Laporan Praktikum 6 – Prinsip SOLID

Nama : Cania Nabilatul Adawah

NIM : 2403102

Kelas : D3TI2C

Mata Kuliah : Pemrograman Berbasis Objek

1. Single Responsibility Principle (S)
   "Sebuah class harus memiliki satu dan hanya satu tanggung jawab"

   a. Case Study 1 Melalui Pembuatan Kode Program

      a) Main.java

```
package srp;

public class Main {
public static void main(String[] args) {
        Persegi p = new Persegi(5);
        System.out.println("Luas: " + PenghitungPersegi.hitungLuas(p));
    }
}
```

      b) PenghitungPersegi.java

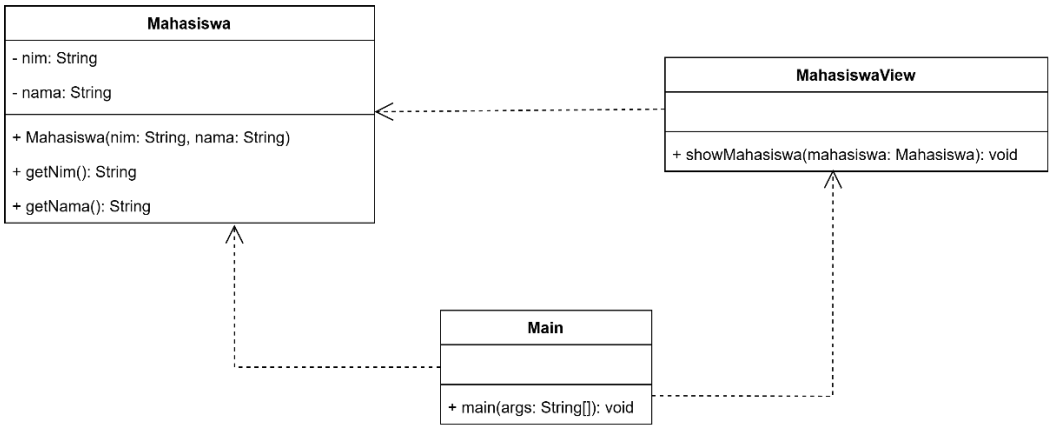
```
package srp;

public class PenghitungPersegi {
    public static int hitungLuas(Persegi persegi) {
        return persegi.getSisi() * persegi.getSisi();
    }
}
```

      c) Persegi.java

```
package srp;

public class Persegi {
    private int sisi;

    public Persegi(int sisi) {
        this.sisi = sisi;
```

```
    }

    public int getSisi() {
        return sisi;
    }
}
```

b. Case Study 2 Melalui Pembuatan Class Diagram



2. Open-closed Principle (O)
   "Sebuah class harus terbuka terhadap penambahan (extension), tetapi tertutup terhadap modifikasi"

   a. Case Study 1 Melalui Pembuatan Kode Program
      a) Cash.java

```
package ocp;

public class Cash extends TipePembayaran {
    @Override
    public void memprosesPembayaran() {
        System.out.println("Proses pembayaran dengan Cash");
    }
}
```

      b) Debit.java

```
package ocp;

public class Debit extends TipePembayaran {
    @Override
```

```
    public void memprosesPembayaran() {
        System.out.println("Proses pembayaran dengan Debit");
    }
}
```

c) Kredit.java

```
package ocp;

public class Kredit extends TipePembayaran {
    @Override
    public void memprosesPembayaran() {
        System.out.println("Proses pembayaran dengan Kredit");
    }
}
```

d) Main.java

```
package ocp;

public class Main {
    public static void main(String[] args) {
        PembayaranCustomer pembayaran = new PembayaranCustomer();

        pembayaran.menerimaPembayaran(new Debit());
        pembayaran.menerimaPembayaran(new Kredit());
        pembayaran.menerimaPembayaran(new Cash());
    }
}
```

e) PembayaranCustomer.java

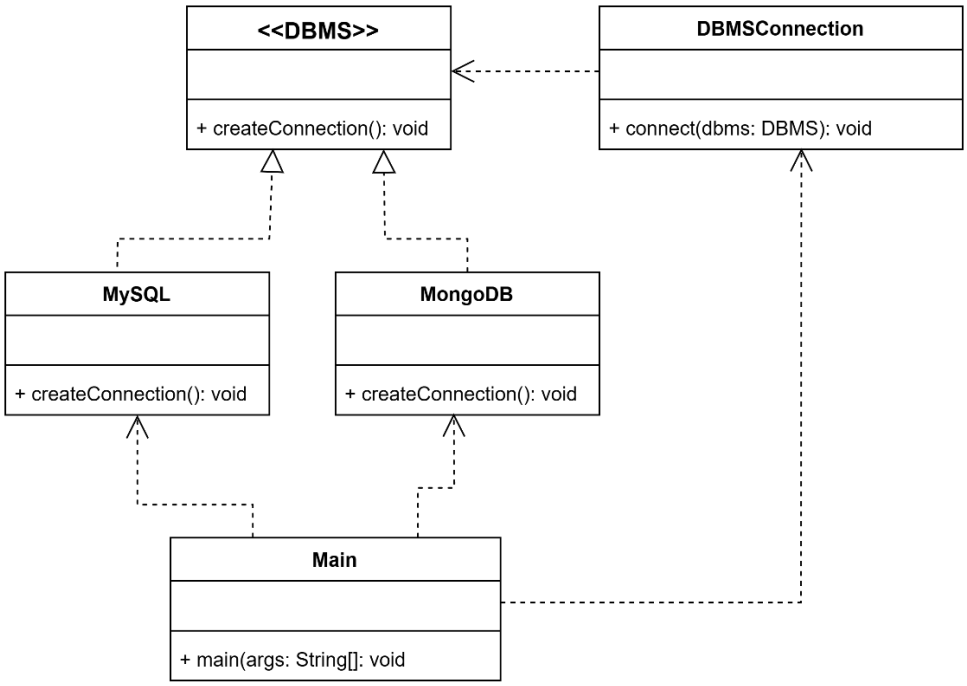
```
package ocp;

public class PembayaranCustomer {
    public void menerimaPembayaran(TipePembayaran tipe) {
        tipe.memprosesPembayaran();
    }
}
```

f) TipePembayaran.java

```
package ocp;
```

```
public abstract class TipePembayaran {
    public abstract void memprosesPembayaran();
}
```

b. Case Study 2 Melalui Pembuatan Class Diagram



3. Liskov Substitution Principle (L)
   "Object superclass harus dapat sepenuhnya diganti (replaceable) oleh object subclass-nya tanpa merusak program"

   a. Case Study 1 Melalui Pembuatan Kode Program
      a) Instagram.java

```
package lsp;

public class Instagram implements PostMediaManager {

    @Override
    public void chat() {
        System.out.println("Fitur Chat");
    }

    @Override
    public void sendPhotosAndVideos() {
```

```
        System.out.println("Fitur pengriman foto dan video");
    }

    @Override
    public void publishPost() {
        System.out.println("Fitur posting feed");
    }

}
```

b) Main.java

```
package lsp;

public class Main {

    public static void main(String[] args) {
        WhatsApp wa = new WhatsApp();
        Instagram ig = new Instagram();

        wa.chat();
        ig.chat();
        wa.sendPhotosAndVideos();
        ig.sendPhotosAndVideos();
        wa.callGroupVideo();
        ig.publishPost();
    }

}
```

c) PostMediaManager.java

```
package lsp;

public interface PostMediaManager extends SocialMedia{

    public void publishPost();

}
```

d) SocialMedia.java

```
package lsp;

public interface SocialMedia {
```

```
    public void chat();
    public void sendPhotosAndVideos();

}
```

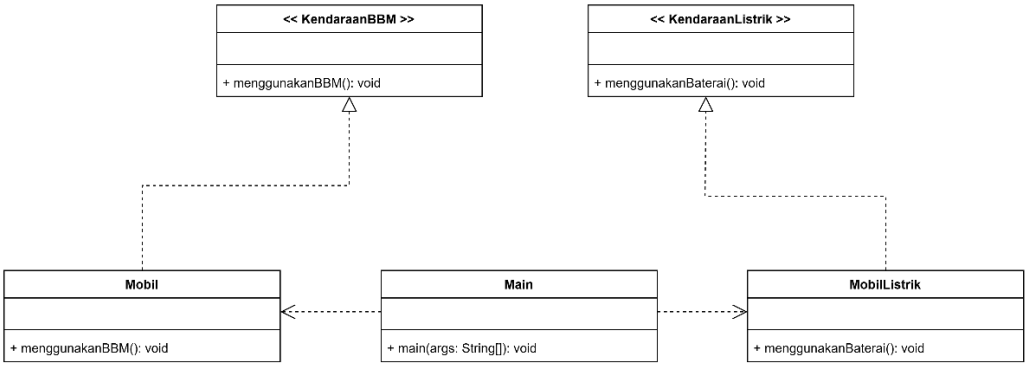
e) VideoGroupManager.java

```
package lsp;

public interface VideoGroupManager extends SocialMedia{

    public void callGroupVideo();

}
```

f) WhatsApp.java

```
package lsp;

public class WhatsApp implements VideoGroupManager {

    @Override
    public void chat() {
        System.out.println("Fitur Chat");
    }

   @Override
    public void sendPhotosAndVideos() {
        System.out.println("Fitur pengiriman foto dan video");    }
    }

    @Override
    public void callGroupVideo() {
        System.out.println("Fitur telepon video grup");
    }

}
```

b. Case Study 2 Melalui Pembuatan Class Diagram



4. Interface Segregation Principle (I)
   "Tidak ada class yang dipaksa (forced) untuk mengimplementasikan method yang tidak terpakai / tidak digunakan"

   a. Case Study 1 Melalui Pembuatan Kode Program
      a) Kubus.java

```java
package isp;

public class Kubus implements Shape3Dimension,Shape2Dimension {
    @Override
    public void calculateArea() {
        System.out.println("Menghitung luas permukaan kubus...");
    }

    @Override
    public void calculateVolume() {
        System.out.println("Menghitung volume kubus...");
    }
}
```

      b) Main.java

```java
package isp;
public class Main {
    public static void main(String[] args) {
        Kubus kubus = new Kubus();
        kubus.calculateArea();
        kubus.calculateVolume();

        Persegi persegi = new Persegi();
        persegi.calculateArea();
    }
```

```
}
```

c) Persegi.java

```
package isp;

public class Persegi implements Shape2Dimension {
    @Override
    public void calculateArea() {
        System.out.println("Menghitung luas persegi...");
    }
}
```

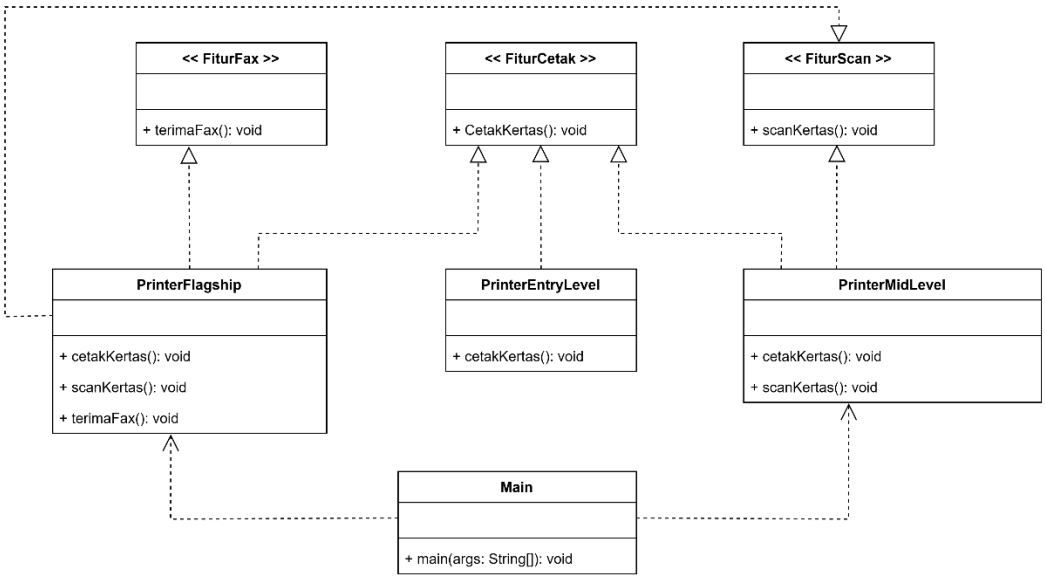
d) Shape2Dimension.java

```
package isp;

public interface Shape2Dimension {
    public void calculateArea();
}
```

e) Shape3Dimension.java

```
package isp;

public interface Shape3Dimension {
    public void calculateVolume();
}
```

b. Case Study 2 Melalui Pembuatan Class Diagram



5. Dependency Inversion Principle (D)
   "Modul tingkat tinggi tidak boleh bergantung pada modul tingkat rendah, keduanya harus bergantung pada abstraksi"

   a. Case Study 1 Melalui Pembuatan Kode Program
      a) Database.java

```
package dip;

public interface Database {
    public void insert(String data);

}
```

      b) Main.java

```
package dip;

public class Main {
    public static void main(String[] args) {

        Database db = new MySQL();
        UserService userService = new UserService(db);
        userService.registerUser("cania@gmail.com");
    }
}
```

c) MySQL.java

```
package dip;

public class MySQL implements Database {

    @Override
    public void insert(String data) {
        System.out.println("Insert data ke MySQL: " + data);
    }
}
```

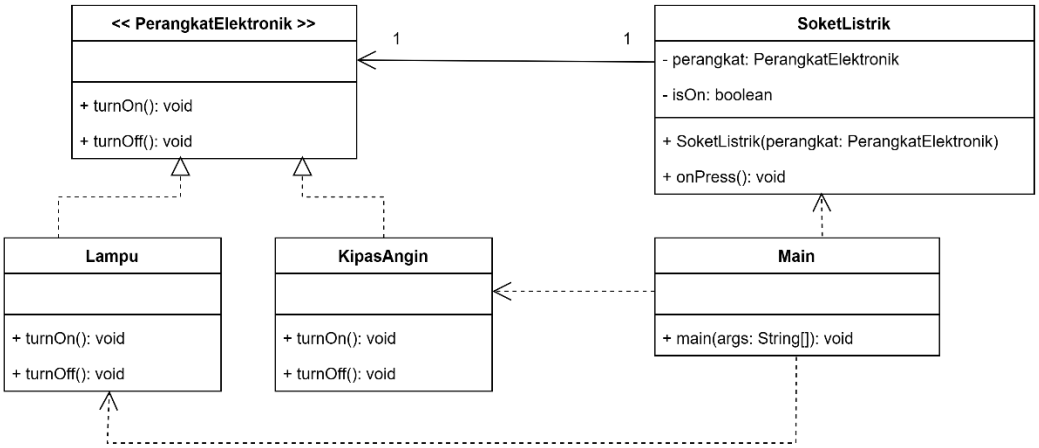
d) PostgreSQL.java

```
package dip;

public class PostgreSQL implements Database {

    @Override
    public void insert(String data) {
        System.out.println("Insert data ke PostgreSQL: " + data);
    }
}
```

e) UserService.java

```
package dip;

public class UserService {

    private Database database;

    public UserService(Database database) {
        this.database = database;
    }

    public void registerUser(String user) {
        System.out.println("Registering user: " + user);
        database.insert(user);
    }
}
```

b. Case Study 2 Melalui Pembuatan Class Diagram


<< PerangkatElektronik >>
+ turnOn(): void
+ turnOff(): void
1
1
SoketListrik
- perangkat: PerangkatElektronik
- isOn: boolean
+ SoketListrik(perangkat: PerangkatElektronik)
+ onPress(): void
Lampu
+ turnOn(): void
+ turnOff(): void
KipasAngin
+ turnOn(): void
+ turnOff(): void
Main
+ main(args: String[]): void