Bab 4

Encapsulation dan Information Hiding

4.1 Konsep Encapsulation dan Access Modifier

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 1.3: Mendemonstrasikan konsep encapsulation dengan access modifier

Konsep Penting

Encapsulation adalah prinsip OOP yang menggabungkan data (attributes) dan perilaku (methods) dalam satu class, sekaligus menyembunyikan detail internal dari akses langsung luar [1]. Tujuannya adalah menjaga *class invariant* agar data selalu dalam kondisi valid.

Dengan encapsulation, kita dapat:

- Melindungi data dari perubahan yang tidak sah
- Mengontrol akses melalui method publik
- Memudahkan perawatan dan perubahan implementasi internal

4.1.1 Access Modifier di Java

Java menyediakan access modifier untuk mengatur visibilitas anggota class:

4.1.2 Contoh Encapsulation Sederhana

Modifier	**Akses**	**Keterangan**
public	Semua class	Dapat diakses dari mana saja
protected	Package + subclass	Dapat diakses dari package yang sama dan subclass
(default)	Package	Tanpa keyword, hanya dalam package yang sama
private	Class sendiri	Hanya dapat diakses di dalam class

Tabel 4.1: Access Modifier di Java

```
1  public class RekeningBank {
2      private String nomorRekening;
3      private String namaPemilik;
4      private double saldo;
5
6      public RekeningBank(String nomorRekening, String
           namaPemilik) {
7          this.nomorRekening = nomorRekening;
8          this.namaPemilik = namaPemilik;
9          this.saldo = 0.0;
10     }
11
12     public void setor(double jumlah) {
13         if (jumlah <= 0) {
14             throw new IllegalArgumentException("Jumlah harus
                   positif");
15         }
16         saldo += jumlah;
17     }
18
19     public void tarik(double jumlah) {
20         if (jumlah <= 0 || jumlah > saldo) {
21             throw new IllegalArgumentException("Saldo tidak
                   mencukupi");
22         }
23         saldo -= jumlah;
24     }
25
26     public double getSaldo() {
27         return saldo;
28     }
29 }
```

Kode Program 4.1: Encapsulation pada class RekeningBank

Contoh di atas menunjukkan bahwa `saldo` hanya dapat diubah melalui method `setor()` dan `tarik()`, sehingga aturan bisnis dapat dijaga dengan konsisten.

4.2 Getter/Setter, Validasi, dan Immutability

4.2.1 Getter dan Setter

Getter dan setter adalah method publik yang mengontrol akses terhadap attribute private. Pola ini memungkinkan validasi dan konsistensi data.

```
public class Produk {
    private String kode;
    private String nama;
    private double harga;

    public Produk(String kode, String nama, double harga) {
        this.kode = kode;
        this.nama = nama;
        setHarga(harga);
    }

    public String getKode() {
        return kode;
    }

    public String getNama() {
        return nama;
    }

    public double getHarga() {
        return harga;
    }

    public void setHarga(double harga) {
        if (harga <= 0) {
            throw new IllegalArgumentException("Harga harus
                positif");
        }
        this.harga = harga;
    }
}
```

Kode Program 4.2: Contoh Getter dan Setter dengan Validasi

4.2.2 Immutability

Object *immutable* tidak dapat diubah setelah dibuat. Pendekatan ini meminimalkan bug dan meningkatkan keamanan data.

Catatan

Untuk membuat class immutable:

- Gunakan `private final` untuk attributes

- Tidak menyediakan setter
- Inisialisasi di constructor
- Jika menyimpan objek mutable, lakukan defensive copy

Aktivitas Pembelajaran

1. **Refactoring Encapsulation**: Diberikan class dengan attributes public, ubah menjadi private dan buat getter/setter yang tepat.
2. **Validasi Data**: Implementasikan validasi untuk class `Mahasiswa` (nilai 0-100, IPK 0.0-4.0).
3. **Design Review**: Diskusikan kapan sebaiknya sebuah attribute tidak diberi setter.
4. **Studi Kasus**: Buat class `BookingHotel` dengan aturan: tanggal check-out harus setelah check-in.

Latihan dan Refleksi

1. Jelaskan manfaat utama encapsulation dalam pengembangan software.
2. Apa perbedaan antara attribute `public` dan `private`? Berikan contoh.
3. Buat class `KartuMember` dengan attribute nomorMember, nama, poin. Terapkan encapsulation dan validasi poin tidak boleh negatif.
4. Kapan sebaiknya sebuah class dibuat immutable? Jelaskan dengan contoh.
5. **Refleksi**: Bagian apa dari encapsulation yang paling sering Anda lupakan saat menulis kode?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 1.3

A. Pilihan Ganda

1. Tujuan utama encapsulation adalah:
   (a) Mempercepat eksekusi program

(b) Menyembunyikan detail internal dan mengontrol akses data

(c) Mengurangi jumlah class

(d) Menghapus kebutuhan constructor

2. Keyword yang paling tepat untuk attribute yang hanya boleh diakses dalam class:

(a) public

(b) protected

(c) private

(d) default

B. Tugas Praktik

- Buat class **`AkunPengguna`** dengan validasi email dan password minimal 8 karakter.
- Sertakan getter yang diperlukan dan batasi setter pada data yang tidak boleh diubah.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat menjelaskan konsep encapsulation dan information hiding

☐ Saya dapat menggunakan access modifier dengan tepat

☐ Saya mampu membuat getter/setter dengan validasi

☐ Saya mengetahui kapan menggunakan object immutable

☐ Saya dapat menjaga class invariant dengan encapsulation

Rangkuman

Encapsulation menjaga data tetap aman dan konsisten melalui access modifier dan method pengontrol [1]. Getter/setter memungkinkan validasi, sedangkan immutability membantu mencegah perubahan yang tidak diinginkan.

Kata Kunci: ***Encapsulation***, ***Access Modifier***, ***Getter/Setter***, ***Immutable***, ***Validation***

Daftar Pustaka

[1] Joshua Bloch. *Effective Java*. Addison-Wesley Professional, 3rd edition, 2018.