MATERI PEMBELAJARAN

PROGRAMMER

ELEMEN PEMROGRAMAN BERORIENTASI OBJEK

Oleh:

Dimas Adiz Setiawan

NIM 22050974088

UNIVERSITAS NEGERI SURABAYA

FAKULTAS TEKNIK

PROGRAM STUDI S1 PENDIDIKAN TEKNOLOGI INFORMASI

2026

PROGRAMMER

Elemen	:	Pemrograman Berorientasi Objek
Capaian Pembelajaran	:	1. Memahami konsep dasar OOP seperti class, attribute, method, serta prinsip enkapsulasi, inheritance, dan polymorphism tingkat dasar. 2. Merancang struktur program berorientasi objek memlalui pembuatan class diagram sederhana dan penentuan relasi antar class. 3. Mengimplementasikan program berbasis objek menggunakan class, object, constructor, overloading, inheritance, polymorphism, dan enkapsulasi sesuai studi kasus.
Kelas/Fase	:	XI/F
Tujuan Pembelajaran	:	Peserta didik mampu memahami konsep dasar OOP, merancang struktur program melalui class diagram sederhana, serta mengimplementasikan program berbasis objek dengan menerapkan prinsip enkapsulasi, inheritance, dan polymorphism sesuai studi kasus.

MATERI PEMBELAJARAN
PEMROGRAMAN BERORIENTASI OBJEK
Menggunakan Bahasa Java

MEMBUAT PROGRAM BERORIENTASI OBJEK DENGAN MEMANFAATKAN CLASS

1.1 Pengenalan Class dan Object

Apa itu Class?

Class adalah blueprint atau template untuk membuat objek dalam pemrograman berorientasi objek (OOP). Class mendefinisikan struktur dan perilaku yang akan dimiliki oleh objek-objek yang dibuat darinya.

Apa itu Object?

Object adalah instance atau perwujudan nyata dari sebuah class. Jika class adalah cetakan kue, maka object adalah kue yang sudah jadi dari cetakan tersebut.

Ilustrasi Sederhana:

- • Class Mobil = Blueprint mobil yang mendefinisikan mobil punya warna, merk, kecepatan
- • Object mobil1 = Mobil merah merk Toyota dengan kecepatan 120 km/jam
- • Object mobil2 = Mobil biru merk Honda dengan kecepatan 100 km/jam

1.2 Membuat Class dalam Java

Sintaks dasar untuk membuat class dalam Java:

```
public class NamaClass {

    // Atribut (variabel)

    tipeData namaVariabel;


    // Constructor

    public NamaClass() {

        // Inisialisasi

    }


    // Method (fungsi)

    public void namaMethod() {

        // Logika method

    }
```

```
}
```

Contoh Class Sederhana:

```
public class Mahasiswa {
    // Atribut
    String nama;
    String nim;
    double ipk;

    // Method untuk menampilkan info
    public void tampilkanInfo() {
        System.out.println("Nama: " + nama);
        System.out.println("NIM: " + nim);
        System.out.println("IPK: " + ipk);
    }
}
```

1.3 Properti Class (Atribut dan Method)

A. Atribut (Variabel Instance)

Atribut adalah variabel yang menyimpan data atau karakteristik dari sebuah objek. Atribut juga disebut sebagai field atau instance variable.

Contoh atribut pada class Mahasiswa:

- • nama (String) - menyimpan nama mahasiswa
- • nim (String) - menyimpan nomor induk mahasiswa
- • ipk (double) - menyimpan indeks prestasi kumulatif

B. Method (Fungsi)

Method adalah fungsi atau prosedur yang mendefinisikan perilaku atau aksi yang dapat dilakukan oleh sebuah objek.

Jenis-jenis Method:

- • Method tanpa return value (void)
- • Method dengan return value
- • Method dengan parameter
- • Method tanpa parameter

Contoh Lengkap Method:

```
public class Kalkulator {


    // Method void tanpa parameter

    public void tampilkanMenu() {

        System.out.println("=== KALKULATOR ===");

    }


    // Method dengan return value dan parameter

    public int tambah(int a, int b) {

        return a + b;

    }


    // Method dengan return value tanpa parameter

    public double getPi() {

        return 3.14159;

    }

}
```

1.4 Hak Akses (Access Modifier)

Access modifier menentukan tingkat akses atau visibility dari class, atribut, dan method. Java memiliki 4 jenis access modifier:

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓

protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

Penjelasan Detail:

1. public

Dapat diakses dari mana saja (class lain, package lain, bahkan di luar project)

```
public class Mobil {
    public String merk;  // Bisa diakses dari mana saja

    public void jalan() {
        System.out.println("Mobil berjalan");
    }
}
```

2. private

Hanya dapat diakses dalam class yang sama. Ini adalah prinsip enkapsulasi yang baik.

```
public class RekeningBank {
    private double saldo;  // Tidak bisa diakses langsung dari luar

    // Akses saldo melalui method public
    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        if (saldo >= 0) {
            this.saldo = saldo;
```

```
        }
    }
}
```

3. protected

Dapat diakses dalam package yang sama dan oleh subclass (turunan class) meskipun berbeda package.

```
public class Kendaraan {
    protected int kecepatanMaks;  // Bisa diakses subclass
}
```

4. default (package-private)

Jika tidak ada modifier, maka akses hanya dalam package yang sama.

```
class Helper {  // Tidak ada modifier = default
    int nilai;  // Hanya bisa diakses dalam package yang sama
}
```

1.5 Constructor

Constructor adalah method khusus yang dipanggil secara otomatis ketika objek dibuat. Constructor memiliki nama yang sama dengan nama class dan tidak memiliki return type.

Karakteristik Constructor:

- • Nama constructor sama dengan nama class
- • Tidak memiliki return type (bahkan void)
- • Dipanggil otomatis saat objek dibuat dengan kata kunci "new"
- • Digunakan untuk inisialisasi atribut objek

Jenis-jenis Constructor:

1. Default Constructor

Constructor tanpa parameter. Jika tidak ada constructor yang didefinisikan, Java otomatis membuat default constructor.

```
public class Mahasiswa {
```

```
    String nama;
    String nim;

    // Default constructor
    public Mahasiswa() {
        nama = "Belum diisi";
        nim = "00000000";
    }
}
```

2. Parameterized Constructor

Constructor dengan parameter untuk menginisialisasi objek dengan nilai tertentu.

```
public class Mahasiswa {
    String nama;
    String nim;
    double ipk;

    // Parameterized constructor
    public Mahasiswa(String nama, String nim, double ipk) {
        this.nama = nama;  // 'this' merujuk ke atribut class
        this.nim = nim;
        this.ipk = ipk;
    }
}
```

Penggunaan Constructor:

```
public class Main {
```

```
    public static void main(String[] args) {
        // Membuat objek dengan default constructor
        Mahasiswa mhs1 = new Mahasiswa();

        // Membuat objek dengan parameterized constructor
        Mahasiswa mhs2 = new Mahasiswa("Budi", "12345678", 3.75);
    }
}
```

1.6 Getter dan Setter (Enkapsulasi)

Getter dan Setter adalah method khusus untuk mengakses dan mengubah nilai atribut private. Ini adalah implementasi dari prinsip enkapsulasi dalam OOP.

Mengapa Perlu Getter dan Setter?

- • Melindungi data dari akses langsung yang tidak terkontrol
- • Memberikan validasi sebelum mengubah nilai
- • Memberikan fleksibilitas untuk mengubah implementasi internal
- • Memungkinkan read-only atau write-only properties

Contoh Implementasi Lengkap:

```
public class RekeningBank {
    private String nomorRekening;
    private String namaPemilik;
    private double saldo;

    // Constructor
    public RekeningBank(String noRek, String nama) {
        this.nomorRekening = noRek;
        this.namaPemilik = nama;
        this.saldo = 0.0;
    }
```

```
// Getter untuk nomor rekening
public String getNomorRekening() {
    return nomorRekening;
}

// Getter untuk nama pemilik
public String getNamaPemilik() {
    return namaPemilik;
}

// Setter untuk nama pemilik dengan validasi
public void setNamaPemilik(String nama) {
    if (nama != null && !nama.isEmpty()) {
        this.namaPemilik = nama;
    }
}

// Getter untuk saldo
public double getSaldo() {
    return saldo;
}

// Method untuk setor (bukan setter langsung)
public void setor(double jumlah) {
    if (jumlah > 0) {
        saldo += jumlah;
        System.out.println("Setor berhasil. Saldo: " + saldo);
```

```
        }
    }

    // Method untuk tarik
    public void tarik(double jumlah) {
        if (jumlah > 0 && jumlah <= saldo) {
            saldo -= jumlah;
            System.out.println("Tarik berhasil. Saldo: " + saldo);
        } else {
            System.out.println("Saldo tidak cukup!");
        }
    }
}
```

MENGGUNAKAN TIPE DATA DAN CONTROL PROGRAM

2.1 Tipe Data dalam Java

Java memiliki dua kategori utama tipe data:

A. Tipe Data Primitif

Tipe data primitif adalah tipe data dasar yang sudah built-in dalam Java.

Tipe Data	Ukuran	Rentang Nilai	Contoh
byte	8 bit	-128 hingga 127	byte umur = 25;
short	16 bit	-32,768 hingga 32,767	short tahun = 2024;
int	32 bit	-2^31 hingga 2^31-1	int jumlah = 1000;
long	64 bit	-2^63 hingga 2^63-1	long populasi = 7800000000L;
float	32 bit	~1.4E-45 hingga 3.4E+38	float nilai = 3.14f;
double	64 bit	~4.9E-324 hingga 1.8E+308	double ipk = 3.75;
char	16 bit	Unicode: 0 hingga 65,535	char huruf = 'A';
boolean	1 bit	true atau false	boolean aktif = true;

B. Tipe Data Reference (Non-Primitif)

Tipe data reference merujuk ke objek dan dapat memiliki method.

Contoh tipe data reference:

- • String - untuk menyimpan teks
- • Array - untuk menyimpan kumpulan data
- • Class - objek yang dibuat dari class
- • Interface - kontrak yang harus diimplementasikan

Contoh Penggunaan:

```
public class ContohTipeData {

    public static void main(String[] args) {

        // Tipe data primitif

        int angka = 100;

        double desimal = 3.14;

        char huruf = 'A';
```

```
        boolean benar = true;


        // Tipe data reference
        String nama = "John Doe";
        int[] nilai = {80, 90, 75, 85};


        // Menampilkan
        System.out.println("Angka: " + angka);
        System.out.println("Nama: " + nama);
        System.out.println("Nilai pertama: " + nilai[0]);
    }
}
```

2.2 Control Program - Percabangan

A. If Statement

Digunakan untuk melakukan pengujian kondisi.

```
int nilai = 75;


if (nilai >= 80) {
    System.out.println("Grade: A");
} else if (nilai >= 70) {
    System.out.println("Grade: B");
} else if (nilai >= 60) {
    System.out.println("Grade: C");
} else {
    System.out.println("Grade: D");
}
```

B. Switch Statement

Digunakan untuk pemilihan berdasarkan nilai tertentu.

```
int hari = 3;
String namaHari;


switch (hari) {
    case 1:
        namaHari = "Senin";
        break;
    case 2:
        namaHari = "Selasa";
        break;
    case 3:
        namaHari = "Rabu";
        break;
    case 4:
        namaHari = "Kamis";
        break;
    case 5:
        namaHari = "Jumat";
        break;
    default:
        namaHari = "Akhir pekan";
}


System.out.println("Hari: " + namaHari);
```

2.3 Control Program - Perulangan

A. For Loop

Digunakan ketika jumlah iterasi sudah diketahui.

```
// Menampilkan angka 1-5
for (int i = 1; i <= 5; i++) {
    System.out.println("Angka: " + i);
}

// Iterasi array
String[] nama = {"Ali", "Budi", "Citra"};
for (int i = 0; i < nama.length; i++) {
    System.out.println(nama[i]);
}
```

B. While Loop

Digunakan ketika jumlah iterasi belum diketahui, kondisi dicek sebelum loop.

```
int counter = 1;

while (counter <= 5) {
    System.out.println("Counter: " + counter);
    counter++;
}
```

C. Do-While Loop

Mirip while, tapi kondisi dicek setelah loop (minimal sekali eksekusi).

```
int angka = 1;

do {
    System.out.println("Angka: " + angka);
    angka++;
```

```
} while (angka <= 5);
```

D. Enhanced For Loop (For-Each)

Digunakan untuk iterasi collection atau array.

```
String[] buah = {"Apel", "Mangga", "Jeruk"};


for (String item : buah) {
    System.out.println("Buah: " + item);
}
```

2.4 Operator dalam Java

A. Operator Aritmatika

```
int a = 10, b = 3;


System.out.println(a + b);  // 13 (Penjumlahan)
System.out.println(a - b);  // 7  (Pengurangan)
System.out.println(a * b);  // 30 (Perkalian)
System.out.println(a / b);  // 3  (Pembagian)
System.out.println(a % b);  // 1  (Modulus/Sisa bagi)
```

B. Operator Perbandingan

```
int x = 5, y = 10;


System.out.println(x == y);  // false (sama dengan)
System.out.println(x != y);  // true  (tidak sama dengan)
System.out.println(x > y);   // false (lebih besar)
System.out.println(x < y);   // true  (lebih kecil)
System.out.println(x >= y);  // false (lebih besar sama dengan)
System.out.println(x <= y);  // true  (lebih kecil sama dengan)
```

C. Operator Logika

```
boolean p = true, q = false;


System.out.println(p && q);  // false (AND)
System.out.println(p || q);  // true  (OR)
System.out.println(!p);      // false (NOT)
```

MEMBUAT PROGRAM DENGAN KONSEP BERBASIS OBJEK

3.1 Inheritance (Pewarisan)

Inheritance adalah kemampuan sebuah class untuk mewarisi properti dan method dari class lain. Class yang mewariskan disebut superclass/parent class, dan class yang mewarisi disebut subclass/child class.

Keuntungan Inheritance:

- • Reusability - kode dapat digunakan kembali
- • Extensibility - mudah untuk memperluas fungsi
- • Maintainability - mudah untuk maintenance

Contoh Implementasi:

```
// Superclass (Parent)
public class Kendaraan {
    protected String merk;
    protected int kecepatan;

    public Kendaraan(String merk) {
        this.merk = merk;
        this.kecepatan = 0;
    }

    public void jalan() {
        System.out.println(merk + " sedang berjalan");
    }

    public void tambahKecepatan(int tambahan) {
        kecepatan += tambahan;
        System.out.println("Kecepatan: " + kecepatan + " km/jam");
    }
```

```
}

// Subclass (Child)
public class Mobil extends Kendaraan {
    private int jumlahPintu;

    public Mobil(String merk, int jumlahPintu) {
        super(merk);  // Memanggil constructor parent
        this.jumlahPintu = jumlahPintu;
    }

    public void bukaPintu() {
        System.out.println("Membuka " + jumlahPintu + " pintu");
    }
}

// Subclass lain
public class Motor extends Kendaraan {
    private String jenisMotor;

    public Motor(String merk, String jenis) {
        super(merk);
        this.jenisMotor = jenis;
    }

    public void wheelie() {
        System.out.println(jenisMotor + " melakukan wheelie!");
    }
```

```
}
```

Penggunaan:

```
public class Main {
    public static void main(String[] args) {
        Mobil mobil1 = new Mobil("Toyota", 4);
        mobil1.jalan();              // Method dari parent
        mobil1.tambahKecepatan(60);  // Method dari parent
        mobil1.bukaPintu();          // Method dari child

        Motor motor1 = new Motor("Honda", "Sport");
        motor1.jalan();              // Method dari parent
        motor1.wheelie();            // Method dari child
    }
}
```

3.2 Polymorphism (Polimorfisme)

Polymorphism adalah kemampuan objek untuk memiliki banyak bentuk. Dalam Java, polymorphism dapat dicapai melalui method overriding dan method overloading.

A. Method Overriding

Method overriding adalah teknik dimana subclass memberikan implementasi spesifik dari method yang sudah didefinisikan di superclass.

```
// Superclass
public class Hewan {
    public void bersuara() {
        System.out.println("Hewan bersuara");
    }

    public void makan() {
```

```
        System.out.println("Hewan sedang makan");
    }
}

// Subclass - Override method bersuara()
public class Kucing extends Hewan {
    @Override
    public void bersuara() {
        System.out.println("Meong meong!");
    }
}

public class Anjing extends Hewan {
    @Override
    public void bersuara() {
        System.out.println("Guk guk!");
    }
}

public class Sapi extends Hewan {
    @Override
    public void bersuara() {
        System.out.println("Moo moo!");
    }
}
```

Contoh Polymorphism:

```
public class Main {
```

```
    public static void main(String[] args) {
        // Polymorphism - tipe parent, objek child
        Hewan hewan1 = new Kucing();
        Hewan hewan2 = new Anjing();
        Hewan hewan3 = new Sapi();

        // Memanggil method yang sama, hasil berbeda
        hewan1.bersuara();  // Output: Meong meong!
        hewan2.bersuara();  // Output: Guk guk!
        hewan3.bersuara();  // Output: Moo moo!

        // Method yang tidak di-override tetap sama
        hewan1.makan();     // Output: Hewan sedang makan
    }
}
```

Aturan Method Overriding:

- • Method harus memiliki nama yang sama dengan method di parent
- • Parameter harus sama persis
- • Return type harus sama atau covariant
- • Access modifier tidak boleh lebih restriktif
- • Gunakan anotasi @Override (opsional tapi direkomendasikan)

3.3 Overloading

Method overloading adalah teknik dimana sebuah class memiliki lebih dari satu method dengan nama yang sama tetapi parameter berbeda (jumlah, tipe, atau urutan).

Contoh Method Overloading:

```
public class Kalkulator {

    // Overloading dengan jumlah parameter berbeda
```

```
    public int tambah(int a, int b) {
        return a + b;
    }

    public int tambah(int a, int b, int c) {
        return a + b + c;
    }

    // Overloading dengan tipe parameter berbeda
    public double tambah(double a, double b) {
        return a + b;
    }

    public String tambah(String a, String b) {
        return a + b;  // Konkatenasi string
    }
}
```

Penggunaan:

```
public class Main {
    public static void main(String[] args) {
        Kalkulator kalkulator = new Kalkulator();

        System.out.println(kalkulator.tambah(5, 3));        // 8
        System.out.println(kalkulator.tambah(5, 3, 2));     // 10
        System.out.println(kalkulator.tambah(5.5, 3.2));    // 8.7
        System.out.println(kalkulator.tambah("Hello", " World"));  // Hello World
    }
```

```
}
```

Constructor Overloading:

Constructor juga dapat di-overload.

```
public class Mahasiswa {
    private String nama;
    private String nim;
    private double ipk;

    // Constructor tanpa parameter
    public Mahasiswa() {
        this.nama = "Belum diisi";
        this.nim = "00000000";
        this.ipk = 0.0;
    }

    // Constructor dengan 2 parameter
    public Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
        this.ipk = 0.0;
    }

    // Constructor dengan 3 parameter
    public Mahasiswa(String nama, String nim, double ipk) {
        this.nama = nama;
        this.nim = nim;
```

```
        this.ipk = ipk;
    }

    public void tampilkanInfo() {
        System.out.println("Nama: " + nama);
        System.out.println("NIM: " + nim);
        System.out.println("IPK: " + ipk);
    }
}
```

Perbedaan Overriding vs Overloading:

Aspek	Overriding	Overloading
Definisi	Mendefinisikan ulang method parent	Method dengan nama sama, parameter beda
Lokasi	Di subclass	Di class yang sama
Parameter	Harus sama	Harus berbeda
Waktu	Runtime polymorphism	Compile-time polymorphism

MEMBUAT PROGRAM OBJECT ORIENTED DENGAN INTERFACE DAN PAKET

4.1 Interface

Interface adalah kontrak atau blueprint yang berisi deklarasi method (tanpa implementasi). Class yang mengimplementasikan interface harus menyediakan implementasi untuk semua method di interface tersebut.

Karakteristik Interface:

- • Semua method dalam interface bersifat abstract (tanpa body)
- • Semua method otomatis public
- • Dapat memiliki konstanta (public static final)
- • Class dapat mengimplementasi multiple interface
- • Sejak Java 8, dapat memiliki default dan static method

Contoh Interface:

```
// Interface untuk perilaku hewan
public interface Bersuara {
    void bersuara();  // Method abstract
    void info();      // Method abstract
}

// Interface untuk perilaku bergerak
public interface Bergerak {
    void berjalan();
    void berlari();
}

// Class yang mengimplementasi satu interface
public class Kucing implements Bersuara {
    private String nama;
```

```
    public Kucing(String nama) {
        this.nama = nama;
    }

    @Override
    public void bersuara() {
        System.out.println(nama + " bersuara: Meong meong!");
    }

    @Override
    public void info() {
        System.out.println("Ini adalah kucing bernama " + nama);
    }
}

// Class yang mengimplementasi multiple interface
public class Anjing implements Bersuara, Bergerak {
    private String nama;

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

    @Override
    public void bersuara() {
        System.out.println(nama + " bersuara: Guk guk!");
    }
```

```
    @Override
    public void info() {
        System.out.println("Ini adalah anjing bernama " + nama);
    }

    @Override
    public void berjalan() {
        System.out.println(nama + " sedang berjalan");
    }

    @Override
    public void berlari() {
        System.out.println(nama + " sedang berlari");
    }
}
```

Penggunaan Interface:

```
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Kitty");
        kucing.bersuara();
        kucing.info();

        Anjing anjing = new Anjing("Bobby");
        anjing.bersuara();
        anjing.berjalan();
        anjing.berlari();
```

```
        // Polymorphism dengan interface
        Bersuara hewan1 = new Kucing("Mimi");
        Bersuara hewan2 = new Anjing("Rocky");

        hewan1.bersuara();  // Meong meong!
        hewan2.bersuara();  // Guk guk!
    }
}
```

4.2 Package (Paket)

Package adalah cara untuk mengorganisir class dan interface ke dalam namespace atau grup. Package membantu menghindari konflik nama dan membuat kode lebih terstruktur.

Keuntungan Menggunakan Package:

- • Organisasi - mengelompokkan class yang berkaitan
- • Menghindari konflik nama
- • Access control - mengatur visibility
- • Reusability - mudah digunakan kembali

Struktur Package:

```
// File: com/klinik/model/Pasien.java
package com.klinik.model;

public class Pasien {
    private String idPasien;
    private String nama;
    private int umur;

    public Pasien(String id, String nama, int umur) {
        this.idPasien = id;
```

```
        this.nama = nama;
        this.umur = umur;
    }

    // Getter dan setter
    public String getIdPasien() { return idPasien; }
    public String getNama() { return nama; }
    public int getUmur() { return umur; }
}

// File: com/klinik/model/Dokter.java
package com.klinik.model;

public class Dokter {
    private String idDokter;
    private String nama;
    private String spesialisasi;

    public Dokter(String id, String nama, String spesialisasi) {
        this.idDokter = id;
        this.nama = nama;
        this.spesialisasi = spesialisasi;
    }

    public String getIdDokter() { return idDokter; }
    public String getNama() { return nama; }
    public String getSpesialisasi() { return spesialisasi; }
}
```

Menggunakan Class dari Package:

```
// File: com/klinik/service/AppointmentService.java
package com.klinik.service;

// Import class dari package lain
import com.klinik.model.Pasien;
import com.klinik.model.Dokter;

public class AppointmentService {

    public void buatJanji(Pasien pasien, Dokter dokter, String tanggal) {
        System.out.println("Janji dibuat:");
        System.out.println("Pasien: " + pasien.getNama());
        System.out.println("Dokter: " + dokter.getNama() +
                           " (" + dokter.getSpesialisasi() + ")");
        System.out.println("Tanggal: " + tanggal);
    }
}
```

Cara Import Package:

```
// Import class tertentu
import com.klinik.model.Pasien;

// Import semua class dari package
import com.klinik.model.*;
```

```
// Import package Java built-in

import java.util.ArrayList;

import java.util.Scanner;

import java.time.LocalDate;
```

MENGKOMPILASI PROGRAM

5.1 Kompilasi Program Java

Kompilasi adalah proses mengubah kode sumber (source code) menjadi bytecode yang dapat dijalankan oleh Java Virtual Machine (JVM).

Langkah Kompilasi:

1. 1. Tulis kode Java dengan ekstensi .java
2. 2. Kompilasi dengan command javac
3. 3. Jalankan dengan command java

Contoh Kompilasi:

```
// File: HelloWorld.java

public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello, World!");

    }

}


// Kompilasi di command line:

// javac HelloWorld.java

//

// Hasil: HelloWorld.class (bytecode)

//

// Jalankan:

// java HelloWorld

//

// Output: Hello, World!
```

5.2 Jenis-jenis Error

A. Syntax Error (Compile-time Error)

Error yang terjadi saat kompilasi karena pelanggaran aturan sintaks Java.

```
// ERROR: Lupa titik koma
int angka = 10

// ERROR: Salah penulisan keyword
publik class Test {  // Seharusnya 'public'
}

// ERROR: Tanda kurung tidak seimbang
if (x > 5 {  // Kurang tanda kurung tutup
}

// Pesan error contoh:
// HelloWorld.java:5: error: ';' expected
//     int angka = 10
//         ^
```

B. Runtime Error

Error yang terjadi saat program berjalan.

```
// Runtime Error: Division by zero
public class RuntimeErrorExample {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;
        int hasil = a / b;  // ArithmeticException
    }
}
```

```
// Runtime Error: Array index out of bounds
public class ArrayError {
    public static void main(String[] args) {
        int[] angka = {1, 2, 3};
        System.out.println(angka[5]);  // ArrayIndexOutOfBoundsException
    }
}
```

C. Logical Error

Program berjalan tanpa error tapi hasil tidak sesuai harapan.

```
// Logical Error: Logika salah
public class LogicalError {
    public static void main(String[] args) {
        int nilai = 75;

        // Logic error: seharusnya >= 70
        if (nilai > 70) {  // Nilai 70 tidak lulus
            System.out.println("Lulus");
        } else {
            System.out.println("Tidak Lulus");
        }
    }
}
```

5.3 Debugging dan Troubleshooting

Teknik Debugging:

1. Print Debugging

```
public int hitungTotal(int[] nilai) {
    int total = 0;
```

```
        for (int i = 0; i < nilai.length; i++) {
            total += nilai[i];
            System.out.println("Debug: i=" + i + ", nilai[i]=" + nilai[i] +
                               ", total=" + total);  // Debug print
        }
        return total;
    }
```

2. Try-Catch untuk Error Handling

```
    public class ErrorHandling {
        public static void main(String[] args) {
            try {
                int[] angka = {1, 2, 3};
                System.out.println(angka[5]);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Error: Index array melebihi batas!");
                System.out.println("Detail: " + e.getMessage());
            }
        }
    }
```

3. Gunakan IDE Debugger

IDE seperti IntelliJ IDEA, Eclipse, atau NetBeans memiliki debugger untuk:

- • Set breakpoint
- • Step through code
- • Watch variables
- • Inspect call stack

Tips Menghindari Error:

- • Gunakan IDE dengan syntax highlighting dan code completion
- • Ikuti naming convention Java

- • Gunakan indentasi yang konsisten
- • Beri komentar pada kode yang kompleks
- • Test code secara bertahap
- • Gunakan version control (Git)

PROJECT PRAKTIKUM

SISTEM MANAJEMEN KLINIK KESEHATAN

Project ini mengimplementasikan seluruh konsep OOP yang telah dipelajari dalam aplikasi nyata Sistem Manajemen Klinik Kesehatan.

Deskripsi Project

Sistem Klinik Kesehatan yang dapat:

- • Mengelola data pasien dan rekam medis
- • Mengelola data tenaga medis (dokter, perawat)
- • Membuat dan mengelola appointment/janji temu
- • Melakukan pemeriksaan dan pencatatan diagnosa
- • Mengelola resep obat

Diagram Class

Struktur hirarki class dalam sistem:

Interface:

- Identifiable (getId(), tampilkanInfo())

- Treatable (periksaPasien(), buatResep())

Abstract Class:

- TenagaMedis (parent untuk Dokter & Perawat)

Concrete Class:

- Pasien implements Identifiable

- Dokter extends TenagaMedis implements Treatable, Identifiable

- Perawat extends TenagaMedis implements Identifiable

- Appointment

- RekamMedis

- Resep

Struktur Package

klinik/

```
├── model/
│   ├── Pasien.java
│   ├── TenagaMedis.java (abstract)
│   ├── Dokter.java
│   ├── Perawat.java
│   ├── Appointment.java
│   ├── RekamMedis.java
│   └── Resep.java
├── interfaces/
│   ├── Identifiable.java
│   └── Treatable.java
├── service/
│   ├── KlinikService.java
│   └── AppointmentService.java
└── Main.java
```

Implementasi Code

1. Interface Identifiable

```
package klinik.interfaces;


public interface Identifiable {
    String getId();
    void tampilkanInfo();
}
```

2. Interface Treatable

```
package klinik.interfaces;
import klinik.model.*;
```

```
public interface Treatable {
    void periksaPasien(Pasien pasien, String keluhan);
    Resep buatResep(Pasien pasien, String namaObat, String dosis);
}
```

3. Class Pasien

```
package klinik.model;
import klinik.interfaces.Identifiable;
import java.util.ArrayList;

public class Pasien implements Identifiable {
    private String idPasien;
    private String nama;
    private int umur;
    private String alamat;
    private String nomorTelepon;
    private ArrayList<RekamMedis> riwayatMedis;

    public Pasien(String id, String nama, int umur, String alamat, String telp) {
        this.idPasien = id;
        this.nama = nama;
        this.umur = umur;
        this.alamat = alamat;
        this.nomorTelepon = telp;
        this.riwayatMedis = new ArrayList<>();
    }

    @Override
```

```
public String getId() {
    return idPasien;
}

@Override
public void tampilkanInfo() {
    System.out.println("=== INFORMASI PASIEN ===");
    System.out.println("ID Pasien: " + idPasien);
    System.out.println("Nama: " + nama);
    System.out.println("Umur: " + umur + " tahun");
    System.out.println("Alamat: " + alamat);
    System.out.println("Telepon: " + nomorTelepon);
}

public void tambahRekamMedis(RekamMedis rekam) {
    riwayatMedis.add(rekam);
}

public void tampilkanRiwayatMedis() {
    System.out.println("\n=== RIWAYAT MEDIS ===");
    if (riwayatMedis.isEmpty()) {
        System.out.println("Belum ada riwayat medis");
    } else {
        for (RekamMedis rekam : riwayatMedis) {
            rekam.tampilkanInfo();
            System.out.println("---");
        }
    }
```

```
    }

    // Getters
    public String getNama() { return nama; }
    public int getUmur() { return umur; }
    public String getAlamat() { return alamat; }
    public String getNomorTelepon() { return nomorTelepon; }
}
```

4. Abstract Class TenagaMedis

```
package klinik.model;
import klinik.interfaces.Identifiable;

// Abstract class sebagai parent untuk Dokter dan Perawat
public abstract class TenagaMedis implements Identifiable {
    protected String id;
    protected String nama;
    protected String nomorLisensi;
    protected int tahunPengalaman;

    public TenagaMedis(String id, String nama, String lisensi, int pengalaman) {
        this.id = id;
        this.nama = nama;
        this.nomorLisensi = lisensi;
        this.tahunPengalaman = pengalaman;
    }

    @Override
```

```
    public String getId() {
        return id;
    }

    // Abstract method - harus diimplementasi oleh subclass
    public abstract String getTipe();

    // Method biasa yang bisa di-override
    public void bertugas() {
        System.out.println(nama + " sedang bertugas");
    }

    // Getters
    public String getNama() { return nama; }
    public String getNomorLisensi() { return nomorLisensi; }
    public int getTahunPengalaman() { return tahunPengalaman; }
}
```

5. Class Dokter (Inheritance + Multiple Interface)

```
package klinik.model;
import klinik.interfaces.Treatable;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Dokter extends TenagaMedis implements Treatable {
    private String spesialisasi;
    private String jadwalPraktek;
```

```
public Dokter(String id, String nama, String lisensi, int pengalaman,
              String spesialisasi, String jadwal) {
    super(id, nama, lisensi, pengalaman);
    this.spesialisasi = spesialisasi;
    this.jadwalPraktek = jadwal;
}

@Override
public String getTipe() {
    return "Dokter";
}

@Override
public void tampilkanInfo() {
    System.out.println("=== INFORMASI DOKTER ===");
    System.out.println("ID: " + id);
    System.out.println("Nama: Dr. " + nama);
    System.out.println("Spesialisasi: " + spesialisasi);
    System.out.println("Lisensi: " + nomorLisensi);
    System.out.println("Pengalaman: " + tahunPengalaman + " tahun");
    System.out.println("Jadwal Praktek: " + jadwalPraktek);
}

@Override
public void periksaPasien(Pasien pasien, String keluhan) {
    System.out.println("\n=== PEMERIKSAAN PASIEN ===");
    System.out.println("Dokter: Dr. " + nama + " (" + spesialisasi + ")");
    System.out.println("Pasien: " + pasien.getNama());
```

```
        System.out.println("Keluhan: " + keluhan);
        System.out.println("Sedang melakukan pemeriksaan...");
    }

    @Override
    public Resep buatResep(Pasien pasien, String namaObat, String dosis) {
        Resep resep = new Resep(this, pasien, namaObat, dosis);
        System.out.println("Resep berhasil dibuat untuk " + pasien.getNama());
        return resep;
    }

    // Method overriding
    @Override
    public void bertugas() {
        System.out.println("Dr. " + nama + " sedang memeriksa pasien");
    }

    // Getters
    public String getSpesialisasi() { return spesialisasi; }
    public String getJadwalPraktek() { return jadwalPraktek; }
}
```

6. Class Perawat

```
package klinik.model;

public class Perawat extends TenagaMedis {
    private String shift;  // Pagi, Siang, Malam
    private String ruangan;
```

```
public Perawat(String id, String nama, String lisensi, int pengalaman,
               String shift, String ruangan) {
    super(id, nama, lisensi, pengalaman);
    this.shift = shift;
    this.ruangan = ruangan;
}

@Override
public String getTipe() {
    return "Perawat";
}

@Override
public void tampilkanInfo() {
    System.out.println("=== INFORMASI PERAWAT ===");
    System.out.println("ID: " + id);
    System.out.println("Nama: " + nama);
    System.out.println("Lisensi: " + nomorLisensi);
    System.out.println("Pengalaman: " + tahunPengalaman + " tahun");
    System.out.println("Shift: " + shift);
    System.out.println("Ruangan: " + ruangan);
}

@Override
public void bertugas() {
    System.out.println(nama + " sedang merawat pasien di " + ruangan);
}
```

```
    public void cekVitalSign(Pasien pasien) {
        System.out.println("\nPerawat " + nama + " mengecek vital sign:");
        System.out.println("Pasien: " + pasien.getNama());
        System.out.println("Tekanan darah, suhu, nadi - Normal");
    }

    // Getters
    public String getShift() { return shift; }
    public String getRuangan() { return ruangan; }
}
```

7. Class Appointment

```
package klinik.model;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Appointment {
    private static int counter = 1;
    private String idAppointment;
    private Pasien pasien;
    private Dokter dokter;
    private LocalDateTime waktuJanji;
    private String keluhan;
    private String status;  // Terjadwal, Selesai, Dibatalkan

    public Appointment(Pasien pasien, Dokter dokter,
                       LocalDateTime waktu, String keluhan) {
```

```
        this.idAppointment = "APT" + String.format("%05d", counter++);
        this.pasien = pasien;
        this.dokter = dokter;
        this.waktuJanji = waktu;
        this.keluhan = keluhan;
        this.status = "Terjadwal";
    }

    public void tampilkanInfo() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern(
            "dd-MM-yyyy HH:mm");
        System.out.println("=== APPOINTMENT ===");
        System.out.println("ID: " + idAppointment);
        System.out.println("Pasien: " + pasien.getNama());
        System.out.println("Dokter: Dr. " + dokter.getNama() +
                           " (" + dokter.getSpesialisasi() + ")");
        System.out.println("Waktu: " + waktuJanji.format(formatter));
        System.out.println("Keluhan: " + keluhan);
        System.out.println("Status: " + status);
    }

    public void selesaikan() {
        this.status = "Selesai";
        System.out.println("Appointment selesai");
    }

    public void batalkan() {
        this.status = "Dibatalkan";
```

```
            System.out.println("Appointment dibatalkan");
        }

        // Getters
        public String getIdAppointment() { return idAppointment; }
        public Pasien getPasien() { return pasien; }
        public Dokter getDokter() { return dokter; }
        public String getStatus() { return status; }
    }
```

8. Class RekamMedis

```
    package klinik.model;
    import java.time.LocalDate;
    import java.time.format.DateTimeFormatter;

    public class RekamMedis {
        private static int counter = 1;
        private String idRekam;
        private LocalDate tanggal;
        private String keluhan;
        private String diagnosa;
        private String tindakan;
        private Dokter dokter;

        public RekamMedis(Dokter dokter, String keluhan,
                          String diagnosa, String tindakan) {
            this.idRekam = "RM" + String.format("%05d", counter++);
            this.tanggal = LocalDate.now();
```

```
        this.dokter = dokter;
        this.keluhan = keluhan;
        this.diagnosa = diagnosa;
        this.tindakan = tindakan;
    }

    public void tampilkanInfo() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
        System.out.println("ID Rekam: " + idRekam);
        System.out.println("Tanggal: " + tanggal.format(formatter));
        System.out.println("Dokter: Dr. " + dokter.getNama());
        System.out.println("Keluhan: " + keluhan);
        System.out.println("Diagnosa: " + diagnosa);
        System.out.println("Tindakan: " + tindakan);
    }

    // Getters
    public String getIdRekam() { return idRekam; }
    public LocalDate getTanggal() { return tanggal; }
    public String getDiagnosa() { return diagnosa; }
}
```

9. Class Resep

```
package klinik.model;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Resep {
```

```
private static int counter = 1;
private String idResep;
private Dokter dokter;
private Pasien pasien;
private String namaObat;
private String dosis;
private String aturanPakai;
private LocalDate tanggal;

public Resep(Dokter dokter, Pasien pasien,
             String namaObat, String dosis) {
    this.idResep = "RSP" + String.format("%05d", counter++);
    this.dokter = dokter;
    this.pasien = pasien;
    this.namaObat = namaObat;
    this.dosis = dosis;
    this.tanggal = LocalDate.now();
}

public void setAturanPakai(String aturan) {
    this.aturanPakai = aturan;
}

public void tampilkanInfo() {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
    System.out.println("\n=== RESEP OBAT ===");
    System.out.println("ID Resep: " + idResep);
    System.out.println("Tanggal: " + tanggal.format(formatter));
```

```
        System.out.println("Dokter: Dr. " + dokter.getNama());
        System.out.println("Pasien: " + pasien.getNama());
        System.out.println("Obat: " + namaObat);
        System.out.println("Dosis: " + dosis);
        if (aturanPakai != null) {
            System.out.println("Aturan Pakai: " + aturanPakai);
        }
    }

    // Getters
    public String getIdResep() { return idResep; }
    public String getNamaObat() { return namaObat; }
}
```

10. Class KlinikService (Service Layer)

```
package klinik.service;

import klinik.model.*;

import java.util.ArrayList;

import java.time.LocalDateTime;


public class KlinikService {

    private ArrayList<Pasien> daftarPasien;

    private ArrayList<Dokter> daftarDokter;

    private ArrayList<Perawat> daftarPerawat;

    private ArrayList<Appointment> daftarAppointment;


    public KlinikService() {

        this.daftarPasien = new ArrayList<>();

        this.daftarDokter = new ArrayList<>();

        this.daftarPerawat = new ArrayList<>();

        this.daftarAppointment = new ArrayList<>();

    }


    // === MANAJEMEN PASIEN ===

    public void daftarPasienBaru(Pasien pasien) {

        daftarPasien.add(pasien);

        System.out.println("Pasien berhasil didaftarkan: " + pasien.getNama());

    }


    public Pasien cariPasien(String idPasien) {

        for (Pasien pasien : daftarPasien) {

            if (pasien.getId().equals(idPasien)) {
```

```
            return pasien;
        }
    }
    return null;
}

// === MANAJEMEN TENAGA MEDIS ===
public void tambahDokter(Dokter dokter) {
    daftarDokter.add(dokter);
    System.out.println("Dokter ditambahkan: Dr. " + dokter.getNama());
}

public void tambahPerawat(Perawat perawat) {
    daftarPerawat.add(perawat);
    System.out.println("Perawat ditambahkan: " + perawat.getNama());
}

public Dokter cariDokter(String idDokter) {
    for (Dokter dokter : daftarDokter) {
        if (dokter.getId().equals(idDokter)) {
            return dokter;
        }
    }
    return null;
}

public void tampilkanDaftarDokter() {
    System.out.println("\n=== DAFTAR DOKTER ===");
```

```
        for (Dokter dokter : daftarDokter) {
            dokter.tampilkanInfo();
            System.out.println("---");
        }
    }

    // === MANAJEMEN APPOINTMENT ===
    public Appointment buatAppointment(String idPasien, String idDokter,
                                       LocalDateTime waktu, String keluhan) {
        Pasien pasien = cariPasien(idPasien);
        Dokter dokter = cariDokter(idDokter);

        if (pasien == null) {
            System.out.println("Pasien tidak ditemukan!");
            return null;
        }

        if (dokter == null) {
            System.out.println("Dokter tidak ditemukan!");
            return null;
        }

        Appointment appointment = new Appointment(pasien, dokter, waktu,
keluhan);
        daftarAppointment.add(appointment);
        System.out.println("Appointment berhasil dibuat!");
        appointment.tampilkanInfo();
        return appointment;
```

```
    }

    // === PROSES PEMERIKSAAN ===
    public void prosesPemeriksaan(Appointment appointment,
                                  String diagnosa, String tindakan) {
        Dokter dokter = appointment.getDokter();
        Pasien pasien = appointment.getPasien();

        // Polymorphism - memanggil method interface
        dokter.periksaPasien(pasien, "Pemeriksaan rutin");

        // Buat rekam medis
        RekamMedis rekam = new RekamMedis(dokter, "Pemeriksaan",
                                          diagnosa, tindakan);
        pasien.tambahRekamMedis(rekam);

        appointment.selesaikan();
        System.out.println("Pemeriksaan selesai dan rekam medis tersimpan");
    }
}
```

11. Main Program - Demonstrasi Sistem

```
package klinik;
import klinik.model.*;
import klinik.service.KlinikService;
import java.time.LocalDateTime;
import java.util.Scanner;
```

```
public class Main {
    public static void main(String[] args) {
        KlinikService klinik = new KlinikService();
        Scanner scanner = new Scanner(System.in);

        // Inisialisasi data dummy
        initData(klinik);

        while (true) {
            System.out.println("\n=== SISTEM KLINIK KESEHATAN ===");
            System.out.println("1. Daftar Pasien Baru");
            System.out.println("2. Buat Appointment");
            System.out.println("3. Lihat Daftar Dokter");
            System.out.println("4. Demonstrasi OOP");
            System.out.println("5. Keluar");
            System.out.print("Pilih menu: ");

            int pilihan = scanner.nextInt();
            scanner.nextLine();

            switch (pilihan) {
                case 1:
                    daftarPasienBaru(scanner, klinik);
                    break;
                case 2:
                    buatAppointment(scanner, klinik);
                    break;
                case 3:
```

```
                klinik.tampilkanDaftarDokter();
                break;
            case 4:
                demonstrasiOOP(klinik);
                break;
            case 5:
                System.out.println("Terima kasih!");
                return;
        }
    }
}

private static void initData(KlinikService klinik) {
    // Tambah dokter
    Dokter dokter1 = new Dokter("D001", "Ahmad Santoso", "SIP-001",
        10, "Umum", "Senin-Jumat 08:00-16:00");
    Dokter dokter2 = new Dokter("D002", "Siti Nurhaliza", "SIP-002",
        8, "Anak", "Senin-Sabtu 09:00-17:00");

    klinik.tambahDokter(dokter1);
    klinik.tambahDokter(dokter2);

    // Tambah perawat
    Perawat perawat1 = new Perawat("P001", "Rina", "STR-001",
        5, "Pagi", "Ruang Rawat Inap");
    klinik.tambahPerawat(perawat1);

    // Tambah pasien dummy
```

```
    Pasien pasien1 = new Pasien("PS001", "Budi Santoso",
        35, "Jl. Merdeka No. 10", "08123456789");
    klinik.daftarPasienBaru(pasien1);
}

private static void daftarPasienBaru(Scanner scanner,
                                     KlinikService klinik) {
    System.out.println("\n=== PENDAFTARAN PASIEN ===");
    System.out.print("Nama: ");
    String nama = scanner.nextLine();
    System.out.print("Umur: ");
    int umur = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Alamat: ");
    String alamat = scanner.nextLine();
    System.out.print("Telepon: ");
    String telp = scanner.nextLine();

    String id = "PS" + String.format("%03d",
        (int)(Math.random() * 1000));
    Pasien pasien = new Pasien(id, nama, umur, alamat, telp);
    klinik.daftarPasienBaru(pasien);
    pasien.tampilkanInfo();
}

private static void buatAppointment(Scanner scanner,
                                    KlinikService klinik) {
    System.out.print("ID Pasien: ");
```

```
        String idPasien = scanner.nextLine();
        System.out.print("ID Dokter: ");
        String idDokter = scanner.nextLine();
        System.out.print("Keluhan: ");
        String keluhan = scanner.nextLine();

        LocalDateTime waktu = LocalDateTime.now().plusDays(1);
        klinik.buatAppointment(idPasien, idDokter, waktu, keluhan);
    }

    private static void demonstrasiOOP(KlinikService klinik) {
        System.out.println("\n=== DEMONSTRASI KONSEP OOP ===");

        // 1. Inheritance
        System.out.println("\n1. INHERITANCE:");
        Dokter dokter = new Dokter("D003", "Dr. Rudi", "SIP-003",
            12, "Bedah", "Senin-Jumat");
        Perawat perawat = new Perawat("P002", "Ani", "STR-002",
            3, "Siang", "IGD");

        System.out.println("Dokter dan Perawat mewarisi TenagaMedis:");
        dokter.bertugas();  // Override method
        perawat.bertugas(); // Override method

        // 2. Polymorphism
        System.out.println("\n2. POLYMORPHISM:");
        TenagaMedis[] tim = {dokter, perawat};
        for (TenagaMedis tenaga : tim) {
```

```
            System.out.println("Tipe: " + tenaga.getTipe());
            tenaga.tampilkanInfo();
            System.out.println("---");
        }

        // 3. Interface
        System.out.println("\n3. INTERFACE (Treatable):");
        Pasien pasien = new Pasien("PS999", "Test", 25,
            "Jakarta", "081234");
        dokter.periksaPasien(pasien, "Demam");
        Resep resep = dokter.buatResep(pasien, "Paracetamol", "500mg");
        resep.setAturanPakai("3x sehari setelah makan");
        resep.tampilkanInfo();

        // 4. Encapsulation
        System.out.println("\n4. ENCAPSULATION:");
        System.out.println("Data pasien diakses via getter/setter:");
        System.out.println("Nama: " + pasien.getNama());
        System.out.println("ID (read-only): " + pasien.getId());
    }
}
```

Konsep OOP yang Diimplementasikan

1. Class dan Object

Semua entitas (Pasien, Dokter, Perawat, dll) dimodelkan sebagai class dengan atribut dan method

2. Enkapsulasi

Atribut di-set private, akses melalui getter/setter dengan validasi

3. Inheritance

• TenagaMedis (abstract parent) → Dokter & Perawat (child)

• Mewarisi atribut dan method, override method bertugas()

4. Polymorphism

• TenagaMedis dapat merujuk ke Dokter atau Perawat

• Method getTipe() dan bertugas() dipanggil berbeda-beda

5. Interface

• Identifiable - implementasi getId() dan tampilkanInfo()

• Treatable - implementasi periksaPasien() dan buatResep()

6. Package

• model - class entitas

• interfaces - interface

• service - business logic

7. Access Modifier

• private untuk data sensitif (id, nomor telepon)

• protected untuk data yang diakses subclass

• public untuk method yang diakses dari luar

REFERENSI

Berikut adalah referensi buku dan sumber pembelajaran yang digunakan dalam penyusunan materi ini:

Buku Referensi:

Adiputra, F. (2022). *Pemrograman berorientasi objek dengan Java*. MNC Publishing. https://books.google.co.id/books/about/Pemrograman_Berorientasi_Objek_Dengan_Ja.html?id=QYueEAAAQBAJ

Java How to Program, Early Objects – Edisi ke-11 (11th edition)
Deitel, P., & Deitel, H. (2018). *Java How to Program, Early Objects* (11th ed.). Pearson.

Head First Java – Edisi ke-2 (2nd edition)
Sierra, K., & Bates, B. (2005). *Head First Java* (2nd ed.). O'Reilly Media.

Core Java Volume I - Fundamentals – Edisi ke-11 (11th edition)
Horstmann, C. S. (2019). *Core Java Volume I - Fundamentals* (11th ed.). Prentice Hall.

Effective Java – Edisi ke-3 (3rd edition)
Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.

Thinking in Java – Edisi ke-4 (4th edition)
Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.

Introduction to Programming in Java – Edisi pertama (tidak mencantumkan edisi, berarti 1st edition)
Sedgewick, R., & Wayne, K. (2011). *Introduction to Programming in Java*. Addison-Wesley.

Java: The Complete Reference – Edisi ke-11 (11th edition)
Schildt, H. (2018). *Java: The Complete Reference* (11th ed.). McGraw-Hill Education.

Sumber Online:

1. Oracle Java Documentation - https://docs.oracle.com/javase/
2. Java Tutorials - https://docs.oracle.com/javase/tutorial/
3. GeeksforGeeks Java - https://www.geeksforgeeks.org/java/
4. JavaPoint - https://www.javatpoint.com/java-tutorial
5. W3Schools Java - https://www.w3schools.com/java/

--- Akhir Materi ---

Semoga materi ini bermanfaat untuk pembelajaran Anda.
Selamat belajar dan terus berlatih!