PERTEMUAN KEENAM

Abstract dan Interface

Prepared by Adi Wahyu Pribadi

Abstract

Class abstract adalah class yang masih dalam bentuk abstract. Karena bentuknya masih abstract, dia tidak bisa dibuat langsung menjadi objek. Sebuah class agar dapat disebut class abstract setidaknya memiliki satu atau lebih method abstract. Objek hanya bisa dibuat dari non-abstract class maka suatu abstract class haruslah diturunkan dimana pada suatu subclass tersebut berisi implementasi dari abstract method yang ada pada Super/Parent Class-nya.

Method abstract adalah method yang tidak memiliki implementasi atau tidak ada bentuk konkritnya (hanya deklarasi method).

Mengapa Menggunakan Abstract?

Class abstrak belum bisa digunakan secara langsung. Karena itu, agar class abstrak dapat digunakan, maka ia harus dibuat bentuk konkritnya. Cara membuat class abstrak menjadi konkrit adalah dengan membuat implementasi dari method-method yang masih abstrak. Ini bisa kita lakukan dengan pewarisan (inheritance).

Class abstrak biasanya digunakan sebagai class induk dari class-class yang lain. Class anak akan membuat versi konkrit dari class abstrak.

Mengapa sih class harus dibuat menjadi abstrak? Pada suatu kondisi tertentu, class induk tidak ingin kita buat sebagai objek. Bisa jadi kode methodnya belum jelas mau bagaimana implementasinya.

Sebagai perumpamaan terdapat class Database yang digunakan untuk keperluan koneksi dan query database. Implementasi dari class Database tersebut adalah sesuai dari aplikasi database yang digunakan seperti MySQL, PostgreSQL, dan lain sebagainya.

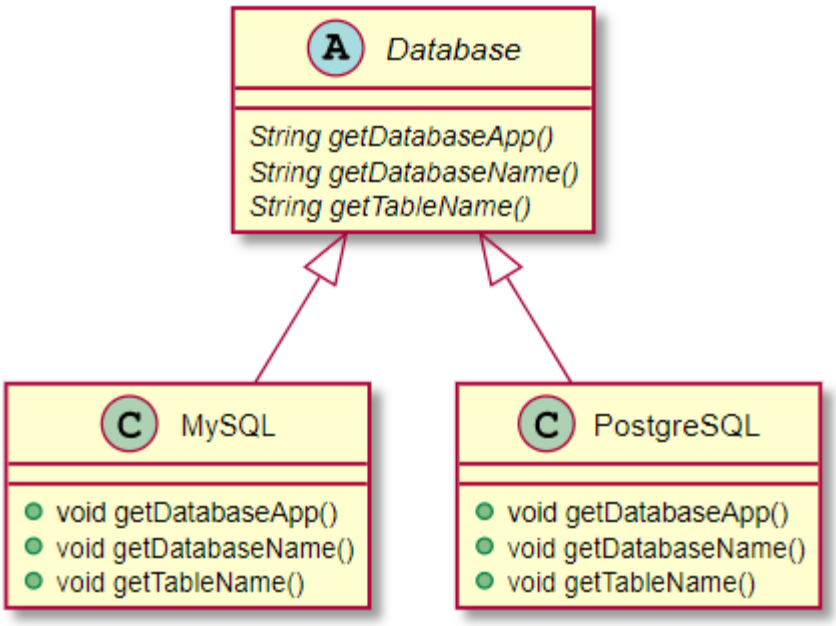
```
class Database {
    String getTableName() {
        return null;
    }
}

class MySQL extends Database {
    @Override
    String getTableName() {
        return "SELECT table_name FROM DATABASE";
    }
}

public class App {
    public static void main(String[] args) throws Exception {
        Database db = new Database();
        System.out.println(db.getTableName());
        MySQL mysql = new MySQL();
        System.out.println(mysql.getTableName());
    }
}
```

Output

```
null
SELECT table_name FROM DATABASE
```

Dari output terlihat bahwa, jika membuat objek dari class Database, belum ada implementasi bagaimana cara mendapatkan nama tabel. Namun pada class MySQL yang merupakan implementasi class Database, baru bisa mendapatkan nama tabel.

Solusinya adalah dengan membuat abstract class Database, sehingga ketika akan menggunakan database harus memilih implementasi dari aplikasi Databasenya seperti MySQL atau PostgreSQL. Lihat Class Diagram berikut. Abstract class terdapat huruf A besar dan nama abstract class ditulis miring, begitu juga dengan abstract method ditulis miring.



Implementasinya adalah:

Abstract Class Database

```
abstract class Database {
    abstract String getDatabaseApp();
    abstract String getDatabaseName();
    abstract String getTableName();
}
```

Class MySQL

```
class MySQL extends Database {
    String getDatabaseApp() {
        return "MySQL DATABASE Server";
    }
    String getDatabaseName() {
        return "SIAK";
    }
    String getTableName() {
        return "Table Mahasiswa";
    }
```

```
}
```

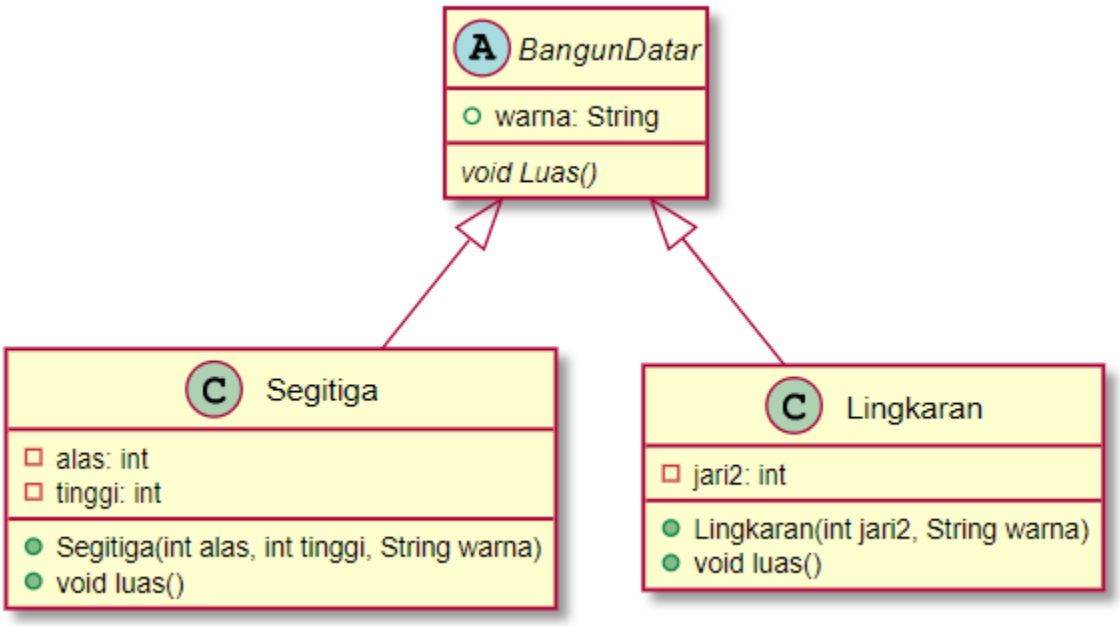
Class PostgreSQL

```
class PostgreSQL extends Database {
    String getDatabaseApp() {
        return "PostgreSQL DATABASE Server";
    }
    String getDatabaseName() {
        return "Rumah Sakit";
    }
    String getTableName() {
        return "Table Pasien";
    }
}
```

Implementasi App

```
public class App {
    public static void main(String[] args) throws Exception {
        MySQL mysql = new MySQL();
        System.out.println(mysql.getDatabaseApp());
        System.out.println(mysql.getDatabaseName());
        System.out.println(mysql.getTableName());

        PostgreSQL psql = new PostgreSQL();
        System.out.println(psql.getDatabaseApp());
        System.out.println(psql.getDatabaseName());
        System.out.println(psql.getTableName());
    }
}
```

Contoh lain: BangunDatar



Implementasinya adalah

Abstract Class BangunDatar

```
abstract class BangunDatar {
    public String warna;
    abstract void luas();
}
```

Class Segitiga

```
class Segitiga extends BangunDatar {
    private int alas;
    private int tinggi;

    public Segitiga(int alas, int tinggi, String warna) {
        this.alas = alas;
        this.tinggi = tinggi;
        this.warna = warna;
    }
```

```
    public void luas() {
        float l = (float) (this.alas * this.tinggi) / 2;
        System.out.println("Luas Segitiga: " + l);
    }
}
```

Class Lingkaran

```
class Lingkaran extends BangunDatar {
    private int jari2;

    public Lingkaran(int jari2, String warna) {
        this.jari2 = jari2;
        this.warna = warna;
    }

    public void luas() {
        float l = (float) (this.jari2 * this.jari2 * (22.0/7.0));
        System.out.println("Luas Lingkaran: " + l);
    }
}
```

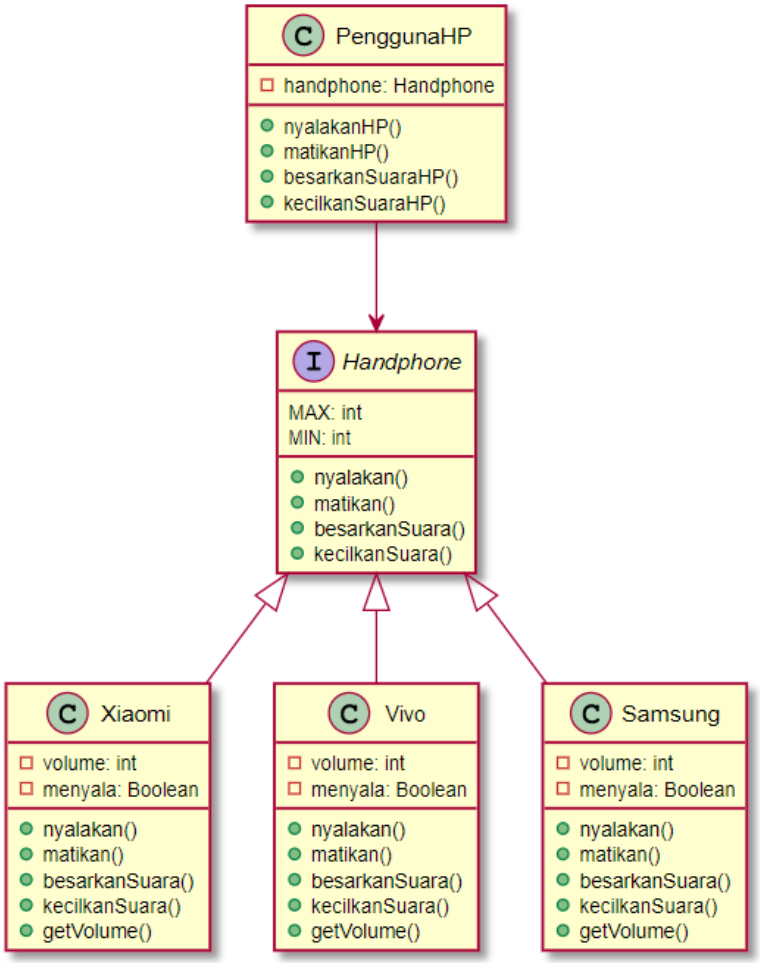
Implementasi App

```
public class App {
    public static void main(String[] args) throws Exception {
        Segitiga s1 = new Segitiga(10, 6, "Merah");
        Lingkaran l1 = new Lingkaran(10, "Biru");

        s1.luas();
        l1.luas();
    }
}
```

Interface

Interface secara harfiah adalah antar muka. Secara umum, interface berfungsi sebagai penghubung antara sesuatu yang 'abstrak' dengan sesuatu yang nyata. Seperti sebuah kelas, interface dapat memiliki metode dan variabel, tetapi method yang dideklarasikan pada interface hanya nama method tanpa body.

Sebagai contoh Interface adalah tombol on/off dan tombol up/down volume suara pada handphone. Cukup tekan tombol on/off untuk menyalakan atau mematikan handphone dan menekan tombol up untuk mengeraskan suara dan tombol down untuk mengecilkan suara. Sebagai pengguna, kita tidak mau tahu bagaimana caranya handphone bisa menyala, mati, volume membesar maupun mengecil.



Class Diagram adalah contoh bagaimana implementasi Interface Handphone. Pengguna HP pasti memiliki salah satu dari Xiaomi, Vivo, atau Samsung. Karena ketiga merek Handphone tersebut merupakan implementasi dari Handphone, maka ketiganya memiliki method yang sama namun bisa saja implementasi di dalam kodenya berbeda-beda.

Implementasinya adalah:

PenggunaHP

```
public class PenggunaHP {
    private Handphone phone;

    public PenggunaHP(Handphone phone) {
        this.phone = phone;
    }

    void nyalakanHP(){
        this.phone.nyalakan();
    }

    void matikanHP(){
        this.phone.matikan();
    }

    void besarkanSuaraHP(){
        this.phone.besarkanSuara();
    }

    void kecilkanSuaraHP(){
        this.phone.kecilkanSuara();
    }
}
```

Handphone

```
public interface Handphone {
    int MAX_VOLUME = 100;
    int MIN_VOLUME = 0;

    void nyalakan();
    void matikan();
    void besarkanSuara();
    void kecilkanSuara();
}
```

Xiaomi

```
public class Xiaomi implements Handphone {

    private int volume;
    private boolean menyala;

    public Xiaomi() {
        // set volume awal
        this.volume = 50;
    }

    @Override
    public void nyalakan() {
        menyala = true;
        System.out.println("Handphone menyala...");
        System.out.println("Selamat datang di XIAOMI");
        System.out.println("Android version 10");
    }

    @Override
    public void matikan() {
        menyala = false;
        System.out.println("Handphone dimatikan");
    }

    @Override
    public void besarkanSuara() {
```

```
        if (menyala) {
            if (this.volume == MAX_VOLUME) {
                System.out.println("Volume FULL!!");
                System.out.println("sudah " + this.getVolume() + "%");
            } else {
                this.volume += 10;
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }

    @Override
    public void kecilkanSuara() {
        if (menyala) {
            if (this.volume == MIN_VOLUME) {
                System.out.println("Volume = 0%");
            } else {
                this.volume -= 10;
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }

    public int getVolume() {
        return this.volume;
    }
}
```

Vivo

```
public class Vivo implements Handphone {
    private int volume;
    private boolean menyala;

    public Vivo() {
        // set volume awal
        this.volume = 50;
```

```
}

@Override
public void nyalakan() {
    menyala = true;
    System.out.println("Handphone menyala...");
    System.out.println("Selamat datang di Vivo");
    System.out.println("Android version 10");
}

@Override
public void matikan() {
    menyala = false;
    System.out.println("Handphone dimatikan");
}

@Override
public void besarkanSuara() {
    if (menyala) {
        if (this.volume == MAX_VOLUME) {
            System.out.println("Volume FULL!!");
            System.out.println("sudah " + this.getVolume() + "%");
        } else {
            this.volume += 10;
            System.out.println("Volume sekarang: " + this.getVolume());
        }
    } else {
        System.out.println("Nyalakan dulu donk HP-nya!!");
    }
}

@Override
public void kecilkanSuara() {
    if (menyala) {
        if (this.volume == MIN_VOLUME) {
            System.out.println("Volume = 0%");
        } else {
            this.volume -= 10;
            System.out.println("Volume sekarang: " + this.getVolume());
        }
    } else {
        System.out.println("Nyalakan dulu donk HP-nya!!");
```

```
        }
    }

    public int getVolume() {
        return this.volume;
    }
}
```

Samsung

```
public class Samsung implements Handphone {
    private int volume;
    private boolean menyala;

    public Samsung() {
        // set volume awal
        this.volume = 50;
    }

    @Override
    public void nyalakan() {
        menyala = true;
        System.out.println("Handphone menyala...");
        System.out.println("Selamat datang di SAMSUNG");
        System.out.println("Android version 11");
    }

    @Override
    public void matikan() {
        menyala = false;
        System.out.println("Handphone dimatikan");
    }

    @Override
    public void besarkanSuara() {
        if (menyala) {
            if (this.volume == MAX_VOLUME) {
                System.out.println("Volume FULL!!");
                System.out.println("sudah " + this.getVolume() + "%");
            } else {
                this.volume += 10;
```

```
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }


    @Override
    public void kecilkanSuara() {
        if (menyala) {
            if (this.volume == MIN_VOLUME) {
                System.out.println("Volume = 0%");
            } else {
                this.volume -= 10;
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }

    public int getVolume() {
        return this.volume;
    }
}
```

Implementasi App.java

```
package app;
import java.util.Scanner; // library untuk input data, bisa dari Console

public class App {
    public static void main(String[] args) throws Exception {
        Handphone redmiNote8 = new Xiaomi();

        PenggunaHP dian = new PenggunaHP(redmiNote8);

        dian.nyalakanHP();

        Scanner input = new Scanner(System.in);
```

```
        String aksi;

        while (true) {
            System.out.println("=== APLIKASI INTERFACE ===");
            System.out.println("[1] Nyalakan HP");
            System.out.println("[2] Matikan HP");
            System.out.println("[3] Perbesar Volume");
            System.out.println("[4] Kecilkan Volume");
            System.out.println("[0] Keluar");
            System.out.println("--------------------------");
            System.out.print("Pilih aksi> ");
            aksi = input.nextLine();

            if(aksi.equalsIgnoreCase("1")){
                dian.nyalakanHP();
            } else if (aksi.equalsIgnoreCase("2")){
                dian.matikanHP();
            } else if (aksi.equalsIgnoreCase("3")){
                dian.besarkanSuaraHP();
            } else if (aksi.equalsIgnoreCase("4")){
                dian.kecilkanSuaraHP();
            } else if (aksi.equalsIgnoreCase("0")){
                input.close();
                System.exit(0);
            } else {
                System.out.println("Kamu memilih aksi yang salah!");
            }
        }
    }
}
```

Perbedaan Abstract dan Interface

- **Jenis method**: Interface hanya dapat memiliki metode abstrak. Abstract dapat memiliki metode abstract dan non-abstract. Sejak Java 8, ia dapat memiliki metode default dan statis juga.
- **Final Variables**: Variabel yang dideklarasikan dalam interface Java secara default adalah final. Class abstract dapat berisi variabel non-final.
- **Jenis variabel**: Class abstract dapat memiliki variabel final, non-final, statis, dan non-statis. Interface hanya memiliki variabel statis dan final.
- **Implementasi**: Abstract class dapat menyediakan implementasi interface. Interface tidak dapat memberikan implementasi abstract class.
- **Inheritance vs Abstraction**: Interface Java dapat diimplementasikan menggunakan kata kunci "implement" dan class abstract dapat diperpanjang menggunakan kata kunci "extends".
- **Multi Implementasi**: Intreface hanya dapat memperluas interface Java lainnya saja, class abstract dapat memperluas class Java lainnya dan mengimplementasikan beberapa antarmuka Java.
- **Aksesibilitas Data Members**: Members dari Java interface secara default ada public. Class abstract java bisa memiliki members dengan aksesibilitas seperti private, protected, etc.
- **Class abstract** kita bisa buat properti atau variabel sedangkan di **interface** kita cuma bisa buat konstanta saja.
- **Class abstract** kita bisa implementasikan kode method seperti class biasa, sedangkan di **interface** harus menggunakan default
- **Class abstract** dapat memiliki member private dan protected sedangkan **interface** harus publik semua
- **Class abstract** diimpelentasikan dengan pewarisan (extends) sedangkan **interaface** menggunakan implements