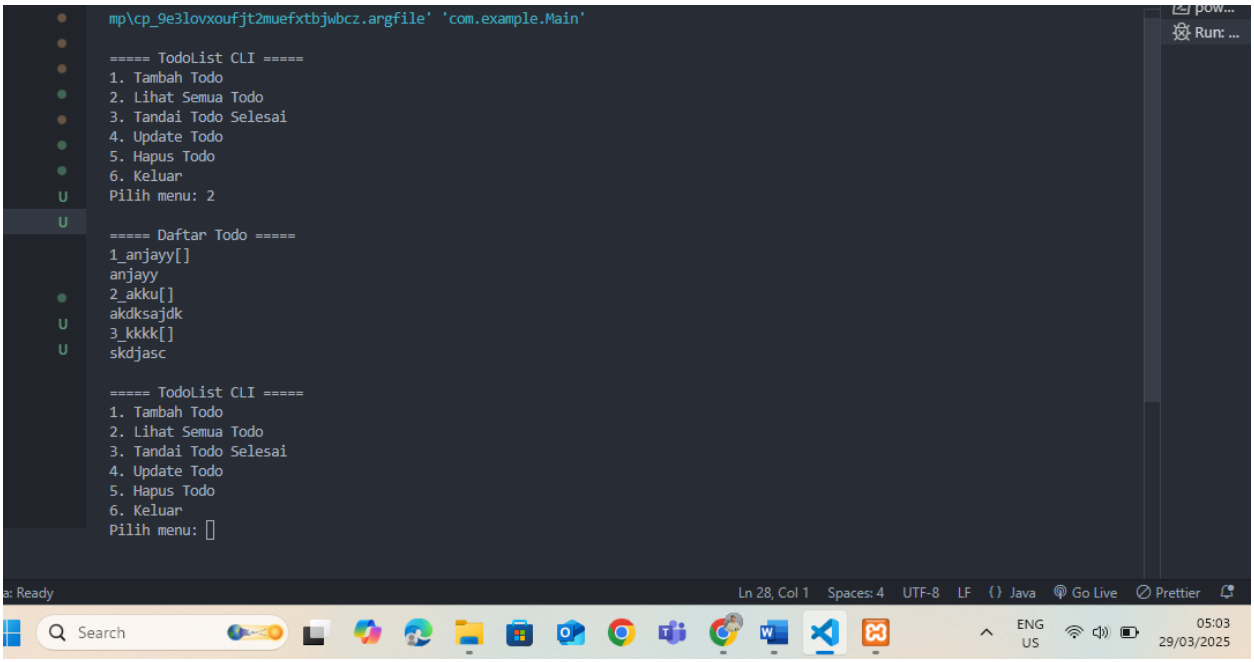
```
        todoService.deleteTodo(id);
        System.out.println("Todo berhasil dihapus!");
    }
}
```

Ya itu kita membuat tampilannya dalam source tersebut juga kita memanggil fungsi" dari beberapa kelas tadi.

Dan yaps untuk finalnnya adalah berada pada file main yang mana ini yang akan menjalankan semua file" atau  kelas" yang sudah kita buatsemua kita akan panggil dalam main ini untuk source codennya sendiri itu seperti ini.

```
package com.example;

import java.sql.SQLException;

import com.example.View.TodoCLI;
import com.example.dao.TodoDao;
import com.example.dao.TodoDaoJdbc;
import com.example.database.DatabaseConnection;
import com.example.service.TodoService;
import com.example.service.TodoServiceImpl;

public class Main {
    public static void main(String[] args) throws SQLException {
            try {
            TodoDao todoDao = new TodoDaoJdbc();
            TodoService todoService = new TodoServiceImpl(todoDao);
            TodoCLI cli = new TodoCLI(todoService);
            cli.start();

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            DatabaseConnection.closeConnection();
        }
    }
    }
```

Yak itu adalah main class kita yang akan menjalankan program kita yang mana kita memanggil semua kelasnnya bahkan juga finalnnya nantik ketika data sudah di inputkan akan diakhir I denga  closeconnection untuk mengakhiri program untuk hasil outputnnya sendiri seperti ini.




Dan yaps untuk hasil outputnnya seperti ini sekain terima kasih sudah saya coba aman semua kok.