BAB IV
ENCAPSULATION

A. Konsep Dasar Encapsulation

Encapsulation merupakan salah satu konsep dasar dalam pemrograman berorientasi objek (OOP) yang mengacu pada pengemasan data dan perilaku terkait dalam sebuah unit tunggal, yaitu sebuah kelas. Dalam konsep ini, akses langsung ke data yang terdapat dalam sebuah objek dibatasi dan hanya memperbolehkan akses melalui method (metode) yang telah ditentukan.

Dalam Java, encapsulation dapat dicapai dengan mengubah aksesibilitas suatu variabel atau metode pada kelas. Atribut atau variabel yang ingin disembunyikan (private) dideklarasikan dengan modifier access private dan hanya dapat diakses di dalam kelas tersebut. Sedangkan method yang mengakses dan memanipulasi variabel tersebut dideklarasikan sebagai public.

Dengan menerapkan encapsulation, kita dapat meningkatkan keamanan program dan menghindari akses yang tidak sah atau salah pada data kelas. Selain itu, encapsulation juga memungkinkan kita untuk mengubah implementasi kelas tanpa mempengaruhi kelas lain yang menggunakannya.

B. Access Modifier

Dalam Java dikenal 4 macam modifier sebagai berikut

: 1. Publik

Public adalah access modifier yang paling umum digunakan dan memungkinkan variabel, method, atau kelas (harus dalam file terpisah) dapat diakses dari mana saja, baik dari dalam maupun luar kelas. Atribut atau method dengan access modifier public bisa diakses dan dimanipulasi dari kelas lain yang menggunakan kelas tersebut. Penggunaan public modifier biasanya untuk method-method yang bekerja

sebagai transportasi data seperti Setter Method dan Getter.

```
public class Main {
    public static void main(String[] args) {
        Car bbmw = new Car();
        bbmw.start();
        // System.out.println(bbmw.brand);
        System.out.println(bbmw.getBrand());
    }

}
```

2. Protected

Protected adalah access modifier yang memungkinkan variabel, method, atau kelas dapat diakses dari dalam kelas itu sendiri, subclass (kelas turunan) serta pada package yang sama. Atribut atau method dengan access modifier protected bisa diakses dan dimanipulasi dari subclass yang meng-extend kelas tersebut

```
// Kelas Induk (superclass)
class Kendaraan {
    // Variabel instance protected
    protected String merek;

    // Konstruktor
    public Kendaraan(String merek) {
        this.merek = merek;
    }

    // Method protected
    protected void bergerak() {
        System.out.println("Kendaraan bergerak.");
    }
}

// Kelas Turunan (subclass)
class Mobil extends Kendaraan {
    // Konstruktor
    public Mobil(String merek) {
        // Memanggil konstruktor superclass dengan menggunakan kata
kunci super
        super(merek);
    }
```

```
    // Method public yang memanggil method protected dari
superclass
    public void jalankan() {
        // Memanggil method protected dari superclass
        bergerak();
        System.out.println("Mobil merek " + merek + " mulai
berjalan.");
    }
}

public class Main {
    public static void main(String[] args) {
        // Membuat objek Mobil
        Mobil mobil1 = new Mobil("Toyota");

        // Memanggil method public dari objek Mobil
        mobil1.jalankan();
    }
}
```

3. Default Modifier

   Default atau package-private adalah access modifier yang tidak menggunakan kata kunci access modifier (tidak menggunakan public, private, atau protected) dan hanya dapat diakses dari kelas yang berada dalam package yang sama. Atribut atau method dengan access modifier default tidak bisa diakses dari package yang berbeda.

```
class Kendaraan {
    String merek;
    Kendaraan(String merek) {
        this.merek = merek;
    }
    void bergerak() {
        System.out.println("Kendaraan bergerak.");
    }
    // jika kita ingin method bergerak() di akses dari luar
    package // maka kita harus mengimport class kendaraan
}
```

4. Private

   Private adalah access modifier yang membatasi aksesibilitas variabel, method, atau kelas hanya pada kelas

itu sendiri. Dengan demikian, variabel dan method yang dideklarasikan sebagai private tidak dapat diakses atau dimanipulasi dari kelas lain.

Contoh :

```
Class ContohPrivate {
    private int angkaPrivate; // Variabel private
    private void metodePrivate() {
        System.out.println("Ini adalah metode private yang hanya dapat diakses
dari dalam kelas ContohPrivate");
    }
    public void panggilMetodePrivate() {
        metodePrivate(); // Memanggil metode private dari dalam kelas
ContohPrivate
    }
}
public class Main{
    public static void main(String[] args) {
        ContohPrivate contoh = new ContohPrivate(10); // Membuat objek dari
kelas ContohPrivate
        // Mencoba mengakses method private (tidak bisa dilakukan dari luar
kelas ContohPrivate)
        // contoh.metodePrivate(); // Ini akan menyebabkan error kompilasi
        // Memanggil method public yang memanggil method private
        contoh.panggilMetodePrivate();
    }
}
```

Selain penerapan Modifire pada atribut dan method akses modifier sendiri juga di terapkan pada class dengan aksesibilitasnya juga. Adapaun penjelasan yang lebih dalam soal modifire pada class adalah sebagai berikut :

1. *Public Class*

   Access Modifier public digunakan untuk membuat class bisa diakses dari mana saja, baik dari package yang sama maupun dari package yang berbeda. Class yang dideklarasikan dengan public bisa dipakai oleh semua class yang ada dalam project, bahkan bisa dipakai oleh project lain.
2. *Default Class*

   Access Modifier default digunakan jika tidak ada access

modifier yang ditulis pada deklarasi class. Class yang dideklarasikan dengan access modifier default hanya bisa diakses oleh class-class yang ada dalam package yang sama dengan class tersebut.

3. Final Class
   Modifier final digunakan untuk mendeklarasikan class yang tidak bisa diwariskan ke class lain. Class yang dideklarasikan dengan access modifier final tidak bisa di-extend oleh class lain.

C. Encapsulation dan Data Hiding

Encapsulation mengacu pada pengelompokan data dan metode yang terkait ke dalam satu unit yang disebut class. Data dan metode yang ada dalam class tersebut bisa diatur tingkat aksesibilitasnya menggunakan access modifier. Dengan cara ini, data dan metode yang ada dalam class tidak dapat diakses atau dimanipulasi dari luar class tersebut, sehingga mencegah penggunaan yang tidak sah dan memungkinkan program untuk lebih terorganisir.

Data Hiding mengacu pada praktik menyembunyikan detail implementasi suatu objek dari dunia luar, sehingga hanya operasi dasar yang bisa dilakukan pada objek tersebut dan tidak dapat dimanipulasi secara langsung. Konsep data hiding bertujuan untuk membatasi aksesibilitas data pada objek, sehingga hanya metode yang ditentukan yang dapat mengakses dan memodifikasi data tersebut. Penerapan Encapsulation dan Data Hiding di Java :

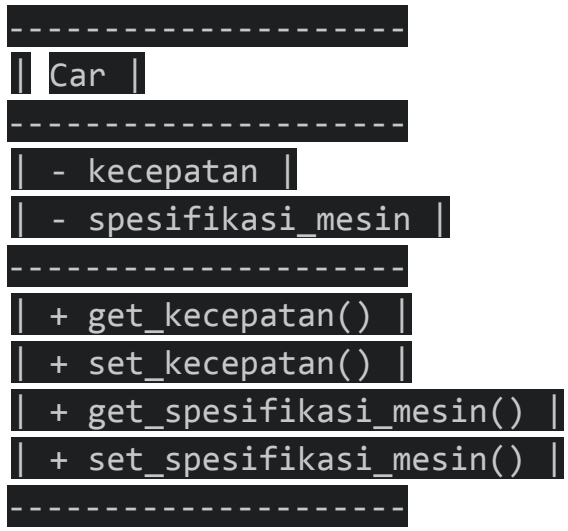
1. Menggunakan Access Modifier
2. Menggunakan Setter dan Getter
3. Menggunakan Constructor

Sudah disebutkan sebelumnya bahwa variabel privat hanya dapat diakses dalam kelas yang sama (kelas di luar tidak memiliki akses kepadanya). Namun, memungkinkan untuk mengaksesnya jika kita menyediakan metode get dan set publik. Metode get mengembalikan nilai variabel, dan metode set mengatur nilainya. Syntax untuk keduanya

adalah dimulai dengan get atau set, diikuti oleh nama variabel, dengan huruf pertama huruf besar.

Cara pemberian nama yang umum pada method setter atau getter untuk atribut tertentu yaitu dengan memberikan kata “set” atau kata “get” disertai dengan “Nama Attribute” dan dengan return type void serta sebuah parameter dengan tipe data yang sama dengan tipe data attribute tersebut.

Dengan memberikan method setter dan getter maka akses pada data akan menjadi lebih aman karena kita bisa menambahkan sebuah argumen kunci yang akan diminta jika kita mencoba untuk mengaakses data tersebut. Sebagi contoh class car dibawah




```
public class Car {
    private String kunci;
    private int kecepatan;
    private String spesifikasiMesin;

    // Constructor
    public Car(String kunci) {
        this.kunci = kunci;
    }

    // Getter untuk atribut kecepatan
    public int getKecepatan(String kunci) {
        if (this.kunci.equals(kunci)) {
            return kecepatan;
        } else {
```

```
            System.out.println("Kunci salah");
            return -1; // Nilai default jika kunci salah
        }
    }

    // Setter untuk atribut kecepatan
    public void setKecepatan(String kunci, int kecepatan) {
        if (this.kunci.equals(kunci)) {
            this.kecepatan = kecepatan;
        } else {
            System.out.println("Kunci salah");
        }
    }

    // Getter untuk atribut spesifikasiMesin
    public String getSpesifikasiMesin(String kunci) {
        if (this.kunci.equals(kunci)) {
            return spesifikasiMesin;
        } else {
            System.out.println("Kunci salah");
            return null; // Nilai default jika kunci salah
        }
    }

    // Setter untuk atribut spesifikasiMesin
    public void setSpesifikasiMesin(String kunci, String spesifikasiMesin) {
        if (this.kunci.equals(kunci)) {
            this.spesifikasiMesin = spesifikasiMesin;
        } else {
            System.out.println("Kunci salah");
        }
    }
}
```

E. Immutable Class

Immutable class adalah class di Java yang tidak bisa diubah setelah dibuat. Artinya, setiap kali perlu mengubah nilai pada class tersebut, kita harus membuat instance baru dari class tersebut.

Contoh immutable class di Java adalah jika terdapat class String. Ketika kita membuat sebuah String, kita tidak bisa mengubah karakter-karakter yang sudah ada di dalamnya, melainkan harus membuat instance baru jika ingin mengubah isi String tersebut. Adapun

contoh lainya adalah sbb:

```
public final class ImmutableClass {
    private final String name; // Deklarasi atribut name sebagai final
    private final int age; // Deklarasi atribut age sebagai final

    // Constructor untuk inisialisasi nilai name dan age
    public ImmutableClass(String name, int age) {
        this.name = name; // Inisialisasi atribut name
        this.age = age; // Inisialisasi atribut age
    }

    // Getter untuk mendapatkan nilai name
    public String getName() {
        return name; // Mengembalikan nilai atribut name
    }

    // Getter untuk mendapatkan nilai age
    public int getAge() {
        return age; // Mengembalikan nilai atribut age
    }
}
```

F. Static attribute dan static method

Static attribute atau method sebenarnya tidak terkait secara langsung dengan konsep encapsulation dalam Java. Namun, penggunaan static attribute atau method dapat berdampak pada implementasi encapsulation dalam suatu program.

Static attribute/method adalah sesuatu yang terkait dengan kelas, bukan dengan instance dari kelas tersebut. Static attribute/method dapat diakses dan dimodifikasi tanpa membuat instance baru dari kelas tersebut. Hal ini memungkinkan data yang sama digunakan di berbagai instance kelas dan menyediakan cara untuk berbagi data di seluruh program.

```
class MathUtils {
    // PI digunakan sebagai konstanta untuk nilai π
    private static final double PI = 3.14159265358979323846;

    // Konstruktor private untuk mencegah instansiasi kelas ini
    private MathUtils() {}

    // Metode untuk menghitung luas lingkaran berdasarkan jari-jari
```

```
    public static double calcCircleArea(double r) {
        return PI * r * r;
    }

    // Metode untuk menghitung luas persegi panjang berdasarkan panjang dan
lebar
    public static double calcRectArea(double l, double w) {
        return l * w;
    }
}

public class Main {
    public static void main(String[] args) {
        // Menghitung luas lingkaran dengan jari-jari 5
        double circleArea = MathUtils.calcCircleArea(5.0);

        // Menghitung luas persegi panjang dengan panjang 4 dan lebar 6
        double rectArea = MathUtils.calcRectArea(4.0, 6.0);

        // Menampilkan hasil perhitungan luas lingkaran dan persegi panjang
        System.out.println("Circle area: " + circleArea);
        System.out.println("Rectangle area: " + rectArea);
    }
}
```