Praktikum 5

Abstract dan Interface

Abstract dan Interface

Abstract Class

Abstraksi data adalah proses menyembunyikan beberapa detail dan hanya menampilkan informasi esensial kepada pengguna.

- Class Abstract dideklarasikan dengan kata kunci 'abstract'.
- Class Abstract tidak dapat diinisiasi menjadi object.
- Class Abstract dapat berisi abstract method.
- ***Abstract method*** hanya dapat digunakan dalam abstract class, dan tidak memiliki tubuh (body). Body (isi program) disediakan oleh subclass (yang mewarisi dari kelas tersebut).

Kendaraan.java

```
abstract class Kendaraan {
    private String merk;

    public Kendaraan(String merk) {
        this.merk = merk;
    }

    public String getMerk() {
        return merk;
    }

    // Metode abstrak
    public abstract void suaraMesin();
}
```

Implementasi Class turunan dari Kendaraan yaitu Mobil.java dan Motor.java

```
class Mobil extends Kendaraan {

    public Mobil(String merk) {
        super(merk);
    }

    // Implementasi dari metode abstrak suaraMesin()
    @Override
    public void suaraMesin() {
        System.out.println("Mobil dengan merk " + getMerk() + "
memiliki suara mesin 'vroom vroom'");
    }
}

class SepedaMotor extends Kendaraan {

    public SepedaMotor(String merk) {
        super(merk);
    }

    // Implementasi dari metode abstrak suaraMesin()
    @Override
    public void suaraMesin() {
        System.out.println("Sepeda motor dengan merk " + getMerk() + "
memiliki suara mesin 'broom broom'");
    }
}
```

Main method (Main.java)

```
public class Main {
    public static void main(String[] args) {
        Mobil mobilToyota = new Mobil("Toyota");
        mobilToyota.suaraMesin();

        SepedaMotor motorHonda = new SepedaMotor("Honda");
        motorHonda.suaraMesin();
    }
}
```

Interface

Sebuah interface dapat digunakan sebagai kontrak yang menjamin bahwa kelas yang mengimplementasikannya akan memiliki metode tertentu.

Berikut contoh terdapat class abstract Hewan lalu terdapat interface Pemakan. Lalu kita buat **class Kucing** yang merupakan turunan **class abstract Hewan** dan implementasikan **Pemakan**.

```
abstract class Hewan {
    private String nama;

    public Hewan(String nama) {
        this.nama = nama;
    }

    public String getNama() {
        return nama;
    }

    // metode abstrak yang akan diimplementasikan oleh subclass
    public abstract void bergerak();
}
```

```
interface Pemakan {
    void makan();
}
```

```
class Kucing extends Hewan implements Pemakan {

    public Kucing(String nama) {
        super(nama);
    }

    @Override
    public void bergerak() {
        System.out.println(getNama() + " berjalan dengan empat
kaki.");
    }
```

```
    @Override
    public void makan() {
        System.out.println(getNama() + " memakan ikan.");
    }
}
```

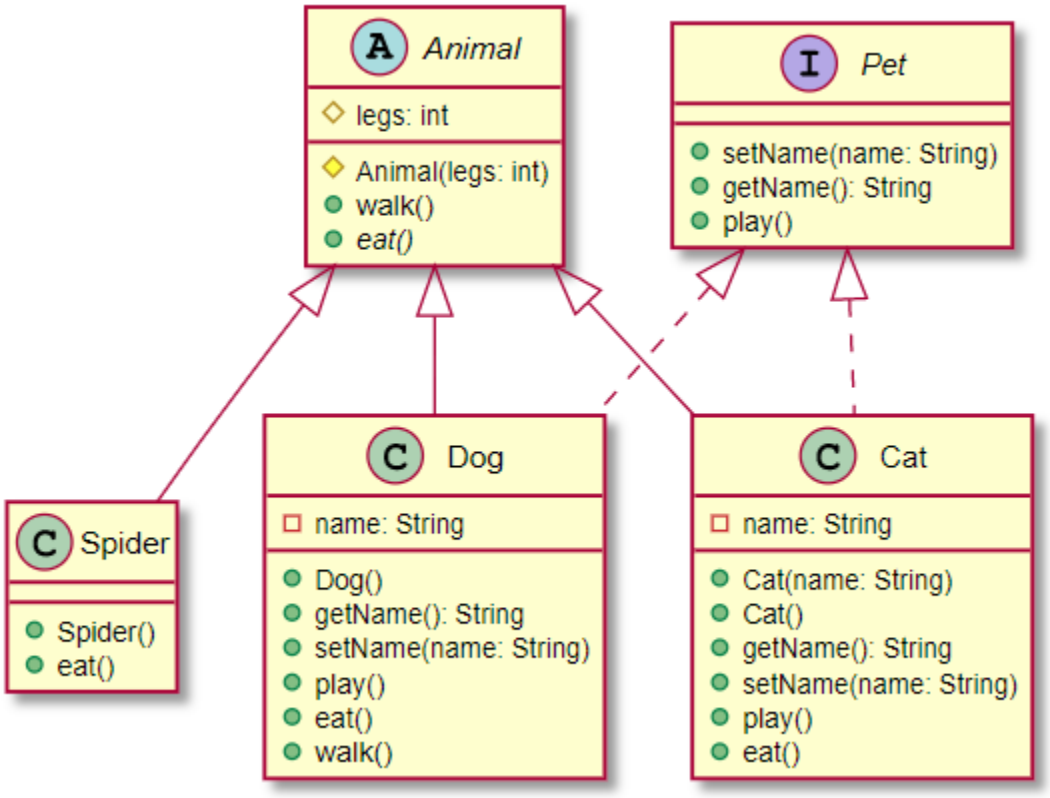
Driver Class / Main Method

```
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Whiskers");
        kucing.bergerak();
        kucing.makan();
    }
}
```

Output

```
Whiskers berjalan dengan empat kaki.
Whiskers memakan ikan.
```

Buatlah source code dari Diagram UML berikut



Untuk Abstract Class Animal

```
+walk()  // menampilkan string "Sedang berjalan"
```

Untuk Interface Pet

```
+setName(name: String)
+getName(): String
+play()
```

Isi dari method berikut kosong ya! Karena berupa interface.

Contoh untuk +play()

```
public void play();
```

Class Spider

```
+Spider() // constractor yang memanggil constructor parent Animal(legs int)
+eat() // menampilkan string "Sedang makan serangga"
```

Class Dog

```
+Dog() // constructor yang memanggil constructor parent Animal(legs int)
+getName(): Strring // mendapatkan nama anjing
+setName(name: String) // memberi nama anjinngnya
+play() // menampilkan string "Bermain tangkap stik kayu"
+eat() // menampilkan string "makan daging"
+walk() // @override method parent dan menampilkan string "jalan bersama
tuannya"
```

Class Cat

```
+Cat(name: String)
/* constructor yang memanggil constructor parent Animal(legs int)
Kemudian set nama kucing */
+Cat() // constructor yang memanggil constructor parent Animal(legs int)
+getName(): Strring // mendapatkan nama kucing
+setName(name: String) // memberi nama kucingnya
+play() // menampilkan string "Bermain bola kecil"
+eat() // menampilkan string "makan ikan"
```

Implementasi App.java

1. Buat objek Spider, nama bebas
2. Objek Spider → eat()
3. Buat objek Dog, nama bebas
4. Kasih nama anjingnya bebas
5. Tampilkan objek namanya
6. Objek dog play()
7. Objek dog eat()
8. Objek dog walk()
9. Buat object Cat, nama bebas dan sekaligus pakai constructor yang ada set namanya
10. Tampilkan objek kucing namanya
11. Object Cat play()
12. Object Cat eat()
13. Object Cat walk()