Laporan Modul 5: Enkapsulasi

Mata Kuliah: Praktikum Pemrograman Berorientasi Objek
Nama: Riski Al Fatah
NIM: 2024573010036
Kelas: TI.2E

Abstrak

Laporan praktikum ini (Modul 5) membahas pilar fundamental Enkapsulasi dalam Pemrograman Berorientasi Objek (OOP). Tujuan utamanya adalah untuk memahami dan mengimplementasikan mekanisme perlindungan data (data protection) dan penyembunyian implementasi (data hiding) melalui access modifier dan metode getter/setter. Metodologi yang digunakan adalah melalui tiga studi kasus praktikum: (1) Uji coba Access Modifier (private, public, protected, default) pada kelas Person untuk membuktikan bagaimana akses langsung ke anggota private dicegah oleh compiler. (2) Implementasi Getter dan Setter pada kelas Mahasiswa untuk menyediakan akses baca-tulis yang terkontrol, sekaligus menerapkan validasi data pada setter (misalnya, validasi IPK dan Nama). (3) Penerapan konsep properti Read-Only, Write-Only, dan Computed, serta validasi keamanan berbasis PIN pada kelas Product. Hasil dari praktikum ini secara kolektif mendemonstrasikan bahwa enkapsulasi bukan hanya menyembunyikan data, tetapi secara aktif melindunginya, memastikan integritas data, dan meningkatkan modularitas serta keamanan kode.

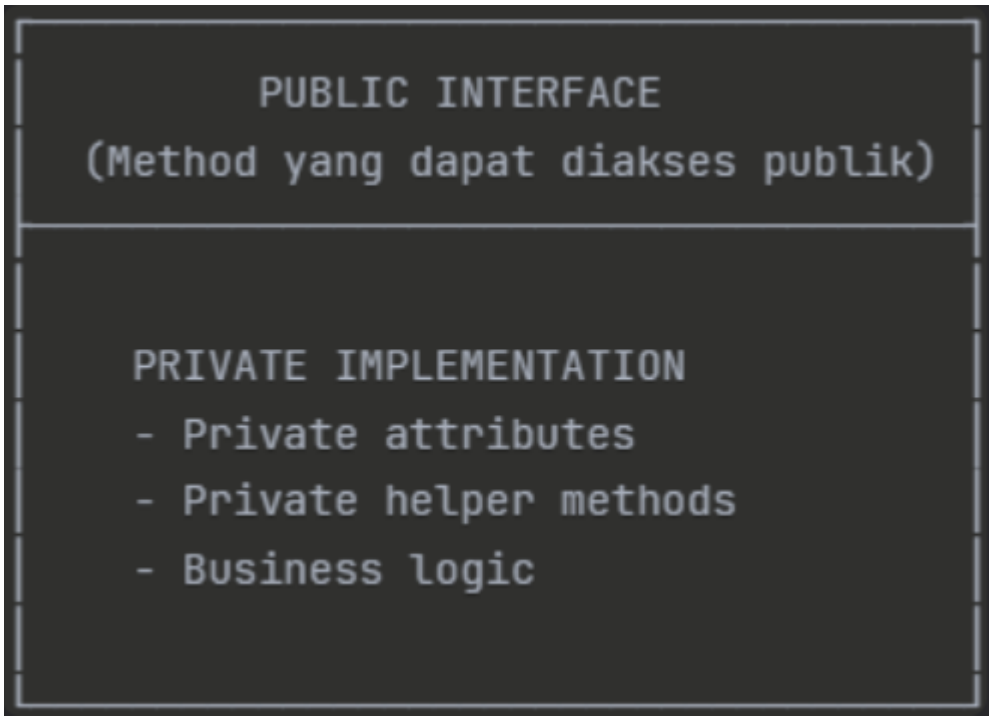
Pendahuluan

Encapsulation (Enkapsulasi) adalah salah satu prinsip fundamental dalam Object-Oriented Programming (OOP) yang membungkus data (attributes) dan method yang bekerja pada data tersebut dalam satu unit (class), serta menyembunyikan detail implementasi internal dari dunia luar.

Tujuan Encapsulation

1. Data Protection - Melindungi data dari akses dan modifikasi yang tidak sah.
2. Data Validation - Memastikan data yang masuk valid sebelum disimpan.
3. Flexibility - Mudah mengubah implementasi internal tanpa mempengaruhi kode luar.
4. Maintainability - Kode lebih mudah dipelihara dan di-debug.
5. Modularity - Membuat kode lebih modular dan terorganisir

Prinsip Utama



Access Modifier:

Access modifier menentukan tingkat akses terhadap class, attributes, dan methods. Java memiliki 4 jenis access modifier:

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

Praktikum 1: Memahami Access Modifier

Contoh Program yang mengikuti modul:

1. Class Person

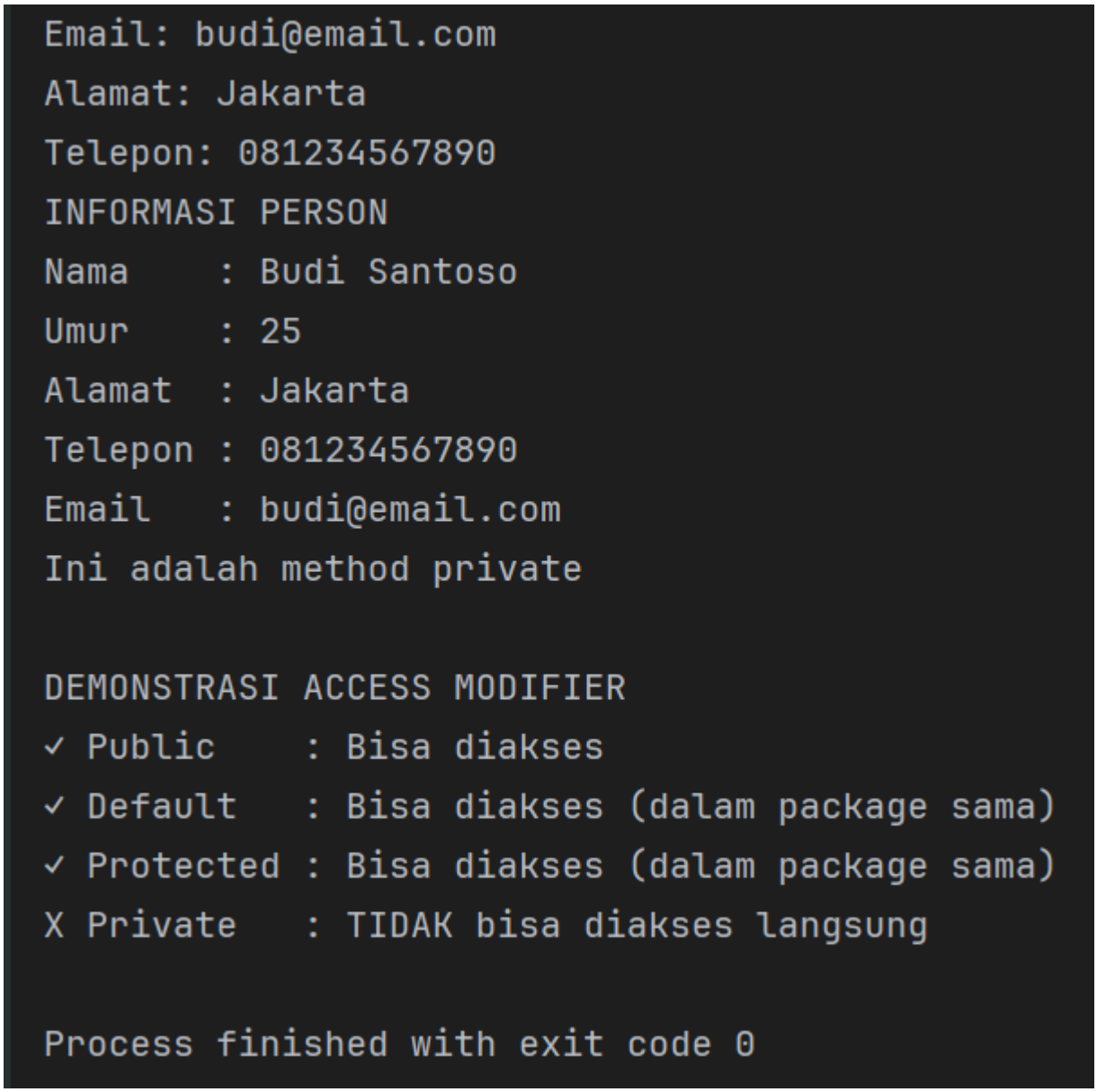
```java
package modul_5.praktikum_1;

public class Person {  2 usages  new *

    // Private - hanya bisa diakses dalam class ini
    private String nama;  2 usages
    private int umur;  2 usages

    // Default (package-private) - bisa diakses dalam package yang sama
    String alamat;  3 usages

    // Protected - bisa diakses dalam package dan subclass
    protected String telepon;  3 usages

    // Public - bisa diakses dari mana saja
    public String email;  3 usages

    // Constructor
    public Person(String nama, int umur) {  2 usages  new *
        this.nama = nama;
        this.umur = umur;
    }

    // Public method untuk menampilkan info
    public void tampilkanInfo() {  1 usage  new *
        System.out.println("INFORMASI PERSON");
        System.out.println("Nama    : " + nama);      // OK - dalam class yang sama
        System.out.println("Umur    : " + umur);      // OK - dalam class yang sama
        System.out.println("Alamat  : " + alamat);    // OK - dalam class yang sama
```

```
        System.out.println("Telepon : " + telepon);   // OK - dalam class yang sama
        System.out.println("Email   : " + email);     // OK - dalam class yang sama
    }

    // Private method - hanya bisa dipanggil dalam class ini
    private void metodePribadi() {  1 usage  new *
        System.out.println("Ini adalah method private");
    }

    // Protected method
    protected void metodeProtected() {  no usages  new *
        System.out.println("Ini adalah method protected");
    }

    // Method untuk mengakses private method
    public void panggilMetodePribadi() {  1 usage  new *
        metodePribadi(); // OK - dalam class yang sama
    }

}
```

2. Class AccessModifierTest

```java
package modul_5.praktikum_1;

public class AccessModifierTest {  new *
    public static void main(String[] args) {  new *
        Person person = new Person( nama: "Budi Santoso",  umur: 25);

        // Test akses public
        person.email = "budi@email.com"; // OK - public
        System.out.println("Email: " + person.email);

        // Test akses default (dalam package yang sama)
        person.alamat = "Jakarta"; // OK - dalam package yang sama
        System.out.println("Alamat: " + person.alamat);

        // Test akses protected (dalam package yang sama)
        person.telepon = "081234567890"; // OK - dalam package yang sama
        System.out.println("Telepon: " + person.telepon);

        // Test akses private - AKAN ERROR jika uncomment
        // person.nama = "Andi";      // ERROR - private
        // person.umur = 30;          // ERROR - private
        // person.metodePribadi();    // ERROR - private

        // Mengakses data private melalui public method
        person.tampilkanInfo();

        // Mengakses private method melalui public method
        person.panggilMetodePribadi();
        System.out.println("\nDEMONSTRASI ACCESS MODIFIER");
        System.out.println("✓ Public    : Bisa diakses");
        System.out.println("✓ Default   : Bisa diakses (dalam package sama)");
        System.out.println("✓ Protected : Bisa diakses (dalam package sama)");
        System.out.println("X Private   : TIDAK bisa diakses langsung");
    }
}
```

Contoh Output:



Analisa:

3. Analisis Class Person.java

File ini adalah blueprint atau cetakan untuk membuat objek Person. Tujuan: Mendefinisikan struktur data (atribut) dan perilaku (method) dari sebuah Person.

Package: modul_5.praktikum_1

Atribut (Fields) Class ini mendefinisikan 5 atribut dengan modifier yang berbeda:

private String nama;

Private: Hanya bisa diakses (dibaca/ditulis) dari dalam class Person ini saja.

private int umur;

Private: Sama seperti nama.

String alamat;

Default (Package-Private): Tidak ada modifier tertulis. Bisa diakses oleh class lain selama masih dalam package yang sama (modul_5.praktikum_1).

protected String telepon;

Protected: Bisa diakses oleh class dalam package yang sama DAN oleh subclass (turunan) di package yang berbeda.

public String email;

Public: Bisa diakses dari mana saja, tanpa batasan.

Methods public Person(String nama, int umur)

Constructor Public: Digunakan untuk membuat instance/objek baru dari Person.

public void tampilkanInfo()

Method Public: Bisa dipanggil dari mana saja. Method ini valid karena dia berada di dalam class Person sehingga berhak mengakses anggota private (nama, umur).

private void metodePribadi()

Method Private: Hanya bisa dipanggil dari dalam class Person itu sendiri.

public void panggilMetodePribadi()

Method Public: Digunakan sebagai "jembatan" atau perantara untuk memanggil metodePribadi() yang private dari luar class.

2. Analisis Class AccessModifierTest.java File ini adalah "driver" atau class penguji yang berisi method main.

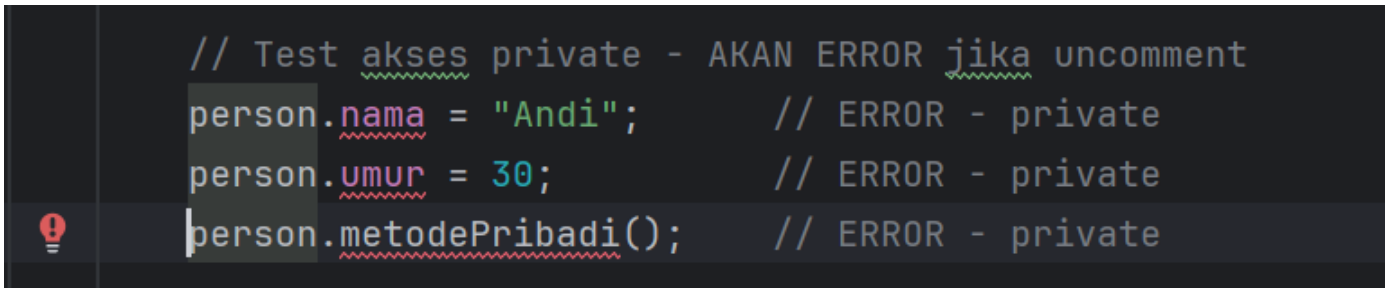
Tujuan: Membuat objek Person dan mencoba mengakses semua atribut dan method-nya untuk menguji aturan Access Modifier.

Package: modul_5.praktikum_1 (Poin penting: Ini ada di package yang sama dengan Person).

Analisis Method main

```java
package modul_5.praktikum_1;

public class AccessModifierTest {  new *
    public static void main(String[] args) {  new *
        Person person = new Person( nama: "Budi Santoso",  umur: 25);

        // Test akses public
        person.email = "budi@email.com"; // OK - public
        System.out.println("Email: " + person.email);

        // Test akses default (dalam package yang sama)
        person.alamat = "Jakarta"; // OK - dalam package yang sama
        System.out.println("Alamat: " + person.alamat);

        // Test akses protected (dalam package yang sama)
        person.telepon = "081234567890"; // OK - dalam package yang sama
        System.out.println("Telepon: " + person.telepon);

        // Test akses private - AKAN ERROR jika uncomment
        // person.nama = "Andi";      // ERROR - private
        // person.umur = 30;          // ERROR - private
        // person.metodePribadi();    // ERROR - private

        // Mengakses data private melalui public method
        person.tampilkanInfo();

        // Mengakses private method melalui public method
        person.panggilMetodePribadi();
```

3. Penjelasan Error: "Coba uncomment baris..." Ini adalah inti dari demonstrasi Enkapsulasi. Jika Anda menghapus tanda // pada baris-baris berikut:



Program tidak akan bisa di-compile. Anda akan mendapatkan compile-time error.

Mengapa Error Terjadi? Penyebab utamanya adalah Pelanggaran Akses private.

Prinsip dasarnya sangat sederhana:

Anggota (atribut atau method) yang ditandai sebagai private HANYA bisa diakses dari dalam class itu sendiri.

Mari kita bedah satu per satu:

person.nama = "Andi"; (ERROR)

Anda mencoba mengakses atribut nama dari objek person.

Atribut nama didefinisikan di class Person sebagai private.

Kode yang mencoba mengaksesnya (person.nama = "Andi";) berada di class AccessModifierTest.

Karena AccessModifierTest bukan class Person, ia tidak memiliki izin untuk mengakses anggota private milik Person.

person.umur = 30; (ERROR)

Alasannya sama persis dengan nama. umur adalah private di Person dan tidak bisa diakses dari luar.

person.metodePribadi(); (ERROR)

Alasannya juga sama. metodePribadi() adalah method private di Person. Hanya method lain di dalam Person (seperti panggilMetodePribadi()) yang boleh memanggilnya.

Analogi Sederhana Bayangkan class Person adalah Rumah Anda:

public (email): Seperti Pagar Depan. Siapa saja di jalan bisa melihatnya.

default (alamat): Seperti Ruang Tamu. Orang yang Anda izinkan masuk ke komplek/package (tetangga) bisa melihatnya.

protected (telepon): Seperti Kamar Tidur Tamu. Tetangga dan keluarga (subclass) boleh mengaksesnya.

private (nama, umur): Seperti Brankas di Kamar Pribadi Anda. HANYA Anda (class Person itu sendiri) yang bisa membukanya.

Ketika class AccessModifierTest (tetangga) mencoba mengakses person.nama, itu sama seperti tetangga Anda mencoba membuka paksa brankas pribadi Anda. Tentu saja tidak bisa dan itu melanggar aturan (error).

Cara Akses yang Benar (Sesuai Konsep OOP) Kode Anda sudah menunjukkan cara yang benar:

Untuk melihat data private: person.tampilkanInfo();

Untuk menjalankan method private: person.panggilMetodePribadi();

Anda berinteraksi melalui "pintu" public yang sudah disediakan, bukan dengan mencoba mendobrak dinding private.

Praktikum 2: Getter dan Setter

Getter dan Setter adalah method yang digunakan untuk mengakses dan mengubah nilai private attributes.

Naming Convention:

- Getter: get + NamaAttribute (contoh: getNama())
- Setter: set + NamaAttribute (contoh: setNama())
- Boolean Getter: is + NamaAttribute (contoh: isActive())

Keuntungan Menggunakan Getter/Setter:

- Kontrol akses terhadap data
- Validasi data sebelum disimpan
- Read-only atau write-only attributes
- Computed attributes
- Lazy initialization

Contoh Program yang mengikuti modul:

1. Class Mahasiswa

```java
package modul_5.praktikum_2;

public class Mahasiswa {  2 usages  new *
    // Atribut
    private String npm;  3 usages
    private String nama;  4 usages
    private String jurusan;  4 usages
    private double ipk;  10 usages
    private int semester;  4 usages
    private boolean aktif;  5 usages

    // Constructor
    public Mahasiswa(String npm, String nama, String jurusan) {  6 usages  new *
        this.npm = npm;
        this.nama = nama;
        this.jurusan = jurusan;
        this.ipk = 0.0;
        this.semester = 1;
        this.aktif = true;
    }

    // Getter Methods
    public String getNpm() {  1 usage  new *
        return npm;
    }
```

```java
    public String getNama() {  new *
        return nama;
    }

    public String getJurusan() {  1 usage  new *
        return jurusan;
    }

    public double getIpk() {  1 usage  new *
        return ipk;
    }

    public int getSemester() {  1 usage  new *
        return semester;
    }

    // Menggunakan awalan 'is' prefix
    public boolean isAktif() {  1 usage  new *
        return aktif;
    }


    // Setter Methods
    // Note: NPM readonly (tidak ada setter)
```

```java
public void setNama(String nama) {  new *
    // Validasi: nama tidak boleh kosong
    if (nama.trim().isEmpty()) {
        System.out.println("Error: Nama tidak boleh kosong!");
        return;
    }

    // Validasi: Nama hanya huruf dan spasi
    if (!nama.matches( regex: "[a-zA-Z .']+")) {
        System.out.println("Error: Nama hanya boleh berisi huruf dan spasi!");
        return;
    }

    this.nama = nama;
    System.out.println("✓ Nama berhasil diubah menjadi: " + nama);
}

public void setJurusan(String jurusan) {  no usages  new *
    // Validasi: Jurusan tidak boleh kosong
    if (jurusan.trim().isEmpty()) {
        System.out.println("Error: Jurusan tidak boleh kosong!");
        return;
    }

    this.jurusan = jurusan;
    System.out.println("✓ Jurusan berhasil diubah menjadi: " + jurusan);
}
```

```java
public void setIpk(double ipk) {  2 usages  new *
    // Validasi: IPK harus valid (0.0 - 4.0)
    if (ipk < 0.0 || ipk > 4.0) {
        System.out.println("Error: IPK harus antara 0.0 - 4.0!");
        return;
    }

    this.ipk = ipk;
    System.out.printf("✓ IPK berhasil diubah menjadi: %.2f%n", ipk);

    // Cek status akademik berdasarkan IPK
    cekStatusAkademik();
}

public void setSemester(int semester) {  2 usages  new *
    // Validasi: Semester harus positif
    if (semester < 1 || semester > 14) {
        System.out.println("Error: Semester harus antara 1 - 14!");
        return;
    }

    this.semester = semester;
    System.out.println("✓ Semester berhasil diubah menjadi: " + semester);
}
```

```java
public void setAktif(boolean aktif) {  1 usage  new *
    this.aktif = aktif;
    System.out.println("✓ Status berhasil diubah menjadi: " + (aktif ? "Aktif" : "Tidak Aktif"));
    System.out.println("Status Mahasiswa: " + getStatus());
}


// Private Method (Helper)
private void cekStatusAkademik() {  1 usage  new *
    if (ipk < 2.0) {
        System.out.println("! Peringatan: IPK di bawah standar!");
    } else if (ipk > 3.5) {
        System.out.println("★ Excellent! IPK sangat baik!");
    }
}


// Public Method
public String getPredikat() {  2 usages  new *
    if (ipk >= 3.5) return "Cum Laude";
    else if (ipk >= 3.0) return "Sangat Memuaskan";
    else if (ipk >= 2.5) return "Memuaskan";
    else if (ipk >= 2.0) return "Cukup";
    else return "Kurang";
}
```

```
    public String getStatus() {  1 usage  new *
        return this.aktif ? "Aktif" : "Tidak Aktif";
    }

    public void tampilkanInfo() {  2 usages  new *
        System.out.println("===== INFORMASI MAHASISWA =====");
        System.out.println("NPM        : " + npm);
        System.out.println("Nama       : " + nama);
        System.out.println("Jurusan    : " + jurusan);
        System.out.printf("IPK        : %.2f (%s)%n", ipk, getPredikat());
        System.out.println("Semester   : " + semester);
        System.out.println("Status     : " + (aktif ? "Aktif" : "Tidak AktIF"));
        System.out.println("================================");
    }
}
```

2. Class GetterSetterTest

```
package modul_5.praktikum_2;

public class GetterSetterTest {  new *
    public static void main(String[] args) {  new *
        System.out.println("TEST GETTER DAN SETTER\n");

        // Membuat object mahasiswa
        Mahasiswa mhs = new Mahasiswa( npm: "20230001",  nama: "Budi Santoso",  jurusan: "Teknik Informatika");

        System.out.println("--- Data Awal ---");
        mhs.tampilkanInfo();

        // Test GETTER
        System.out.println("\n--- TEST GETTER ---");
        System.out.println("Mengambil data menggunakan getter:");
        System.out.println("NPM        : " + mhs.getNpm());
        System.out.println("Nama       : " + mhs.getNama());
        System.out.println("Jurusan    : " + mhs.getJurusan());
        System.out.println("IPK        : " + mhs.getIpk());
        System.out.println("Semester   : " + mhs.getSemester());
        System.out.println("Aktif      : " + mhs.isAktif());


        // Test SETTER dengan validasi
        System.out.println("\n--- TEST SETTER ---");

        // Test 1: Update IPK valid
        System.out.println("\n1. Update IPK (valid):");
        mhs.setIpk(3.75);
```

```java
// Test 2: Update IPK invalid
System.out.println("\n2. Update IPK (invalid):");
mhs.setIpk(5.0); // Akan ditolak

// Test 3: Update nama valid
System.out.println("\n3. Update Nama (valid):");
mhs.setNama("Ahmad Budi Santoso");

// Test 4: Update nama invalid
System.out.println("\n4. Update Nama (invalid - mengandung angka):");
mhs.setNama("Budi123"); // Akan ditolak

// Test 5: Update semester
System.out.println("\n5. Update Semester:");
mhs.setSemester(5);

// Test 6: Update semester invalid
System.out.println("\n6. Update Semester (invalid):");
mhs.setSemester(20); // Akan ditolak

// Test 7: Update status
System.out.println("\n7. Update Status:");
mhs.setAktif(false);

// Tampilkan data akhir
System.out.println("\n--- Data Setelah Update ---");
mhs.tampilkanInfo();
```

```
        // Demonstrasi computed property
        System.out.println("\n--- COMPUTED PROPERTY ---");
        System.out.println("Predikat  : " + mhs.getPredikat());

        // Tidak bisa akses langsung (akan error)
        System.out.println("\nCATATAN ENCAPSULATION:");
        System.out.println("X TIDAK BISA: mhs.npm = \"123\";");
        System.out.println("X TIDAK BISA: mhs.ipk = 5.0;");
        System.out.println("✓ HARUS: mhs.setIpk(3.5);");
        System.out.println("✓ VALIDASI otomatis dilakukan di setter");
    }
}
```

Output:

```
"C:\Program Files\Amazon Corretto\jdk21.0.8_9\bin\java.exe"
TEST GETTER DAN SETTER

--- Data Awal ---
===== INFORMASI MAHASISWA =====
NPM        : 20230001
Nama       : Budi Santoso
Jurusan    : Teknik Informatika
IPK        : 0,00 (Kurang)
Semester   : 1
Status     : Aktif
================================

--- TEST GETTER ---
Mengambil data menggunakan getter:
NPM        : 20230001
Nama       : Budi Santoso
Jurusan    : Teknik Informatika
IPK        : 0.0
Semester   : 1
Aktif      : true
```

```
--- TEST SETTER ---

1. Update IPK (valid):
✓ IPK berhasil diubah menjadi: 3,75
★ Excellent! IPK sangat baik!

2. Update IPK (invalid):
Error: IPK harus antara 0.0 - 4.0!

3. Update Nama (valid):
✓ Nama berhasil diubah menjadi: Ahmad Budi Santoso

4. Update Nama (invalid - mengandung angka):
Error: Nama hanya boleh berisi huruf dan spasi!

5. Update Semester:
✓ Semester berhasil diubah menjadi: 5

6. Update Semester (invalid):
Error: Semester harus antara 1 - 14!

7. Update Status:
✓ Status berhasil diubah menjadi: Tidak Aktif
Status Mahasiswa: Tidak Aktif
```

```
--- Data Setelah Update ---
===== INFORMASI MAHASISWA =====
NPM        : 20230001
Nama       : Ahmad Budi Santoso
Jurusan    : Teknik Informatika
IPK        : 3,75 (Cum Laude)
Semester   : 5
Status     : Tidak AktIF
==============================

--- COMPUTED PROPERTY ---
Predikat   : Cum Laude

CATATAN ENCAPSULATION:
X TIDAK BISA: mhs.npm = "123";
X TIDAK BISA: mhs.ipk = 5.0;
✓ HARUS: mhs.setIpk(3.5);
✓ VALIDASI otomatis dilakukan di setter

Process finished with exit code 0
```

Analisa:

Class Mahasiswa.java Ini adalah class blueprint yang mendefinisikan objek Mahasiswa.

Perlindungan Data (Enkapsulasi) Semua atribut (data) dideklarasikan sebagai private:

private String npm; private String nama; private String jurusan; private double ipk; private int semester; private boolean aktif; Ini berarti mereka tidak bisa diakses atau diubah secara langsung dari luar class Mahasiswa.

1. Getter: Berfungsi untuk Membaca Data Untuk mengizinkan class lain membaca data yang private, kita menyediakan Getter Methods.

public String getNpm()

public String getNama()

public String getJurusan()

public double getIpk()

public int getSemester()

public boolean isAktif() (Khusus untuk boolean, konvensinya menggunakan awalan is bukan get).

Metode-metode ini bersifat public dan satu-satunya tugas mereka adalah mengembalikan nilai dari atribut privat.

2. Setter: Melakukan Validasi Sebelum Mengubah Data Untuk mengizinkan class lain mengubah data, kita menyediakan Setter Methods. Inilah letak kekuatan enkapsulasi: kita bisa menyisipkan logika validasi di dalamnya.

Contoh 1: Validasi IPK

```
public void setIpk(double ipk) {  2 usages  new *
    // Validasi: IPK harus valid (0.0 - 4.0)
    if (ipk < 0.0 || ipk > 4.0) {
        System.out.println("Error: IPK harus antara 0.0 - 4.0!");
        return;
    }

    this.ipk = ipk;
    System.out.printf("✓ IPK berhasil diubah menjadi: %.2f%n", ipk);

    // Cek status akademik berdasarkan IPK
    cekStatusAkademik();
}
```

Contoh 2: Validasi Nama

```
public void setNama(String nama) {  new *
    // Validasi: nama tidak boleh kosong
    if (nama.trim().isEmpty()) {
        System.out.println("Error: Nama tidak boleh kosong!");
        return;
    }

    // Validasi: Nama hanya huruf dan spasi
    if (!nama.matches( regex: "[a-zA-Z .']+")) {
        System.out.println("Error: Nama hanya boleh berisi huruf dan spasi!");
        return;
    }

    this.nama = nama;
    System.out.println("✓ Nama berhasil diubah menjadi: " + nama);
}
```

Jika Anda mencoba setNama("Budi123"), validasi regex akan gagal, dan nama tidak akan diubah.

3. Attribute Read-Only (Tidak Ada Setter) Perhatikan atribut npm.

Ada public String getNpm() (ada Getter).

Tidak ada public void setNpm(String npm) (tidak ada Setter).

Artinya, npm hanya bisa diatur satu kali saat objek dibuat (melalui constructor). Setelah itu, npm bersifat read-only: bisa dibaca oleh siapa saja (via getNpm()), tapi tidak bisa diubah lagi.

2. Analisis Class GetterSetterTest.java Ini adalah class "driver" yang berisi main untuk menguji class Mahasiswa.

Pengamatan Saat Program Dijalankan Data Awal & Test Getter:

Objek mhs dibuat dengan data awal.

Blok "TEST GETTER" membuktikan bahwa kita bisa membaca semua data awal (Budi Santoso, 0.0, true, dll.) menggunakan method get... dan is....

Test Setter (Poin Kunci):

mhs.setIpk(3.75);

Hasil: BERHASIL. Angka 3.75 lolos validasi (0.0 <= ipk <= 4.0).

Output: "✓ IPK berhasil diubah..." dan "★ Excellent! IPK sangat baik!" (karena setIpk memanggil cekStatusAkademik()).

mhs.setIpk(5.0);

Hasil: DITOLAK. Angka 5.0 gagal validasi.

Output: "Error: IPK harus antara 0.0 - 4.0!".

mhs.setNama("Budi123");

Hasil: DITOLAK. String "Budi123" gagal validasi regex [a-zA-Z .']+ (karena mengandung angka).

Output: "Error: Nama hanya boleh berisi huruf dan spasi!".

mhs.setSemester(20);

Hasil: DITOLAK. Angka 20 gagal validasi (1 <= semester <= 14).

Output: "Error: Semester harus antara 1 - 14!".

Data Akhir:

Ketika mhs.tampilkanInfo() dipanggil di akhir, kita melihat:

IPK adalah 3.75 (bukan 0.0 dan bukan 5.0).

Nama tetap "Ahmad Budi Santoso" (bukan "Budi123").

Semester adalah 5 (bukan 1 dan bukan 20).

Ini membuktikan bahwa setter berhasil melindungi data dari nilai yang tidak valid.

Larangan Akses Langsung:

Baris // X TIDAK BISA: mhs.npm = "123"; menegaskan ini. Jika Anda menghapus //, program akan error karena npm adalah private.

Anda dipaksa menggunakan mhs.setIpk(3.5) (yang memiliki validasi) dan dilarang melakukan mhs.ipk = 5.0 (yang berbahaya).

Kesimpulan Program ini mendemonstrasikan fondasi enkapsulasi:

Getter (getNama(), isAktif()) adalah "jendela" publik untuk melihat data privat.

Setter (setNama(), setIpk()) adalah "pintu" publik untuk mengubah data privat, yang dijaga oleh "satpam" (logika validasi).

Read-Only (seperti npm) dicapai dengan hanya menyediakan "jendela" (Getter) tapi tidak ada "pintu" (Setter).

Manfaat utamanya adalah integritas data. Objek Mahasiswa tidak akan pernah bisa memiliki data "sampah" atau tidak valid (seperti IPK 5.0 atau semester 20), karena setter telah mencegahnya di pintu masuk.

Praktikum 3: Read-Only dan Write-Only Properties

Memahami konsep read-only dan write-only properties menggunakan getter/setter.
Contoh Program yang mengikuti modul:

1. Class Product

```java
package modul_5.praktikum_3;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Product {  4 usages  new *

    // Read-only (hanya getter)
    private final String productId;  3 usages
    private final LocalDateTime createdAt;  2 usages

    // Read-write (getter dan setter)
    private String nama;  4 usages
    private double harga;  5 usages
    private int stok;  15 usages

    // Write-only (hanya setter) - untuk password/PIN
    private String adminPin;  3 usages

    // Computed property (tidak ada attribute, hanya getter)
    // Total nilai = harga * stok

    // Counter untuk ID otomatis
    private static int counter = 1000;  2 usages

    // CONSTRUCTOR
    public Product(String nama, double harga, int stok, String adminPin) {  3 usages  new *
        this.productId = generateProductId();
        this.createdAt = LocalDateTime.now();
```

```
        this.nama = nama;
        this.harga = harga;
        this.stok = stok;
        this.adminPin = adminPin;
    }

    // PRIVATE HELPER METHODS
    private String generateProductId() {  1 usage  new *
        counter++;
        return "PRD-" + counter;
    }

    private boolean validatePin(String inputPin) {  5 usages  new *
        return this.adminPin.equals(inputPin);
    }

    // READ-ONLY PROPERTIES (hanya getter)
    public String getProductId() {  1 usage  new *
        return productId;
    }

    public String getCreatedAt() {  2 usages  new *
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        return createdAt.format(formatter);
    }
```

```java
    // READ-WRITE PROPERTIES (getter dan setter)
    public String getNama() {  new *
        return nama;
    }

    public void setNama(String nama, String pin) {  1 usage  new *
        if (!validatePin(pin)) {
            System.out.println("X PIN salah! Gagal mengubah nama.");
            return;
        }

        if (nama == null || nama.trim().isEmpty()) {
            System.out.println("X Nama produk tidak boleh kosong!");
            return;
        }

        this.nama = nama;
        System.out.println("✓ Nama produk berhasil diubah");
    }

    public double getHarga() {  no usages  new *
        return harga;
    }

    public void setHarga(double harga, String pin) {  2 usages  new *
        if (!validatePin(pin)) {
            System.out.println("X PIN salah! Gagal mengubah harga.");
            return;
```

```
        }

        if (harga < 0) {
            System.out.println("X Harga tidak boleh negatif!");
            return;
        }

        this.harga = harga;
        System.out.printf("✓ Harga berhasil diubah menjadi Rp %.2f%n", harga);
    }

    public int getStok() {  no usages  new *
        return stok;
    }

    public void setStok(int stok, String pin) {  1 usage  new *
        if (!validatePin(pin)) {
            System.out.println("X PIN salah! Gagal mengubah stok.");
            return;
        }

        if (stok < 0) {
            System.out.println("X Stok tidak boleh negatif!");
            return;
        }
```

```
        this.stok = stok;
        System.out.println("✓ Stok berhasil diubah menjadi " + stok);
    }

    // WRITE-ONLY PROPERTY (hanya setter)
    // Tidak ada getAdminPin() - untuk keamanan!
    public void changeAdminPin(String oldPin, String newPin) {  2 usages  new *
        if (!validatePin(oldPin)) {
            System.out.println("X PIN lama salah!");
            return;
        }

        if (newPin.length() < 4) {
            System.out.println("X PIN baru minimal 4 karakter!");
            return;
        }

        this.adminPin = newPin;
        System.out.println("✓ PIN berhasil diubah");
    }

    // COMPUTED PROPERTIES (calculated on-the-fly)
    public double getTotalNilai() {  3 usages  new *
        return harga * stok;
    }
```

```java
public String getStatusStok() {  3 usages  new *
    if (stok == 0) return "HABIS";
    else if (stok < 10) return "MENIPIS";
    else if (stok < 50) return "TERSEDIA";
    else return "BANYAK";
}

// PUBLIC METHODS
public boolean tambahStok(int jumlah, String pin) {  1 usage  new *
    if (!validatePin(pin)) {
        System.out.println("X PIN salah!");
        return false;
    }

    if (jumlah <= 0) {
        System.out.println("X Jumlah harus lebih dari 0!");
        return false;
    }

    stok += jumlah;
    System.out.printf("✓ Stok ditambah %d. Stok sekarang: %d%n", jumlah, stok);
    return true;
}

public boolean kurangiStok(int jumlah) {  3 usages  new *
    if (jumlah <= 0) {
        System.out.println("X Jumlah harus lebih dari 0!");
        return false;
```

```
        }

        if (jumlah > stok) {
            System.out.println("X Stok tidak cukup!");
            System.out.printf("  Stok tersedia: %d, Diminta: %d%n", stok, jumlah);
            return false;
        }

        stok -= jumlah;
        System.out.printf("✓ Stok dikurangi %d. Stok sekarang: %d%n", jumlah, stok);

        // Warning jika stok menipis
        if (stok < 10) {
            System.out.println("▲ Peringatan: Stok menipis!");
        }

        return true;
    }

    public void tampilkanInfo() {  4 usages  new *
        System.out.println("-----------------INFORMASI PRODUK-----------------");
        System.out.println("Product ID : " + productId);
        System.out.println("Nama       : " + nama);
        System.out.printf("Harga      : Rp %.2f%n", harga);
        System.out.println("Stok       : " + stok + " (" + getStatusStok() + ")");
        System.out.printf("Total Nilai: Rp %.2f%n", getTotalNilai());
        System.out.println("Dibuat pada: " + getCreatedAt());
        System.out.println("------------------------------------------------");
    }
}
```

2. Class ProductTest

```java
package modul_5.praktikum_3;

import java.util.Scanner;

public class ProductTest {  new *
    public static void main(String[] args) {  new *
        // Scanner input = new Scanner(System.in);

        System.out.println("SISTEM MANAJEMEN PRODUK");

        // Membuat produk dengan PIN
        Product laptop = new Product( nama: "Laptop ASUS",  harga: 8500000,  stok: 15,  adminPin: "1234");
        Product mouse = new Product( nama: "Mouse Logitech",  harga: 150000,  stok: 50,  adminPin: "1234");

        System.out.println("\n--- PRODUK BERHASIL DIBUAT---");
        laptop.tampilkanInfo();
        mouse.tampilkanInfo();

        // TEST READ-ONLY PROPERTIES
        System.out.println("\n--- TEST READ-ONLY PROPERTIES ---");
        System.out.println("Product ID (read-only): " + laptop.getProductId());
        System.out.println("Created At (read-only): " + laptop.getCreatedAt());
        // laptop.productId = "P-001"; // Ini TIDAK BISA diubah setelah dibuat
        System.out.println("✓ Properti ini tidak bisa diubah setelah dibuat");

        // TEST READ-WRITE PROPERTIES
        System.out.println("\n--- TEST READ-WRITE PROPERTIES ---");
```

```java
        System.out.println("\n1. Mencoba ubah harga dengan PIN salah:");
        mouse.setHarga( harga: 160000,  pin: "0000"); // PIN salah

        // Test 2: Ubah harga dengan PIN benar
        System.out.println("\n2. Ubah harga dengan PIN benar:");
        mouse.setHarga( harga: 165000,  pin: "1234");

        // Test 3: Ubah nama
        System.out.println("\n3. Ubah nama produk:");
        laptop.setNama( nama: "Laptop ASUS ROG",  pin: "1234");

        // Test 4: Ubah stok
        System.out.println("\n4. Ubah stok:");
        laptop.setStok( stok: 10,  pin: "1234");

        // TEST WRITE-ONLY PROPERTY
        System.out.println("\n--- TEST WRITE-ONLY PROPERTY (PIN) ---");
        // System.out.println(laptop.adminPin); // ERROR (Tidak ada getter)
        System.out.println("Hanya bisa diubah dengan validasi PIN lama:");
        System.out.println("Coba ganti PIN (gagal):");
        laptop.changeAdminPin( oldPin: "0000",  newPin: "9999"); // Akan gagal

        System.out.println("\nCoba ganti PIN (sukses):");
        laptop.changeAdminPin( oldPin: "1234",  newPin: "5678");
```

```
        // TEST COMPUTED PROPERTIES
        System.out.println("\n--- TEST COMPUTED PROPERTIES ---");
        System.out.printf("Total Nilai Laptop : Rp %.2f%n", laptop.getTotalNilai());
        System.out.println("Status Stok Laptop : " + laptop.getStatusStok());
        System.out.printf("Total Nilai Mouse  : Rp %.2f%n", mouse.getTotalNilai());
        System.out.println("Status Stok Mouse  : " + mouse.getStatusStok());


        // TEST BUSINESS METHODS
        System.out.println("\n--- TEST BUSINESS METHODS ---");

        System.out.println("\n1. Tambah stok laptop:");
        laptop.tambahStok( jumlah: 10,  pin: "5678"); // PIN sudah diubah

        System.out.println("\n2. Kurangi stok mouse (penjualan):");
        mouse.kurangiStok( jumlah: 45);

        System.out.println("\n3. Kurangi stok mouse lagi (stok akan menipis):");
        mouse.kurangiStok( jumlah: 4);

        System.out.println("\n4. Coba kurangi stok lebih dari tersedia:");
        mouse.kurangiStok( jumlah: 10);

        // TAMPILKAN INFO AKHIR
        System.out.println("\n--- INFORMASI PRODUK SETELAH UPDATE ---");
        laptop.tampilkanInfo();
        mouse.tampilkanInfo();
```

```
        // RINGKASAN ENCAPSULATION
        System.out.println("\n--- RINGKASAN ENCAPSULATION ---");
        System.out.println("---------------------------------");
        System.out.println("✓ READ-ONLY     : productId, createdAt      ");
        System.out.println("  (getter✓, setter✗)  (tidak bisa diubah)       ");
        System.out.println();
        System.out.println("✓ READ-WRITE    : nama, harga, stok         ");
        System.out.println("  (getter✓, setter✓)  (dengan validasi)         ");
        System.out.println();
        System.out.println("✓ WRITE-ONLY    : adminPin                  ");
        System.out.println("  (getter✗, setter✓)  (tidak bisa dibaca)       ");
        System.out.println();
        System.out.println("✓ COMPUTED      : totalNilai, statusStok  ");
        System.out.println("  (getter✓, setter✗)  (dihitung otomatis)       ");
        System.out.println("---------------------------------");
    }
}
```

OutPut:

```
"C:\Program Files\Amazon Corretto\jdk21.0.8_9\bin\java.exe"
SISTEM MANAJEMEN PRODUK

--- PRODUK BERHASIL DIBUAT---
-----------------INFORMASI PRODUK-----------------
Product ID : PRD-1001
Nama       : Laptop ASUS
Harga      : Rp 8500000,00
Stok       : 15 (TERSEDIA)
Total Nilai: Rp 127500000,00
Dibuat pada: 02-11-2025 15:12:24
------------------------------------------------
-----------------INFORMASI PRODUK-----------------
Product ID : PRD-1002
Nama       : Mouse Logitech
Harga      : Rp 150000,00
Stok       : 50 (BANYAK)
Total Nilai: Rp 7500000,00
Dibuat pada: 02-11-2025 15:12:24
------------------------------------------------
```

```
--- TEST READ-ONLY PROPERTIES ---
Product ID (read-only): PRD-1001
Created At (read-only): 02-11-2025 15:12:24
✓ Properti ini tidak bisa diubah setelah dibuat

--- TEST READ-WRITE PROPERTIES ---

1. Mencoba ubah harga dengan PIN salah:
X PIN salah! Gagal mengubah harga.

2. Ubah harga dengan PIN benar:
✓ Harga berhasil diubah menjadi Rp 165000,00

3. Ubah nama produk:
✓ Nama produk berhasil diubah

4. Ubah stok:
✓ Stok berhasil diubah menjadi 10

--- TEST WRITE-ONLY PROPERTY (PIN) ---
Hanya bisa diubah dengan validasi PIN lama:
Coba ganti PIN (gagal):
X PIN lama salah!
```

```
Coba ganti PIN (sukses):
✓ PIN berhasil diubah

--- TEST COMPUTED PROPERTIES ---
Total Nilai Laptop : Rp 85000000,00
Status Stok Laptop : TERSEDIA
Total Nilai Mouse  : Rp 8250000,00
Status Stok Mouse  : BANYAK

--- TEST BUSINESS METHODS ---

1. Tambah stok laptop:
✓ Stok ditambah 10. Stok sekarang: 20

2. Kurangi stok mouse (penjualan):
✓ Stok dikurangi 45. Stok sekarang: 5
▲ Peringatan: Stok menipis!

3. Kurangi stok mouse lagi (stok akan menipis):
✓ Stok dikurangi 4. Stok sekarang: 1
▲ Peringatan: Stok menipis!
```

```
4. Coba kurangi stok lebih dari tersedia:
X Stok tidak cukup!
  Stok tersedia: 1, Diminta: 10

--- INFORMASI PRODUK SETELAH UPDATE ---
-----------------INFORMASI PRODUK-----------------
Product ID : PRD-1001
Nama       : Laptop ASUS ROG
Harga      : Rp 8500000,00
Stok       : 20 (TERSEDIA)
Total Nilai: Rp 170000000,00
Dibuat pada: 02-11-2025 15:12:24
--------------------------------------------------
-----------------INFORMASI PRODUK-----------------
Product ID : PRD-1002
Nama       : Mouse Logitech
Harga      : Rp 165000,00
Stok       : 1 (MENIPIS)
Total Nilai: Rp 165000,00
Dibuat pada: 02-11-2025 15:12:24
--------------------------------------------------
```

```
--- RINGKASAN ENCAPSULATION ---
--------------------------------
✓ READ-ONLY     : productId, createdAt
  (getter✓, setter✗)  (tidak bisa diubah)

✓ READ-WRITE    : nama, harga, stok
  (getter✓, setter✓)  (dengan validasi)

✓ WRITE-ONLY    : adminPin
  (getter✗, setter✓)  (tidak bisa dibaca)

✓ COMPUTED      : totalNilai, statusStok
  (getter✓, setter✗)  (dihitung otomatis)
--------------------------------

Process finished with exit code 0
```

Analisa:

Program Product.java dan ProductTest.java merupakan implementasi yang sangat baik untuk mendemonstrasikan konsep Encapsulation dalam pemrograman berorientasi objek (OOP), khususnya dalam penggunaan Access Modifier dan kontrol akses terhadap atribut kelas. Konsep utama yang ditunjukkan adalah bagaimana read-only, write-only, read-write, dan computed properties bekerja dalam menjaga keamanan dan konsistensi data suatu objek, serta bagaimana proses validasi PIN digunakan untuk memastikan perubahan data hanya dilakukan oleh pihak yang berwenang.

Pertama, atribut read-only seperti productId dan createdAt hanya memiliki metode getter tanpa setter. Nilai keduanya ditetapkan sekali pada saat objek dibuat melalui konstruktor dan tidak dapat diubah lagi setelahnya. Hal ini ditunjukkan ketika program mencoba mengubah productId, yang menghasilkan error karena atribut tersebut memiliki modifier final dan tidak menyediakan metode pengubah (setter). Dengan demikian, atribut ini bersifat konstan sepanjang masa hidup objek dan menjamin identitas produk tetap konsisten.

Selanjutnya, atribut write-only seperti adminPin hanya memiliki setter tanpa getter. Hal ini berarti nilainya dapat diubah, tetapi tidak dapat dibaca secara langsung dari luar kelas. Tujuan dari pendekatan ini adalah menjaga keamanan informasi sensitif seperti PIN agar tidak terekspos ke luar. Di dalam program, pengubahan PIN dilakukan menggunakan metode changeAdminPin(), yang mensyaratkan PIN lama dan PIN baru. Apabila PIN lama salah, maka perubahan akan ditolak. Pendekatan ini mencontohkan bagaimana enkapsulasi dapat digunakan untuk melindungi data sensitif dengan memberikan kontrol penuh terhadap akses data.

Selain itu, program juga menampilkan konsep computed properties, yaitu properti yang nilainya tidak disimpan secara langsung dalam atribut, melainkan dihitung setiap kali dibutuhkan. Contohnya adalah getTotalNilai() yang menghitung

total nilai barang berdasarkan harga dikalikan stok, serta getStatusStok() yang menilai kondisi stok berdasarkan jumlah yang tersedia. Karena nilainya dihitung secara langsung dari atribut lain, computed properties selalu memberikan informasi yang mutakhir tanpa memerlukan penyimpanan tambahan. Ini merupakan contoh penggunaan properti dinamis dalam desain kelas yang efisien.

Atribut read-write seperti nama, harga, dan stok memiliki baik getter maupun setter, namun dengan tambahan logika validasi berupa PIN. Setiap kali pengguna ingin mengubah nilai atribut tersebut, program akan memeriksa apakah PIN yang dimasukkan sesuai dengan PIN admin yang tersimpan. Jika PIN salah, perubahan ditolak dan pesan kesalahan akan ditampilkan. Validasi ini menjadi mekanisme pengaman agar hanya pengguna yang memiliki otoritas yang dapat melakukan perubahan terhadap data produk. Dengan cara ini, program memastikan bahwa data penting tidak bisa dimodifikasi sembarangan, mencerminkan praktik keamanan data yang baik dalam OOP.

Secara keseluruhan, hasil dari eksekusi program memperlihatkan bahwa read-only properties tidak dapat diubah, write-only properties tidak dapat dibaca, computed properties dihitung secara otomatis (on-the-fly), dan setiap perubahan data harus melalui validasi PIN yang benar. Semua perilaku ini membuktikan bahwa kelas Product dirancang dengan prinsip enkapsulasi yang kuat — melindungi data internal, memberikan antarmuka publik yang terkontrol, dan memastikan integritas serta keamanan data melalui penggunaan Access Modifier dan validasi logika bisnis yang tepat.

Kesimpulan

Dari seluruh rangkaian praktikum mengenai Enkapsulasi (Encapsulation), dapat disimpulkan bahwa konsep ini merupakan salah satu pilar utama dalam pemrograman berorientasi objek yang berperan penting dalam menjaga keamanan, konsistensi, dan integritas data. Melalui penggunaan access modifier seperti private, protected, public, dan default, programmer dapat mengontrol secara ketat bagaimana data diakses dan dimodifikasi oleh bagian lain dari program.

Praktikum pertama menunjukkan bagaimana setiap jenis access modifier memiliki tingkat akses yang berbeda, dan bagaimana pelanggaran terhadap aturan akses tersebut akan menimbulkan error — hal ini mempertegas fungsi enkapsulasi dalam melindungi atribut dari manipulasi langsung. Praktikum kedua memperkenalkan getter dan setter sebagai mekanisme aman untuk membaca dan mengubah data privat. Dengan menambahkan validasi pada setter, data yang masuk dapat disaring sehingga hanya nilai yang valid yang diterima oleh sistem.

Sementara itu, praktikum ketiga memperdalam konsep read-only, write-only, dan computed properties, serta penerapan validasi keamanan menggunakan PIN untuk mengontrol perubahan data. Pendekatan ini mencontohkan bagaimana enkapsulasi tidak hanya melindungi data, tetapi juga mengatur alur logika bisnis dengan aman dan efisien.

Secara keseluruhan, penerapan enkapsulasi membuat kode menjadi lebih aman, mudah dipelihara, fleksibel, dan terstruktur. Data penting tidak dapat diubah secara sembarangan, sementara perubahan yang diizinkan selalu melalui jalur kontrol yang jelas. Dengan demikian, enkapsulasi bukan hanya tentang menyembunyikan data, tetapi juga tentang menciptakan sistem yang stabil, terpercaya, dan berorientasi pada keamanan informasi dalam pengembangan perangkat lunak berbasis objek.

Referensi

Oracle Corporation. (2024). The Java™ Tutorials: Controlling Access to Members of a Class. Diakses pada 6 November 2025, dari https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html Oracle Corporation. (2024). The

Java™ Tutorials: Encapsulation. Diakses pada 6 November 2025, dari https://www.oracle.com/java/technologies/tutorial-object-oriented-concepts.html#encapsulation