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.

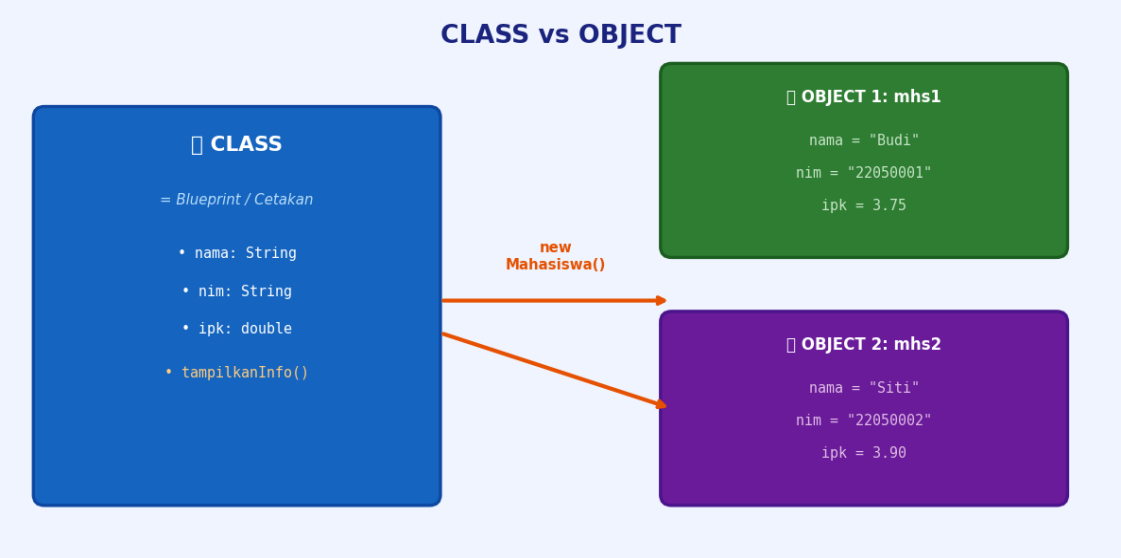
BAB 1: MEMBUAT PROGRAM BERORIENTASI OBJEK DENGAN CLASS

1.1 Pemrograman Berorientasi Objek (OOP)

Pemrograman Berorientasi Objek atau Object Oriented Programming (OOP) adalah pendekatan pemrograman yang memandang program sebagai kumpulan objek yang saling berinteraksi. Setiap objek memiliki data (atribut) dan perilaku (method) sendiri.

> **Analogi Dunia Nyata**
> Bayangkan kamu bermain game RPG. Ada "Karakter" yang punya nama, level, HP (atribut), dan bisa menyerang, bertahan, menggunakan item (method). Itulah konsep OOP - kita memodelkan benda nyata ke dalam kode!

1.2 Pengenalan Class dan Object



Gambar 1.1: Ilustrasi hubungan Class (blueprint) dan Object (hasil nyata)

Perhatikan ilustrasi di atas. Class adalah cetakan, dan Object adalah hasil cetakannya. Dari satu Class Mahasiswa, kita bisa membuat banyak objek mahasiswa yang berbeda-beda.

Penjelasan Konsep

CLASS = Blueprint/Cetakan yang mendefinisikan STRUKTUR dan PERILAKU objek. Seperti cetakan kue - belum jadi kuenya.

OBJECT = Instance/Wujud nyata dari class. Seperti kue yang sudah jadi dari cetakan tersebut.

Contoh Code: Membuat Class Mahasiswa

```
public class Mahasiswa {
```

```
    // ATRIBUT - data yang dimiliki objek
    String nama;
    String nim;
    double ipk;

    // METHOD - perilaku/aksi yang bisa dilakukan objek
    public void tampilkanInfo() {
        System.out.println("Nama: " + nama);
        System.out.println("NIM:  " + nim);
        System.out.println("IPK:  " + ipk);
    }
}
```

Penjelasan Code

• Baris "public class Mahasiswa" = mendefinisikan class baru bernama Mahasiswa • Bagian atribut (nama, nim, ipk) = data yang dimiliki setiap objek mahasiswa • Method tampilkanInfo() = aksi yang bisa dilakukan objek (mencetak info ke layar) • Curly braces { } = menandai awal dan akhir blok class/method

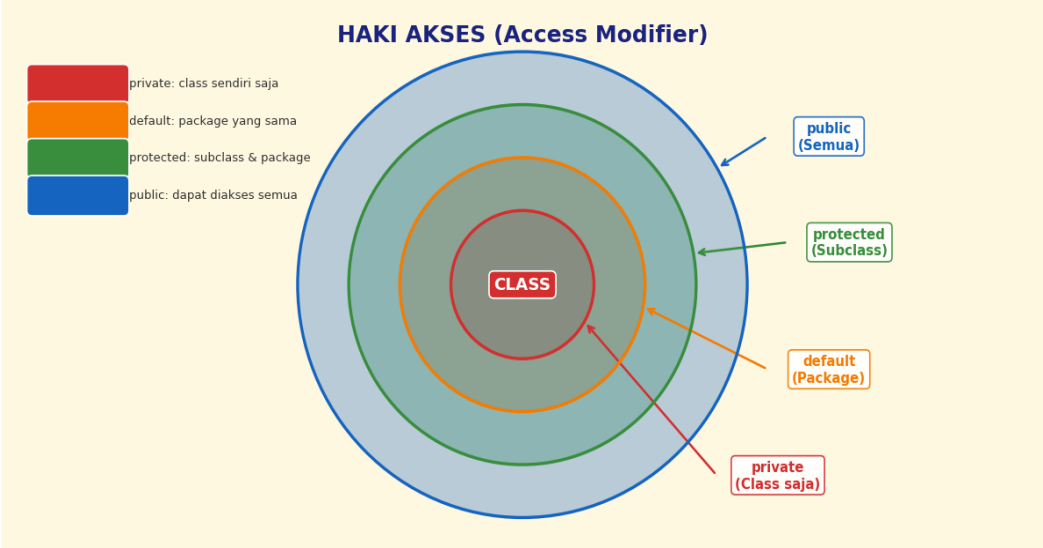
Cara Membuat dan Menggunakan Object

```
public class Main {
    public static void main(String[] args) {
        // Membuat OBJECT dari class Mahasiswa
        Mahasiswa mhs1 = new Mahasiswa();  // "new" = membuat objek baru
        Mahasiswa mhs2 = new Mahasiswa();

        // Mengisi atribut objek
        mhs1.nama = "Budi";
        mhs1.nim  = "22050001";
        mhs1.ipk  = 3.75;

        mhs2.nama = "Siti";
        mhs2.nim  = "22050002";
        mhs2.ipk  = 3.90;

        // Memanggil method objek
        mhs1.tampilkanInfo();  // Output: Nama: Budi, NIM: 22050001, ...
        mhs2.tampilkanInfo();  // Output: Nama: Siti, NIM: 22050002, ...
    }
}
```

Penjelasan Code

• "Mahasiswa mhs1" = mendeklarasikan variabel bertipe Mahasiswa • "= new Mahasiswa()" = membuat objek baru di memori • "mhs1.nama = ..." = mengakses dan mengisi atribut objek mhs1 • Dua objek mhs1 dan mhs2 INDEPENDEN - mengubah mhs1 tidak mempengaruhi mhs2

1.3 Hak Akses (Access Modifier)



Gambar 1.2: Visualisasi tingkat akses - semakin dalam, semakin terbatas

Access Modifier menentukan siapa yang boleh mengakses atribut atau method. Ibarat pintu rumah - ada yang boleh semua orang masuk (public), ada yang hanya keluarga (protected), dan ada yang hanya penghuni (private).

Tabel Access Modifier

Modifier	Class Sendiri	Package Sama	Subclass	Class Lain
public	Ya	Ya	Ya	Ya
protected	Ya	Ya	Ya	Tidak
default	Ya	Ya	Tidak	Tidak
private	Ya	Tidak	Tidak	Tidak

Contoh Penggunaan Access Modifier

```
public class RekeningBank {
    // private: hanya bisa diakses DALAM class ini
    private double saldo;
    private String nomorRekening;

    // public: bisa diakses dari MANA SAJA
    public double getSaldo() {
        return saldo;  // akses saldo via method
    }

    public void setor(double jumlah) {
        if (jumlah > 0) {     // ada validasi!
            saldo += jumlah;
        }
    }
}

// Di Main.java:
```

```
RekeningBank rek = new RekeningBank();
// rek.saldo = 1000;        // ERROR! saldo bersifat private
rek.setor(1000);             // OK! pakai method public
System.out.println(rek.getSaldo());  // OK! getter public
```

Mengapa private itu penting?

Bayangkan kamu punya rekening bank. Apakah wajar jika siapapun bisa langsung mengubah saldo tanpa validasi? Tentu tidak! Dengan private, kita memaksa semua perubahan melalui method yang kita kendalikan, sehingga bisa dipasang validasi (misal: saldo tidak boleh negatif).

1.4 Constructor

Gambar 1.3: Alur kerja Constructor saat objek dibuat

Constructor adalah method khusus yang otomatis dipanggil ketika kita membuat objek baru dengan kata kunci new. Fungsinya untuk menginisialisasi (mengatur nilai awal) atribut objek.

Ciri Khas Constructor

1. Nama HARUS sama persis dengan nama class 2. TIDAK memiliki return type (bahkan void pun tidak) 3. Dipanggil OTOMATIS saat "new NamaClass()" dipanggil 4. Boleh ada lebih dari satu constructor (Constructor Overloading)

Jenis-Jenis Constructor

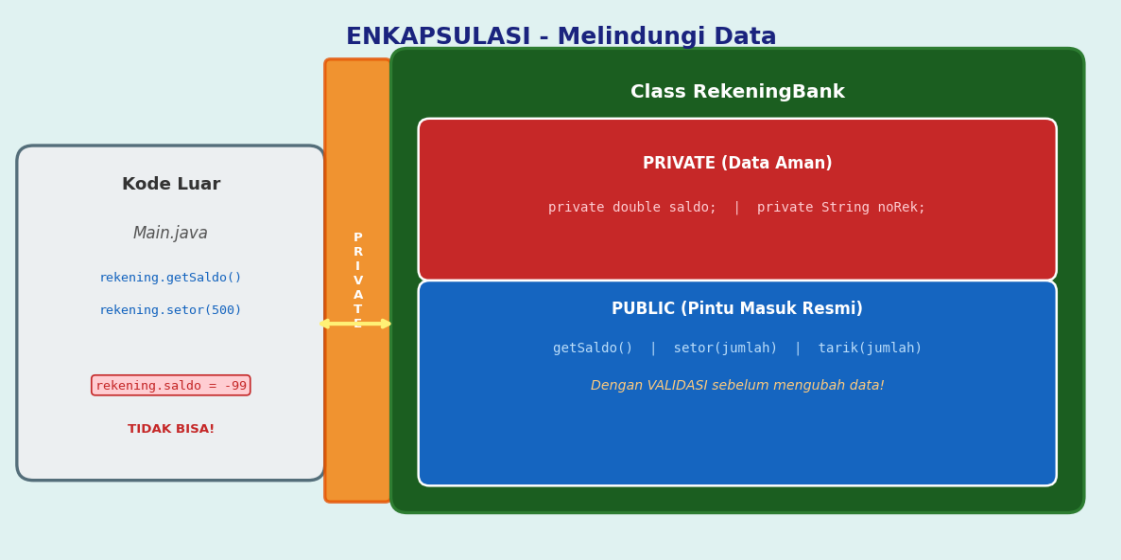
```
public class Mahasiswa {
    private String nama;
    private String nim;
    private double ipk;

    // 1. DEFAULT CONSTRUCTOR - tanpa parameter
    //    Dipakai saat: new Mahasiswa()
    public Mahasiswa() {
        this.nama = "Belum diisi";
        this.nim  = "00000000";
```

```
        this.ipk  = 0.0;
    }

    // 2. PARAMETERIZED CONSTRUCTOR - dengan parameter
    //    Dipakai saat: new Mahasiswa("Budi", "22050001", 3.75)
    public Mahasiswa(String nama, String nim, double ipk) {
        this.nama = nama;  // "this.nama" = atribut class
        this.nim  = nim;   // "nama" tanpa this = parameter
        this.ipk  = ipk;
    }
}

// Penggunaan:
Mahasiswa mhs1 = new Mahasiswa();                    // pakai default
Mahasiswa mhs2 = new Mahasiswa("Budi","22050001",3.75); // pakai parameter
```

Kata Kunci "this"

Keyword "this" digunakan untuk membedakan atribut class dengan parameter yang namanya sama. "this.nama" = atribut class, sedangkan "nama" = parameter konstruktor. Tanpa "this", Java akan bingung mana yang dimaksud!

1.5 Getter dan Setter (Enkapsulasi)



Gambar 1.4: Enkapsulasi - data private diakses melalui "pintu resmi" getter/setter

Getter dan Setter adalah method yang digunakan untuk mengakses (getter) dan mengubah (setter) nilai atribut private. Ini adalah implementasi nyata dari prinsip Enkapsulasi dalam OOP.

```
public class RekeningBank {
    // PRIVATE - tidak bisa diakses langsung dari luar
    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 - hanya untuk MEMBACA nilai
    public String getNomorRekening() { return nomorRekening; }
    public double getSaldo()          { return saldo; }

    // SETTER dengan VALIDASI - untuk MENGUBAH nilai
    public void setNamaPemilik(String nama) {
        if (nama != null && !nama.isEmpty()) {  // validasi dulu!
            this.namaPemilik = nama;
        }
    }

    // Method bisnis (bukan setter langsung)
    public void setor(double jumlah) {
        if (jumlah > 0) { saldo += jumlah; }
    }
    public void tarik(double jumlah) {
        if (jumlah > 0 && jumlah <= saldo) { saldo -= jumlah; }
        else { System.out.println("Saldo tidak cukup!"); }
    }
}
```

Penjelasan Code

• getNomorRekening() = getter untuk membaca nomorRekening (read-only, tidak ada setter) • getSaldo() = getter untuk membaca saldo • setNamaPemilik() = setter dengan validasi - nama tidak boleh kosong • setor() & tarik() = method bisnis yang mengubah saldo dengan aturan tertentu • nomorRekening bersifat read-only karena tidak ada setternya - ini disengaja!

BAB 2: TIPE DATA DAN CONTROL PROGRAM

2.1 Tipe Data dalam Java

Java membagi tipe data menjadi dua kategori utama. Pemilihan tipe data yang tepat sangat penting untuk efisiensi memori dan ketepatan perhitungan.

Tipe Data Primitif

Tipe	Ukuran	Rentang	Contoh Penggunaan
int	32 bit	-2^31 s/d 2^31-1	int jumlah = 1000; // Bilangan bulat umum
double	64 bit	~4.9E-324 s/d 1.8E308	double ipk = 3.75; // Bilangan desimal
boolean	1 bit	true / false	boolean aktif = true; // Logika benar/salah
char	16 bit	Unicode 0-65535	char huruf = 'A'; // Satu karakter
String	Var	Teks bebas	String nama = "Budi"; // Kumpulan karakter
long	64 bit	-2^63 s/d 2^63-1	long populasi = 7800000000L;
float	32 bit	~1.4E-45 s/d 3.4E38	float nilai = 3.14f;

2.2 Control Program - Percabangan

Percabangan digunakan untuk membuat program mengambil keputusan berdasarkan kondisi tertentu. Ada dua jenis utama: if-else dan switch.

If-Else Statement

```
// Contoh: menentukan nilai huruf
int nilai = 75;

if (nilai >= 90) {
    System.out.println("Grade: A");     // jika nilai >= 90
} else if (nilai >= 80) {
    System.out.println("Grade: B");     // jika nilai 80-89
} else if (nilai >= 70) {
    System.out.println("Grade: C");     // jika nilai 70-79
} else {
    System.out.println("Grade: D");     // jika nilai < 70
}
// Output: Grade: C  (karena nilai=75, masuk kondisi ketiga)
```

Penjelasan Alur If-Else

Java memeriksa kondisi dari atas ke bawah secara berurutan. Begitu menemukan kondisi yang TRUE, blok tersebut dieksekusi dan sisanya dilewati. Jika tidak ada yang TRUE, blok else (jika ada) yang dijalankan.

2.3 Control Program - Perulangan

Jenis-Jenis Loop

```
// 1. FOR LOOP - jumlah iterasi sudah diketahui
for (int i = 1; i <= 5; i++) {
    System.out.println("Iterasi ke: " + i);
}  // Output: Iterasi ke: 1, 2, 3, 4, 5

// 2. WHILE LOOP - iterasi selama kondisi terpenuhi
int counter = 1;
while (counter <= 5) {
    System.out.println("Counter: " + counter);
    counter++;  // WAJIB ada agar tidak infinite loop!
}

// 3. FOR-EACH - iterasi semua elemen array/collection
String[] nama = {"Ali", "Budi", "Citra"};
for (String n : nama) {
    System.out.println("Halo, " + n);
}  // Output: Halo Ali, Halo Budi, Halo Citra
```

Kapan Pakai Loop Mana?

• FOR LOOP: gunakan saat kamu TAHU berapa kali harus berulang (contoh: cetak 1-100) • WHILE LOOP: gunakan saat tidak tahu berapa kali, tergantung kondisi (contoh: baca input sampai user ketik "selesai") • FOR-EACH: gunakan untuk melooping semua elemen array/list, paling bersih dan mudah dibaca

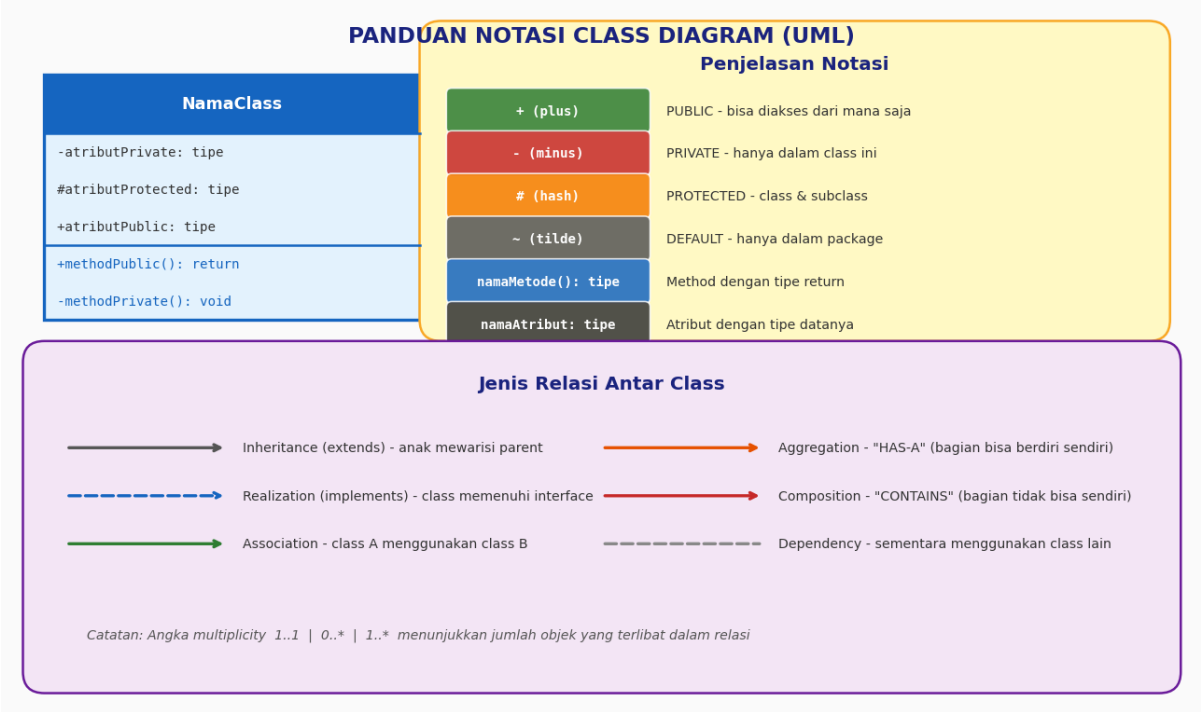
BAB 3: CLASS DIAGRAM

3.1 Apa itu Class Diagram?

Class Diagram adalah diagram visual dalam notasi UML (Unified Modeling Language) yang menggambarkan struktur sistem dengan menampilkan class-class yang ada, atribut dan method di dalamnya, serta relasi antar class. Class Diagram dibuat SEBELUM menulis kode untuk merencanakan arsitektur program.

> **Analogi Sederhana**
>
> Class Diagram seperti denah rumah sebelum dibangun. Denah (Class Diagram) menunjukkan berapa kamar, letaknya di mana, dan bagaimana pintunya terhubung. Setelah denah jadi, barulah tukang (programmer) membangun rumahnya (menulis kode).

3.2 Notasi Class Diagram

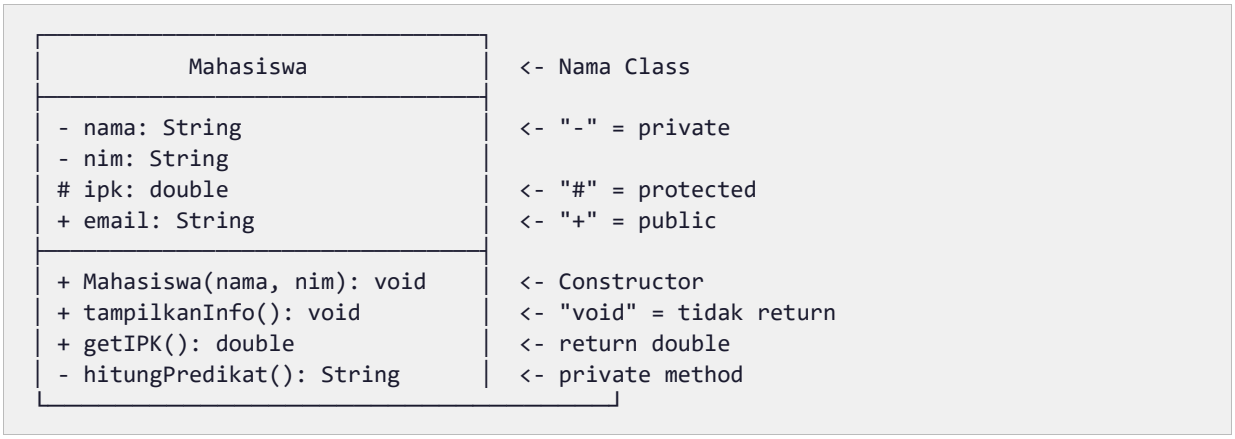


Gambar 3.1: Panduan lengkap notasi Class Diagram UML

Setiap class dalam diagram UML digambarkan sebagai kotak dengan tiga bagian: nama class (atas), atribut (tengah), dan method (bawah). Simbol +, -, # di depan nama menunjukkan access modifier.

3.3 Cara Membaca Class Diagram

Contoh: Membaca Satu Class



Cara Membacanya

• Nama Class di atas: "Mahasiswa" • Atribut: nama(private), nim(private), ipk(protected), email(public) - semua bertipe String/double • Method: constructor Mahasiswa(), getter getIPK() yang return double, dan method private • "void" artinya method tidak mengembalikan nilai apapun

3.4 Jenis Relasi Antar Class

1. Inheritance (Generalization) - Panah Solid ke Parent

```
// Class Diagram:
// Kendaraan <|-- Mobil   (Mobil extends Kendaraan)
// Kendaraan <|-- Motor   (Motor extends Kendaraan)

// Dalam Code Java:
public class Kendaraan {          // PARENT
    protected String merk;
    public void jalan() { ... }
}

public class Mobil extends Kendaraan {  // CHILD
    private int jumlahPintu;
    public void bukaPintu() { ... }
}
```

2. Realization/Implementation - Panah Putus-Putus ke Interface

```
// Class Diagram:
// Bersuara <.. Kucing   (Kucing implements Bersuara)

// Dalam Code Java:
interface Bersuara {
    void bersuara();  // kontrak yang harus dipenuhi
}
```

```
public class Kucing implements Bersuara {
    @Override
    public void bersuara() {
        System.out.println("Meong!");
    }
}
```

3. Association - Class A Menggunakan Class B

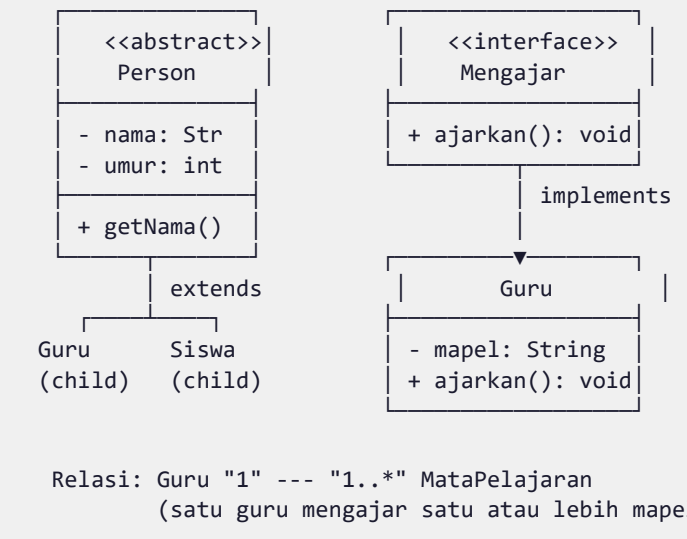
```
// Class Diagram:
// Dokter "1" --> "0..*" Pasien
// (Satu dokter bisa menangani banyak pasien)

// Dalam Code Java:
public class Dokter {
    private ArrayList<Pasien> daftarPasien;  // Dokter punya daftar pasien

    public void periksaPasien(Pasien p) {    // method menggunakan Pasien
        // ...
    }
}
```

3.5 Contoh Membuat Class Diagram Sederhana

Langkah-langkah membuat class diagram:

1. Identifikasi entitas/benda utama dalam sistem (misal: Sekolah, Guru, Siswa, Mata Pelajaran)
2. Tentukan atribut dan method dari setiap class
3. Tentukan relasi antar class: siapa parent/child, siapa yang menggunakan siapa
4. Tentukan multiplicity (1..1, 1..*, 0..*) pada setiap relasi
5. Gambar diagram menggunakan notasi UML

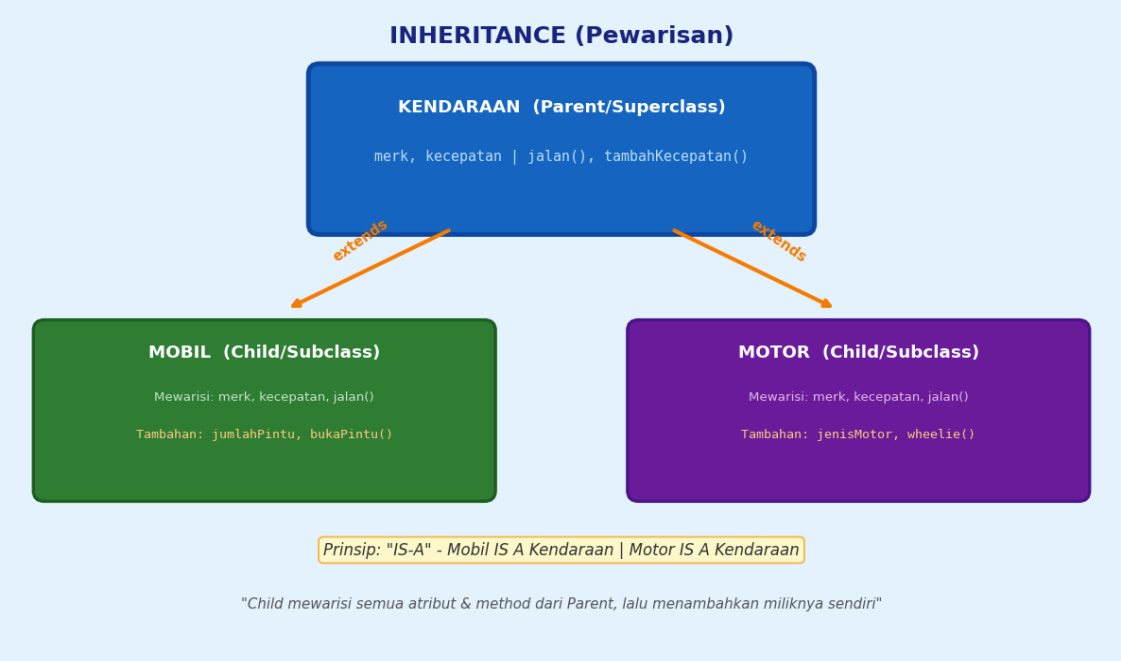
Contoh: Class Diagram Sistem Sekolah




```
Relasi: Guru "1" --- "1..*" MataPelajaran
        (satu guru mengajar satu atau lebih mapel)
```

BAB 4: KONSEP BERBASIS OBJEK - INHERITANCE DAN POLYMORPHISM

4.1 Inheritance (Pewarisan)



Gambar 4.1: Pohon Inheritance - Mobil dan Motor mewarisi Kendaraan

Inheritance memungkinkan sebuah class mewarisi semua atribut dan method dari class lain. Ini mendukung prinsip DRY (Don't Repeat Yourself) - kode yang sama tidak perlu ditulis ulang.

> **Prinsip "IS-A"**
> Inheritance cocok digunakan jika hubungannya adalah "IS-A" (adalah sebuah): • Mobil IS A Kendaraan - benar! (gunakan inheritance) • Motor IS A Kendaraan - benar! • Mahasiswa IS A Person - benar! • Mobil IS A Garasi - salah! (jangan gunakan inheritance, ini "HAS-A")

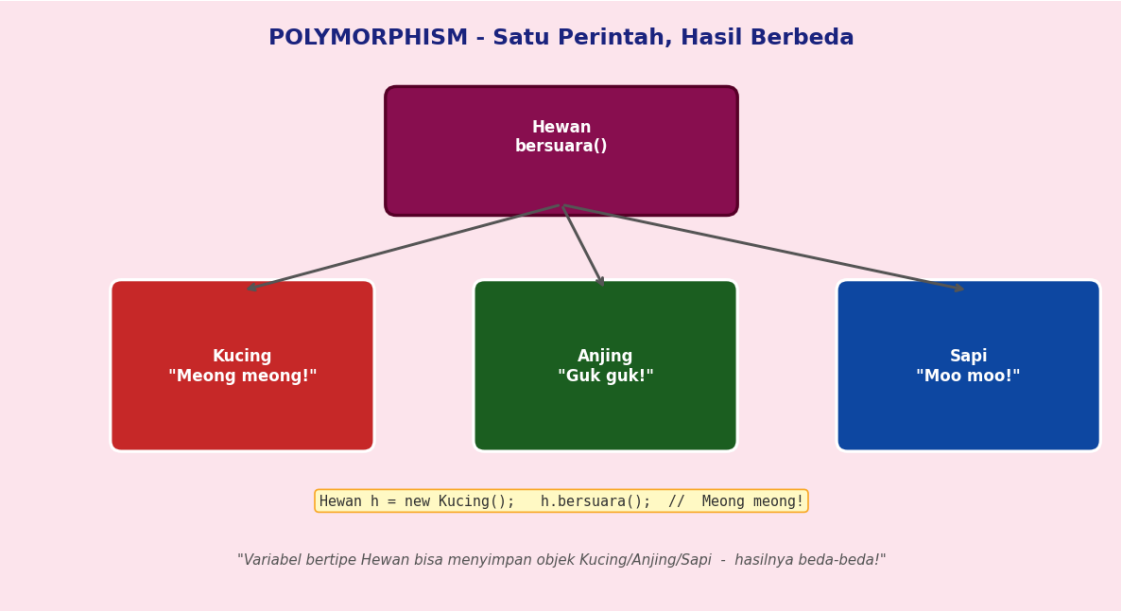
```
// SUPERCLASS / PARENT CLASS
public class Kendaraan {
    protected String merk;        // protected agar subclass bisa akses
    protected int kecepatan;

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

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

// SUBCLASS / CHILD CLASS - menggunakan kata kunci "extends"
public class Mobil extends Kendaraan {
```

```
    private int jumlahPintu;      // atribut TAMBAHAN khusus Mobil

    public Mobil(String merk, int jumlahPintu) {
        super(merk);              // "super()" memanggil constructor PARENT
        this.jumlahPintu = jumlahPintu;
    }

    public void bukaPintu() {     // method TAMBAHAN khusus Mobil
        System.out.println("Membuka " + jumlahPintu + " pintu");
    }
}

// Penggunaan:
Mobil m = new Mobil("Toyota", 4);
m.jalan();      // MEWARISI dari Kendaraan - bisa dipakai!
m.bukaPintu();  // Method milik sendiri
```

Penjelasan Code

• "extends Kendaraan" = Mobil mewarisi semua yang ada di Kendaraan • "super(merk)" = memanggil constructor parent agar atribut parent terinisialisasi • protected pada parent = bisa diakses oleh Mobil (subclass) langsung • Mobil bisa pakai method jalan() meski tidak mendefinisikannya ulang

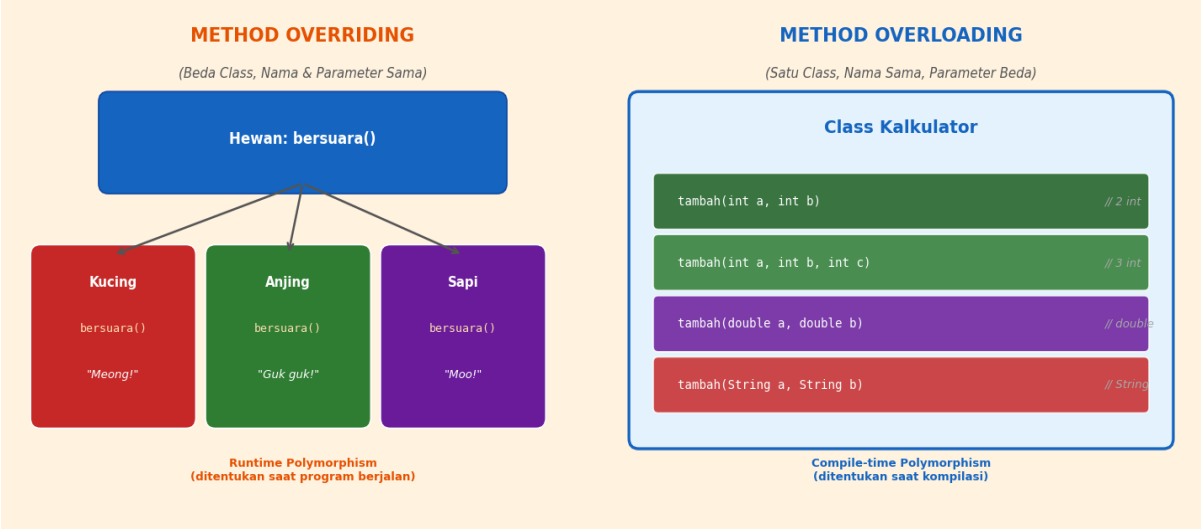
4.2 Polymorphism (Polimorfisme)



Gambar 4.2: Polymorphism - satu perintah bersuara(), tiga hasil berbeda

Polymorphism berarti "banyak bentuk". Dalam OOP, satu method yang sama bisa menghasilkan perilaku berbeda tergantung objek yang memanggilnya. Ada dua cara mencapainya: Method Overriding dan Method Overloading.

4.3 Method Overriding & Overloading



Gambar 4.3: Perbedaan Overriding (beda class) vs Overloading (satu class)

Method Overriding - Mendefinisikan Ulang di Subclass

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

public class Kucing extends Hewan {
    @Override  // anotasi opsional tapi DIREKOMENDASIKAN
    public void bersuara() {         // nama & parameter SAMA persis
        System.out.println("Meong meong!");  // implementasi BERBEDA
    }
}

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

// POLYMORPHISM:
Hewan h1 = new Kucing();  // variabel tipe Hewan, objek Kucing
Hewan h2 = new Anjing();  // variabel tipe Hewan, objek Anjing
h1.bersuara();  // Output: "Meong meong!"  (ditentukan saat runtime)
h2.bersuara();  // Output: "Guk guk!"
```

Method Overloading - Nama Sama, Parameter Beda

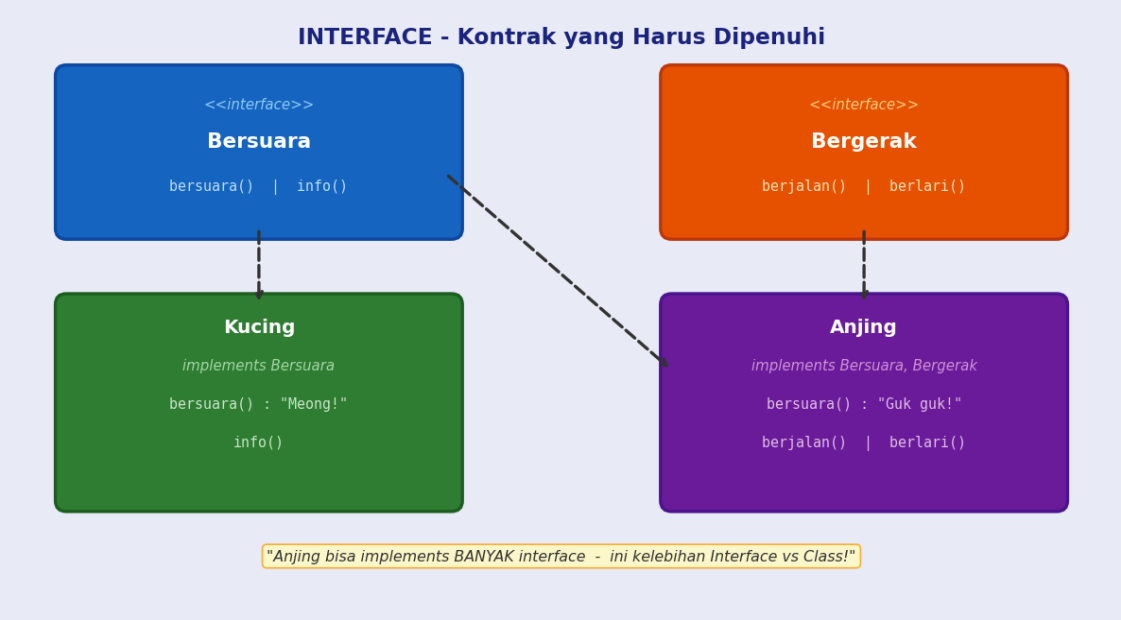
```
public class Kalkulator {
    // Overloading 1: dua int
    public int tambah(int a, int b) {
        return a + b;
```

```
    }

    // Overloading 2: tiga int (jumlah parameter beda)
    public int tambah(int a, int b, int c) {
        return a + b + c;
    }

    // Overloading 3: tipe parameter beda
    public double tambah(double a, double b) {
        return a + b;
    }

    // Overloading 4: tipe String (konkatenasi)
    public String tambah(String a, String b) {
        return a + b;
    }
}

Kalkulator k = new Kalkulator();
System.out.println(k.tambah(5, 3));           // 8   (int)
System.out.println(k.tambah(5, 3, 2));        // 10  (int 3 param)
System.out.println(k.tambah(1.5, 2.5));       // 4.0 (double)
System.out.println(k.tambah("Hello", "!"));   // Hello! (String)
```

METHOD OVERRIDING	METHOD OVERLOADING
• Terjadi di SUBCLASS yang berbeda • Nama dan parameter HARUS sama persis • Mengubah perilaku method parent • Runtime Polymorphism • Gunakan @Override untuk keamanan	• Terjadi di class yang SAMA • Nama sama, parameter HARUS berbeda • Menambah variasi cara panggil method • Compile-time Polymorphism • Tidak perlu anotasi khusus

BAB 5: INTERFACE DAN PAKET

5.1 Interface



Gambar 5.1: Interface sebagai kontrak - Anjing bisa implement banyak interface

Interface adalah "kontrak" yang berisi daftar method yang HARUS diimplementasikan oleh class yang menggunakannya. Berbeda dengan class, satu class bisa mengimplementasikan BANYAK interface sekaligus.

Interface vs Abstract Class

INTERFACE: kontrak murni, semua method abstract, bisa multiple implement ABSTRACT CLASS: class biasa yang punya method abstract, hanya single extend Gunakan Interface saat mendefinisikan "bisa apa" (kemampuan) Gunakan Abstract Class saat mendefinisikan "adalah apa" (identitas)

```
// Mendefinisikan Interface
public interface Bersuara {
    void bersuara();  // otomatis public abstract - tidak perlu ditulis
    void info();
}

public interface Bergerak {
    void berjalan();
    void berlari();
}

// Class mengimplementasikan 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 + ": Meong meong!");
```

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

// Class mengimplementasikan DUA interface sekaligus
public class Anjing implements Bersuara, Bergerak {
    private String nama;
    public Anjing(String nama) { this.nama = nama; }

    @Override public void bersuara()  { System.out.println(nama + ": Guk!"); }
    @Override public void info()      { System.out.println("Ini anjing: " + nama); }
    @Override public void berjalan()  { System.out.println(nama + " berjalan"); }
    @Override public void berlari()   { System.out.println(nama + " berlari"); }
}
```

Penjelasan Code

• Interface hanya berisi deklarasi method, tanpa isi (body) • "implements Bersuara" = Kucing berjanji akan mengimplementasikan semua method di Bersuara • Anjing "implements Bersuara, Bergerak" = mengimplementasikan DUA interface - ini tidak bisa dengan class biasa! • Jika ada method di interface yang tidak diimplementasikan, program akan ERROR saat dikompilasi

5.2 Package (Paket)

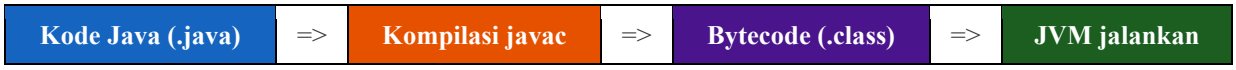
Package adalah cara mengorganisir class ke dalam folder-folder logis berdasarkan fungsinya. Package mencegah konflik nama dan membuat kode lebih terstruktur dan mudah dikelola.

```
// Struktur folder package sistem klinik:
klinik/
├── model/           <- Semua class entitas data
│   ├── Pasien.java
│   ├── Dokter.java
│   └── Perawat.java
├── interfaces/      <- Semua interface
│   ├── Identifiable.java
│   └── Treatable.java
├── service/         <- Logika bisnis
│   └── KlinikService.java
└── Main.java        <- Entry point program

// Deklarasi package (baris pertama di setiap file)
package klinik.model;  // file ini ada di folder klinik/model/

// Menggunakan class dari package lain
import klinik.model.Pasien;     // import satu class
import klinik.model.*;          // import semua class di folder model
import java.util.ArrayList;     // import dari Java standard library
```

BAB 6: MENGKOMPILASI PROGRAM JAVA

6.1 Proses Kompilasi Java

Kompilasi adalah proses mengubah kode Java (.java) menjadi bytecode (.class) yang bisa dijalankan oleh JVM (Java Virtual Machine). Keunikan Java: "Write Once, Run Anywhere" - bytecode yang sama bisa jalan di semua sistem operasi.



6.2 Jenis-Jenis Error dalam Java

1. Syntax Error (Compile-time Error) - Kesalahan Penulisan

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

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

// Pesan error dari Java:
// Main.java:1: error: ';' expected
// int angka = 10
//           ^
// Arti: di baris 1, setelah "10", Java mengharapkan titik koma
```

2. Runtime Error - Error Saat Program Berjalan

```
// NullPointerException - mengakses objek yang null
String nama = null;
System.out.println(nama.length());  // ERROR! nama belum diisi

// ArrayIndexOutOfBoundsException - index melebihi batas array
int[] angka = {1, 2, 3};  // index valid: 0, 1, 2
System.out.println(angka[5]);  // ERROR! index 5 tidak ada

// ArithmeticException - pembagian dengan nol
int hasil = 10 / 0;  // ERROR! tidak boleh bagi nol
```

3. Logical Error - Program Jalan Tapi Hasil Salah

```
// Logical Error: operator salah
int nilai = 70;
if (nilai > 70) {  // seharusnya >= 70, jadi nilai 70 tidak lulus!
    System.out.println("Lulus");
} else {
    System.out.println("Tidak Lulus");  // Output ini yang muncul
}
// Program tidak error, tapi hasilnya SALAH logikanya
// Ini paling susah ditemukan karena tidak ada pesan error
```

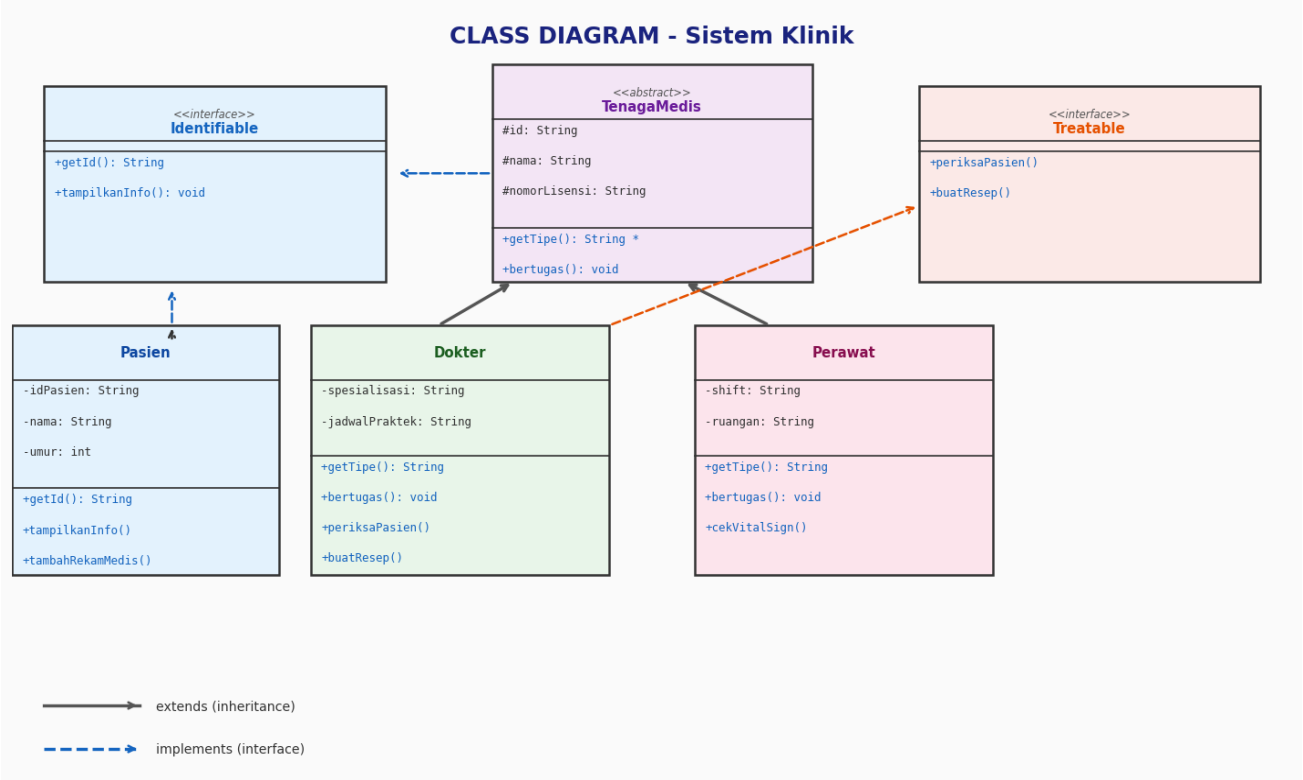
Tips Menghindari Error

1. Syntax Error: Gunakan IDE (IntelliJ/Eclipse) yang langsung memberi tanda merah 2. Runtime Error: Gunakan try-catch untuk menangkap error, selalu cek null sebelum pakai 3. Logical Error: Buat test case, trace kode secara manual, gunakan debugger IDE 4. Umum: Beri nama variabel yang jelas, beri komentar, test setiap fungsi kecil dulu

BAB 7: PROJECT PRAKTIKUM - SISTEM MANAJEMEN KLINIK

7.1 Deskripsi Project

Project ini mengintegrasikan semua konsep OOP yang telah dipelajari dalam satu aplikasi nyata: Sistem Manajemen Klinik Kesehatan. Project ini menunjukkan bagaimana OOP membuat kode terstruktur, mudah dikembangkan, dan mudah dirawat.



Gambar 7.1: Class Diagram lengkap Sistem Klinik - menunjukkan semua relasi antar class

Konsep OOP dalam Project Klinik

• CLASS & OBJECT: Pasien, Dokter, Perawat, Appointment, RekamMedis, Resep • ENCAPSULATION: Semua atribut private, akses via getter/setter • INHERITANCE: TenagaMedis (abstract) <-- Dokter & Perawat • POLYMORPHISM: TenagaMedis[] bisa menampung Dokter/Perawat, getTipe() berbeda • INTERFACE: Identifiable (getId, tampilkanInfo), Treatable (periksaPasien, buatResep) • PACKAGE: model, interfaces, service terpisah untuk kerapihan kode

7.2 Implementasi Code Kunci

Abstract Class TenagaMedis

```
// Abstract class - tidak bisa langsung dibuat objeknya
// Digunakan sebagai PARENT bersama Dokter dan Perawat
public abstract class TenagaMedis implements Identifiable {
    protected String id;
```

```
    protected String nama;
    protected String nomorLisensi;
    protected int tahunPengalaman;

    // Constructor untuk dipakai subclass via super()
    public TenagaMedis(String id, String nama, String lisensi, int pengalaman) {
        this.id = id;
        this.nama = nama;
        this.nomorLisensi = lisensi;
        this.tahunPengalaman = pengalaman;
    }

    // Abstract method - WAJIB diimplementasikan subclass
    public abstract String getTipe();

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

Apa itu Abstract Class?

Abstract class adalah class "setengah jadi" yang tidak bisa langsung diinstansiasi (tidak bisa new TenagaMedis()). Ia mendefinisikan kerangka umum, lalu subclass mengisi detailnya. Method abstract adalah method tanpa implementasi - subclass WAJIB mengisinya.

Class Dokter - Inheritance + Multiple Interface

```
// Dokter mewarisi TenagaMedis DAN mengimplementasikan Treatable
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); // panggil constructor TenagaMedis
        this.spesialisasi = spesialisasi;
        this.jadwalPraktek = jadwal;
    }

    @Override
    public String getTipe() { return "Dokter"; }  // dari abstract class

    @Override
    public void tampilkanInfo() {  // dari Identifiable
        System.out.println("=== INFORMASI DOKTER ===");
        System.out.println("Nama: Dr. " + nama);
        System.out.println("Spesialisasi: " + spesialisasi);
    }

    @Override
    public void periksaPasien(Pasien pasien, String keluhan) {  // dari Treatable
        System.out.println("Dr. " + nama + " memeriksa " + pasien.getNama());
    }
```

```
    @Override
    public Resep buatResep(Pasien pasien, String obat, String dosis) {
        return new Resep(this, pasien, obat, dosis);
    }
}
```

Penjelasan Code

• "extends TenagaMedis" = Dokter mewarisi semua dari TenagaMedis • "implements Treatable" = Dokter berjanji memenuhi kontrak interface Treatable • "super(id, nama, ...)" = memanggil constructor parent TenagaMedis • @Override pada getTipe(), tampilkanInfo(), periksaPasien(), buatResep() = memenuhi kewajiban dari parent dan interface • Ini contoh nyata penggunaan Inheritance + Interface sekaligus!

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:

- Oracle Corporation. (n.d.). *Java documentation*. https://docs.oracle.com/javase/
- Oracle Corporation. (n.d.). *The Java tutorials*. https://docs.oracle.com/javase/tutorial/
- GeeksforGeeks. (n.d.). *Java programming language*. https://www.geeksforgeeks.org/java/
- Javatpoint. (n.d.). *Java tutorial*. https://www.javatpoint.com/java-tutorial
- W3Schools. (n.d.). *Java tutorial*. https://www.w3schools.com/java/

--- Akhir Materi ---

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