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. Public

   Public adalah access modifier yang paling umum digunakan dan memungkinkan attribute, method, atau kelas (harus dalam file terpisah) dapat diakses dari mana saja, baik dari dalam maupun luar kelas. Attribute 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 method Getter – Setter.

   Contoh:

```java
public class Buku {
    public String judul;

    public Buku(String judul) {
        this.judul = judul;
    }

    public static void main(String[] args) {
        Buku bukuBaru = new Buku("Laskar Pelangi");
        System.out.println("Judul buku: " + bukuBaru.judul);
    }
}
```

   Output:

```
Judul buku: Laskar Pelangi
```

2. Protected
   Protected adalah access modifier yang memungkinkan attribute dan method dapat diakses dari dalam kelas itu sendiri, subclass (kelas turunan), serta kelas lain dalam package yang sama. Dengan modifier ini, attribute atau method yang dideklarasikan sebagai protected dapat digunakan oleh subclass yang meng-extend kelas tersebut, tetapi tidak dapat diakses dari luar package jika tidak melalui subclass.

   Contoh:

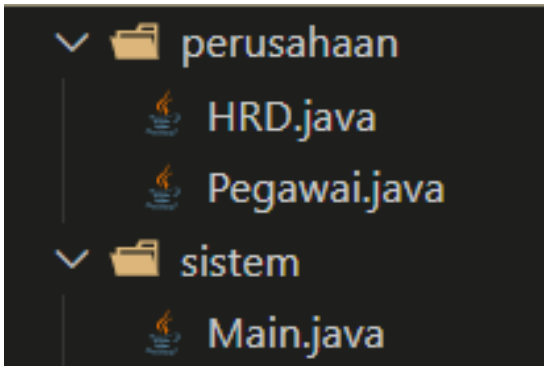
```
class Kendaraan {
    protected String merek; // Attribute protected

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

    // Method protected
    protected void info() {
        System.out.println("Kendaraan merek: " + merek);
    }
}

// Subclass yang mewarisi Kendaraan
class Mobil extends Kendaraan {
    private String tipe;

    // Constructor
    public Mobil(String merek, String tipe) {
        super(merek);
        this.tipe = tipe;
    }

    // Method untuk mengakses method protected dari superclass
    public void tampilkanInfo() {
        info(); // Memanggil method protected dari superclass
        System.out.println("Tipe mobil: " + tipe);
    }
}

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

        mobil1.tampilkanInfo(); // Bisa diakses karena subclass
        // mobil1.info(); // ERROR
    }
}
```

   Output:

```
Kendaraan merek: Toyota
Tipe mobil: SUV
```

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.

   Contoh:
   Perhatikan struktur folder berikut



   Pegawai.java

```java
package perusahaan;

// Tidak menggunakan 'public', jadi class ini merupakan default class
class Pegawai {
    String nama; // Default attribute
    int gaji;

    // Constructor default
    Pegawai(String nama, int gaji) {
        this.nama = nama;
        this.gaji = gaji;
    }

    // Method default
    void tampilkanData() {
        System.out.println("Nama Pegawai: " + nama);
        System.out.println("Gaji: Rp " + gaji);
    }
}
```

   HRD.java

```java
package perusahaan;

public class HRD {
    public static void main(String[] args) {
        // HRD bisa mengakses Pegawai karena dalam package yang sama
        Pegawai pegawai1 = new Pegawai("Budi", 5000000);
        pegawai1.tampilkanData();
    }
}
```

   Output:

```
Nama Pegawai: Budi
Gaji: Rp 5000000
```

Main.java dalam package sistem

```java
package sistem;

// import perusahaan.Pegawai; // ERROR

public class Main {
    public static void main(String[] args) {
        // Pegawai pegawai2 = new Pegawai("Siti", 6000000); // ERROR
        // pegawai2.tampilkanData(); // ERROR
    }
}
```

4. Private

Private adalah access modifier yang membatasi aksesibilitas attribute, method, atau class hanya pada class itu sendiri. Dengan demikian, attribute dan method yang dideklarasikan sebagai private tidak dapat diakses atau dimanipulasi dari class lain.

Contoh :

```java
class Akun {
    private String password; // Attribute private

    public Akun(String password) {
        this.password = password;
    }

    //Getter
    public String getPassword() {
        return password;
    }

    //Setter
    public void setPassword(String password) {
        this.password = password;
    }


}

public class Main {
    public static void main(String[] args) {
        Akun akun1 = new Akun("Admin123");

        System.out.println("Password: " + akun1.getPassword());
        // akun1.password; // ERROR
    }
}
```

Output:

```
Password: Admin123
```

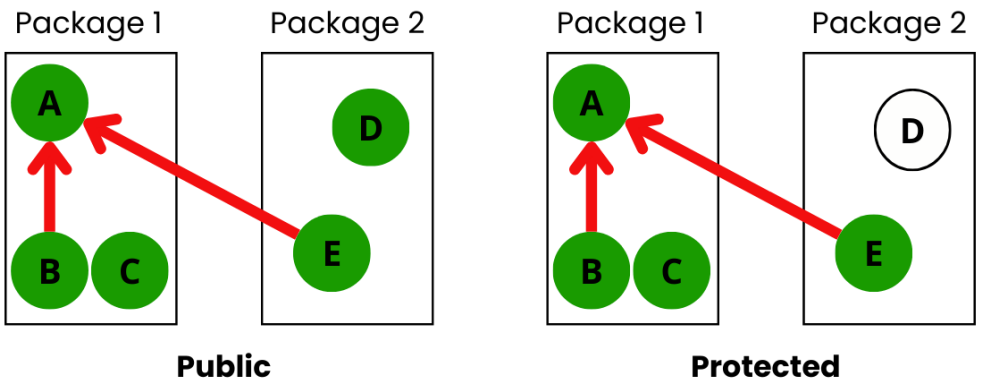
5. Access modifier pada class

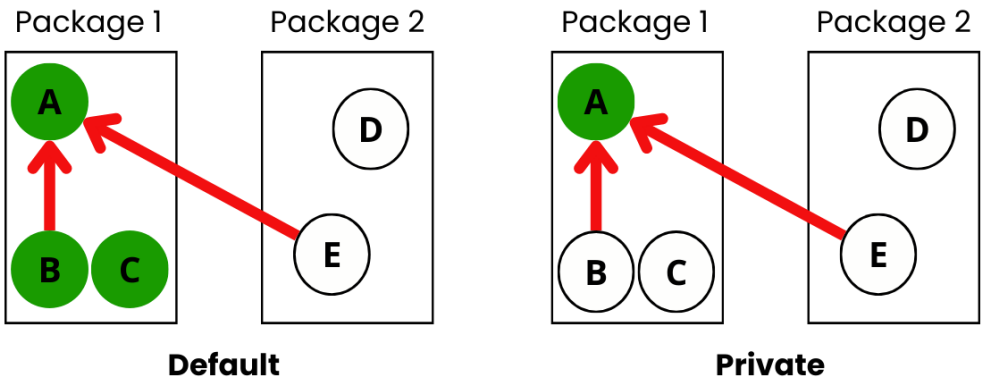
Selain penerapan Modifier pada atribut dan method access modifier sendiri juga di terapkan pada class dengan aksesibilitasnya juga. Adapun penjelasan yang lebih dalam soal modifier pada class adalah sebagai berikut :

a. *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.
b. *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.
c. *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.

Berikut kesimpulan dari access modifier:

Modifier	Class	Package	Subclass	Semua
Public	✓	✓	✓	✓
Protected	✓	✓	✓	
Default	✓	✓		
Private	✓			





C. Encapsulation dan Data Hiding

Encapsulation mengacu pada pengelompokan data dan method yang terkait ke dalam satu unit yang disebut class. Data dan method yang ada dalam class tersebut bisa diatur tingkat aksesibilitasnya menggunakan access modifier. Dengan cara ini, data dan method 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 method yang ditentukan yang dapat mengakses dan memodifikasi data tersebut.
Berdasarkan konsep ini, terdapat dua komponen utama yang harus dipahami:

1. Interface adalah bagian yang terlihat dari luar.
   - Interface menjelaskan karakteristik objek yang dapat diakses oleh dunia luar.
   - Interface data adalah informasi yang sengaja dibuat dapat dilihat atau diakses oleh class atau metode lain.
2. Implementation adalah bagian yang tersembunyi.
   - Implementation mencakup cara kerja internal yang tidak perlu diketahui pengguna.
   - Implementation data hanya dapat diubah dari dalam class itu sendiri, melindunginya dari modifikasi langsung oleh kode eksternal.

Penerapan Encapsulation dan Data Hiding di Java :

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

Sudah disebutkan sebelumnya bahwa attribute private hanya dapat diakses dalam class yang sama (class di luar tidak memiliki akses kepadanya). Namun, memungkinkan untuk mengaksesnya jika kita menyediakan metode get dan set public. Method get mengembalikan nilai attribute, dan method set mengatur nilainya.

Contoh:

```
public class Ninja {
    // Atribut hidden
    private String nama = "Ninja";
```

```
    private int chakra = 0;

    public Ninja(String nama, int chakra) {
        this.nama = nama;
        this.chakra = chakra;
    }

    public String getNama() {
        return nama;
    }

    public int getChakra() {
        return chakra;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }

    public void setChakra(int chakra) {
        if (chakra >= 0) {
            this.chakra = chakra;
        }
    }

    public static void main(String[] args) {
        // Mengubah nilai menggunakan constructor
        Ninja naruto = new Ninja("Naruto Uzumaki", 100);

        System.out.println("Ninja: " + naruto.getNama() +
        ", Chakra: " + naruto.getChakra());

        // Mengubah nilai menggunakan setter
        naruto.setChakra(80);
        System.out.println("Setelah latihan, Chakra: " + naruto.getChakra());
    }
}
```

Output:

```
Ninja: Naruto Uzumaki, Chakra: 100
Setelah latihan, Chakra: 80
```

D. 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 lainnya adalah sebagai berikut:

```
final class ImmutableClass {
    private final String name; // Attribute final
    private final int age; // Attribute final

    public ImmutableClass(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

public class Main {
    public static void main(String[] args) {
        ImmutableClass person = new ImmutableClass("Rudy", 19);

        System.out.println("Nama: " + person.getName());
        System.out.println("Usia: " + person.getAge());

        // person.name = "Fajrin"; // ERROR: Tidak bisa diubah karena final
        // person.setAge(20); // ERROR: Tidak ada setter
    }
}
```

Output:

```
Nama: Rudy
Usia: 19
```

E. Static attribute dan static method

Static attribute atau method sebenarnya tidak langsung terkait dengan konsep encapsulation dalam Java. Namun, penggunaannya dapat mempengaruhi cara data diakses dan dikelola dalam suatu program. Static attribute/method adalah sesuatu yang terkait dengan class, bukan dengan instance dari kelas tersebut. Artinya, nilai dari static attribute akan dibagikan ke semua instance, bukan hanya milik satu objek saja. Static method juga dapat dipanggil tanpa harus membuat instance dari class tersebut.

Jika static attribute dideklarasikan sebagai public, maka data dapat diakses dari mana saja, yang bisa mengurangi prinsip encapsulation. Namun, jika static attribute bersifat private dan hanya bisa diakses melalui method getter/setter, maka encapsulation tetap terjaga

Contoh:

```
class Counter {
    private static int count = 0; // Static attribute dengan encapsulation

    // Method static untuk menambah counter
    public static void tambah() {
        count++;
    }

    public static int getCount() {
        return count;
    }
}

public class Main {
    public static void main(String[] args) {
        Counter.tambah();
        Counter.tambah();
        System.out.println("Jumlah objek: " + Counter.getCount());
    }
}
```