

PENROGRAMAN BERORIENTASI OBJEK

4 Pilar Konsep Pemrograman Berorientasi Objek

I Gusti Ngurah Suryantara, S.Kom., M.Kom

Pendahuluan

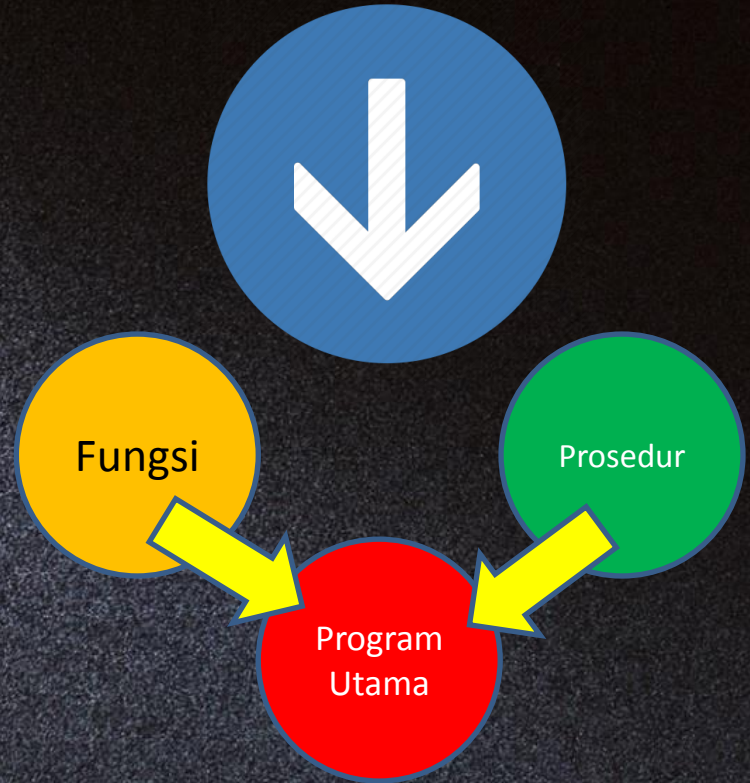
Konsep pemrograman dapat dibedakan menjadi:

- **Non Objek:** strukur / prosedural / fungsional, dll.
- **Berorientasi Objek.**

- Pada pendekatan non objek yang menjadi perhatian adalah: pembuatan spesifikasi dan dekomposisi sistem secara fungsional.
- Sedangkan pada pendekatan berorientasi objek menekankan pada saat melakukan indentifikasi objek dari domain aplikasi, kemudian pembuatan method yang sesuai.

Konsep Non Objek

- Pendekatan bersifat:
 - Top-Down (dari atas ke bawah).
 - Dekomposisi (memecah-mecah program ke dalam sub program / fungsi), hal ini dapat dilakukan dengan membuat prosedur atau fungsi.



Berorientasi Objek

- Pendekatan secara dunia nyata yaitu: objek, apapun di pandang sebagai objek.
- Karena mengadopsi pendekatan dunia nyata maka konsep ini dibentuk dari kelas dan instansiasi kelas menjadi objek.
- Membuat kelas dan objek menjadi kunci utama pada konsep berorientasi objek.
- Objek adalah benda (dalam dunia nyata), baik yang berwujud nyata maupun yang tidak nyata (seperti sistem).
- Objek dalam dunia nyata dapat berupa: Mobil, dll.
- Pada program objek dapat berupa: form, basis data, report, dan lainnya.

Berorientasi Objek (Lanjutan)



Objek nyata: mobil

DETAIL TRANSAKSI PENGADAAN BARANG

Detail Barang Harga Jual Barang

ID TRANSAKSI: NAMA BARANG:

ID BARANG: HARGA BELI:

KATEGORI BARANG: JUMLAH: buah/lusin/pem/kg

TGL. KADALUWARS: ☐ Ada ☐ Tidak Ada

Title 1	Title 2	Title 3	Title 4

Back TOTAL TRANSAKSI : 0 Delete Edit Insert

Objek tak nyata: form aplikasi

Berorientasi Objek (Lanjutan)

Kelas: Mahasiswa

- NIM : String
 - Nama : String
 - UTS : double
 - Tugas : double
 - UAS : double
-
- + TotalNilai() : double
 - + Keterangan() : char

Instansiasi
kelas menjadi
objek

Objek: Mahasiswa

- NIM : 20200001
 - Nama : **Gusti**
 - UTS : 80
 - Tugas : 80
 - UAS : 80
-
- + TotalNilai() : 80
 - + Keterangan() : A

4 Pilar Konsep PBO

- 4 pilar Pemrograman Berorientasi Objek (PBO)
 1. **Abstraksi** (*Abstraction*).
 2. **Pewarisan** (*Inheritance*).
 3. **Pembungkusan** (*Encapsulation*).
 4. **Banyak Bentuk** (*Polymorphism*).



Abstraksi

- **Abstraksi:** cara melihat objek dalam bentuk lebih sederhana.
- Seperti melihat objek komputer bukan sebagai sebuah kumpulan rangkaian elektronik.
- Namun sebagai entitas tunggal yang mempunyai sifat dan karakteristiknya sendiri.



Objek Komputer

Abstraksi (Lanjutan)

- Apa saja yang dapat kita lakukan pada Abstraksi pada konsep PBO:
 - Membuat kelas.
 - Membuat atribut/variabel/field.
 - Property method.
 - Membuat method.
 - Membuat objek.
 - Event.
 - Overloading.
 - Overriding.
 - Constructor
 - Destructor.
 - Kelas Abstrak.
 - Interface.
 - dan yang lainnya

Abstraksi (Lanj...): Class & Object

- Semua objek di dalam bahasa pemrograman Java diturunkan dari kelas.
- Kelas ini bisa disebut sebagai superclass atau nenek moyang dari semua kelas yang ada di dalam bahasa pemrograman Java.

Nama_Kelas
- atribut : int
+ method() : void

- Mendeklarasikan class:
Jangkauan_nilai class
nama_kelas
public class Mahasiswa

Abstraksi (Lanj...): Class & Object

- **Kelas (*Class*):**

- Kelas adalah: cetak biru (*blueprint*) dari objek, yang dapat memiliki atribut, dan method.

Nama kelas

Nama atribut

Nama method

Kelas: Mahasiswa	
- NIM : String	
- Nama : String	
- UTS : double	
- Tugas : double	
- UAS : double	
+ TotalNilai() : double	
+ Keterangan() : char	

Instansiasi
kelas
menjadi
objek

- **Objek (*Object*):**

- Objek adalah instansiasi dari kelas.

Nama objek

Nama atribut

Nama method

Objek: Mahasiswa	
- NIM : 20200001	
- Nama : Gusti	
- UTS : 80	
- Tugas : 80	
- UAS : 80	
+ TotalNilai() : 80	
+ Keterangan() : A	

Abstraksi (Lanj...): Field

- **Atribut/Field** adalah: variabel yang didefinisikan di dalam kelas, dan disebut juga sebagai member variabel.
- Pada umumnya field merupakan variabel private dalam blok kelas.
- Apabila menggunakan kelas yang memungkinkan untuk memodifikasi nilai suatu field, sebaiknya menggunakan property method yang berkaitan.
- Property method yang akan nantinya menyediakan akses ke field.

Abstraksi (Lanj...): Field

- Jangkauan nilai untuk suatu atribut/variabel adalah:
 - Public
 - Private
 - Friend
 - Protected
 - Protected Friend
- Mendeklarasikan field / variabel:

tipe_data nama_variabel;

Atau

Jangkuan_nilai tipe_data
nama_variabel;

Abstraksi (Lanj...): Property Method

- **Property method** adalah sebuah method khusus yang digunakan untuk mendapatkan atau mengubah nilai dari sebuah field dalam kelas. Penggunaan property biasanya ditunjukkan pada field yang bersifat private.
- Field yang bersifat public dapat diakses langsung dari luar kelas, sedangkan field yang bersifat private hanya dapat diakses dari luar kelas melalui property method.

Abstraksi (Lanj...): Property Method

Tanpa keyword this

```
public void setIdDokter (String newValue)
{
    IdDokter = newValue;
}

Public String getIdDokter()
{
    return IdDokter;
}
```

Dengan keyword this

```
public void setIdDokter (String idDokter)
{
    this.IdDokter = IdDokter;
}

Public String getIdDokter()
{
    return IdDokter;
}
```


Abstraksi (Lanj...): Method

- **Method** adalah prosedur dan function yang dimiliki sebuah kelas.
- Dengan kata lain method adalah prosedur dan fungsi di lingkungan kelas.
- Method menunjukkan apa saja yang dapat dilakukan oleh objek hasil instansiasi kelas tersebut, method juga biasanya memiliki kata kunci **public** atau **private**.
- Method (**static & non-static**)
- Method memiliki daftar paramter formal atau tanpa parameter, nilai kembalian atau tanpa nilai kembalian, dan shared atau non non-shared.
- Dimana method shared diakses melalui kelas, sedangkan method non-shared atau biasa disebut instance method, akan diakses melalui instansiasi kelas.

Abstraksi (Lanj...): Method

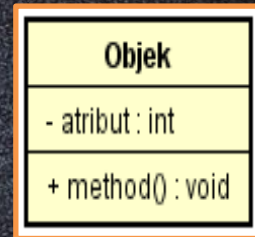
- Mendeklarasikan method dengan prosedur
- Mendeklarasikan method dengan fungsi

```
static void Luas(int Alas, int Tinggi)
{
    LuasSegitiga = 0.5 * Alas * Tinggi;
}
```

```
public double Luas(int Alas, int Tinggi)
{
    return 0.5 * Alas * Tinggi;
}
```


Abstraksi (Lanj...): Object

- **Objek** merupakan representasi nyata atau perwujudan (instance) dari kelas.
- Objek juga memiliki atribut, dan method yaitu tindakan yang dilakukan oleh objek.
- Sedangkan event adalah pemberitahuan yang diterima objek atau dikirimkan ke objek atau aplikasi lain.
- Event akan memicu suatu aksi.
- Sintaks mendeklarasikan objek dari suatu kelas adalah:
 - `nama_kelas nama_objek = new nama_kelas();`
- Contoh:
 - `clsMhs objMhs = new clsMhs();`



Abstraksi (Lanj...): Event

- **Event:** adalah kejadian yang terjadi terhadap sebuah objek.
- Contoh event dalam program yang sering kita temukan adalah mengklik tombol (button) event yang diberikan pada objek button adalah klik.
- Dalam objek textbox misalnya kita memberikan event keypress (menekan enter), dan yang lainnya.

```
private void txtKodeKeyPressed(java.awt.event.KeyEvent evt)
{
    ...
}
```

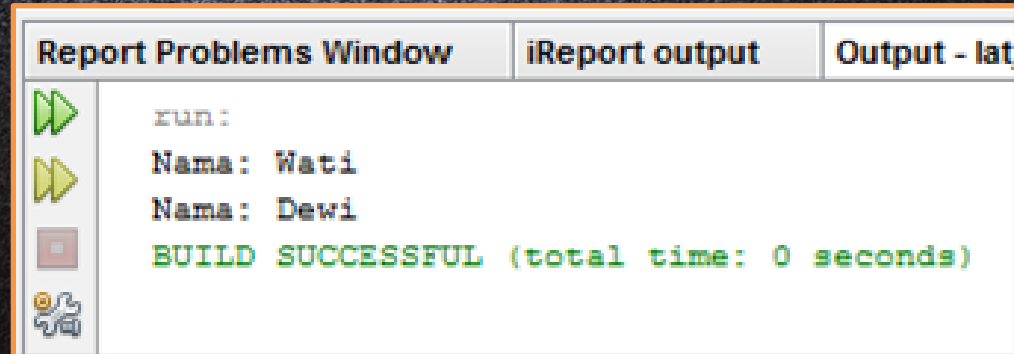

Abstraksi (Lanj...): Overloading

- **Overloading:** memungkinkan suatu kelas memiliki beberapa method dengan nama sama tetapi memiliki implementasi atau argumen yang berbeda, sepanjang deklarasi dan parameternya berbeda, atau disebut signaturenya berbeda.
- Hal ini dimungkinkan asalkan deklarasi method membuat penanda berbeda di satu kelas.

return type	nama method	daftar parameter
void	Coba	(int t1)
void	Coba	(int t1, int t2)
void	Coba	(int t1, int t2, int t3)
void	Coba	(int t1, int t2, int t3, int t4)
<hr/>	<hr/>	<hr/>
↓	↓	↓
sama	sama	berbeda

Abstraksi (Lanj...): Overriding

- **Overriding:** adalah method subclass sama dengan method super class, parameternya sama tetapi pernyataan atau implementasinya berbeda.
- Untuk mengimplementasikan overriding berkaitan dengan konsep pewarisan (Inheritance).



The screenshot shows a window titled "Report Problems Window" with tabs for "iReport output" and "Output - lat". The "iReport output" tab is active, displaying the following text:

```
run:  
Nama: Wati  
Nama: Dewi  
BUILD SUCCESSFUL (total time: 0 seconds)
```

On the left side of the window, there are four icons: a green play button, a yellow play button, a red square, and a blue icon with a magnifying glass.

Abstraksi (Lanj...): Overriding

- **Overriding:** adalah method subclass sama dengan method super class, parameternya sama tetapi pernyataan atau implementasinya berbeda.
- Untuk mengimplementasikan overriding berkaitan dengan konsep pewarisan (Inheritance).
- **Catatan**
Konsep overriding akan banyak digunakan pada konsep inheritance (pewarisan), encapsulation (pembungkusan) dan polymorphism (banyak bentuk).

Abstraksi (Lanj...): Constructor & Destructor

- **Konstruktur** (constructor) digunakan untuk menciptakan suatu objek, dan untuk menghancurkan suatu objek disebut destruktur (destructor).
- Pada bahasa pemrograman Java, menghancurkan objek dilakukan secara otomatis pada saat program selesai dijalankan.

```
Kelas: clsMahasiswa  
public class clsMahasiswa  
{  
    private String NIM;  
    private String Nama;  
  
    public clsMahasiswa(String NIM, String Nama)  
    {  
        this.NIM = NIM;  
        this.Nama = Nama;  
    }  
}
```


Abstraksi (Lanj...): Kelas Abstrak

- **Kelas abstrak:** adalah kelas yang tidak dapat dibuat instansnya.
- Kelas abstrak digunakan untuk menciptakan kelas yang hanya mendeklarasikan format generik tanpa mengimplementasikan secara detail fungsi-fungsi dari kelas abstrak.
- Akan tetapi semua hal yang berkaitan dengan fungsionalitas dari kelas tetap ada, seperti field, method, dan konstruktornya tetap dapat diakses.
- Sebuah kelas dikatakan kelas abstrak jika ada method hanya dideklarasikan saja (tidak diimplementasikan).
- Untuk membuat kelas abstrak menggunakan kata kunci "**abstract**", perintah `abstract` diletakkan sebelum nama Class dan nama Method.
- **Sintaks:** `public abstract class kelasAbstrak`
- Kelas abstrak juga dapat diturunkan.

Abstraksi (Lanj...): Interface

- **Interface:** merupakan kumpulan method-method abstrak.
- Sebuah kelas yang mengimplementasikan interface, mewarisi method-method abstrak dari interface tersebut.
- Persamaan interface dengan kelas:
 - Interface dapat memiliki banyak method.
 - Interface ditulis dalam file dengan ekstensi .java.
- Perbedaan interface dengan kelas:
 - Kita tidak bisa membuat interface dari sebuah interface.
 - Interface tidak memiliki konstruktor.
 - Semua method dalam interface adalah abstrak.
 - Interface tidak bisa memiliki field instance.
 - Interface tidak di-extend, tetapi di-implements oleh kelas.
 - Sebuah interface dapat meng-extend beberapa interface lainnya.

Abstraksi (Lanj...): Interface

- Kata kunci interface digunakan untuk mendeklarasikan sebuah interface.

```
/* File: NamaInterface.java */
Import java.lang.*;
// Satu atau beberapa statemen import lainnya

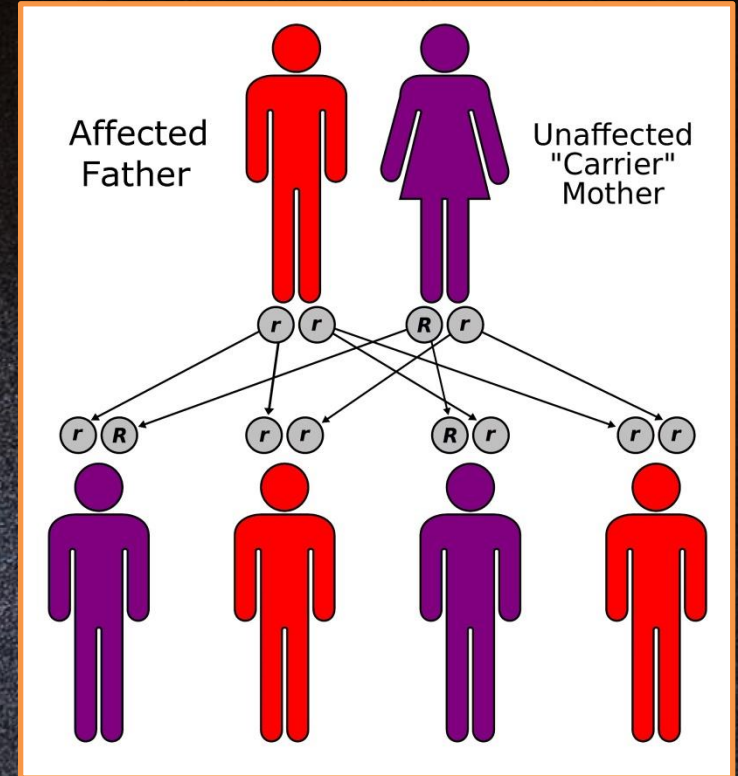
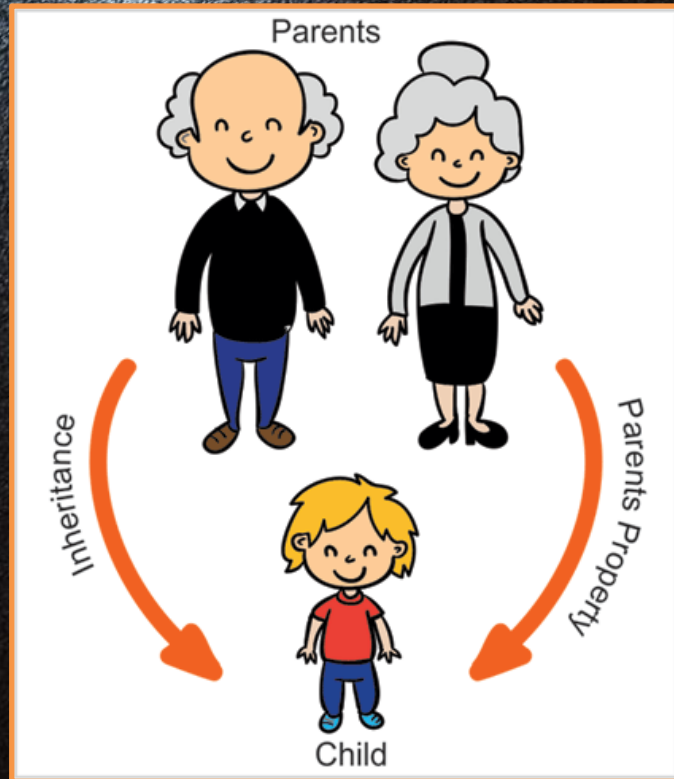
public interface NamaInterface
{
    //Satu atau beberapa field static final
    //Satu atau beberapa deklarasi method abstrak
    //catatan: modifier static tidak boleh digunakan
    dalam interface
}
```

- Interface memiliki properti berikut:
 - Interface secara implisit merupakan abstrak. Kita tidak perlu menggunakan kata kunci abstract ketika mendeklarasikan sebuah interface.
 - Setiap method dalam interface juga secara implisit merupakan abstrak, sehingga kita tidak perlu menggunakan kata kunci abstract.
 - Method dalam interface secara implisit merupakan public.

Pewarisan

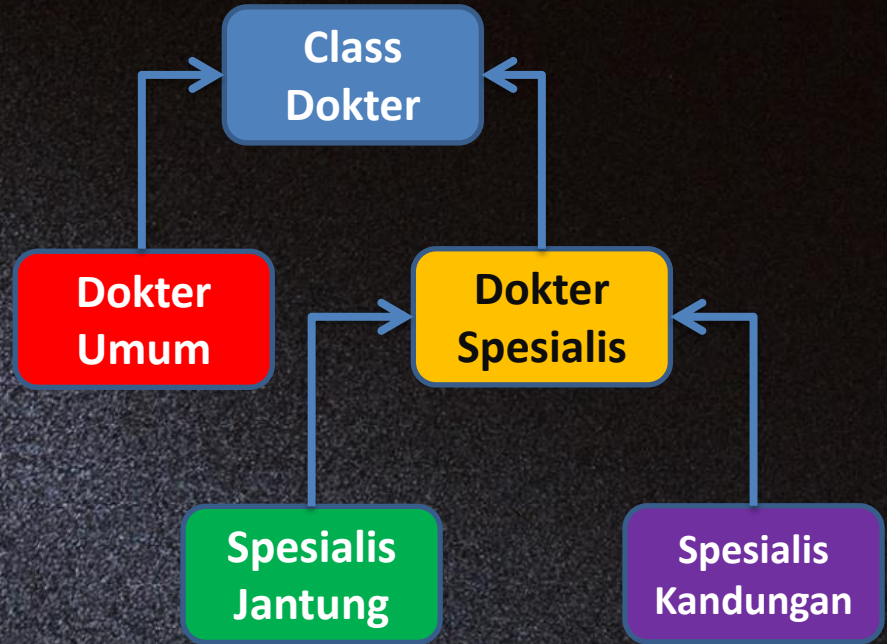
- Salah satu yang menjadi keunggulan konsep pemrograman berorientasi objek dengan non objek adalah: pewarisan (*inheritance*).
- Dengan adanya pewarisan maka: hal yang sama tidak perlu dibuat ulang, cukup diwariskan pada kelas turunan.
- Pada konsep non objek, konsep pewarisan tidak dikenal, sehingga harus membuatnya lagi.
- Hal ini tentu tidak efisien, pada berorientasi objek sesuatu yang sama dapat diwariskan. Sehingga kelas induk mewariskan semua atribut dan method ke kelas turunannya.

Pewarisan (Lanjut)



Pewarisan (Lanjutan)

- Jenis pewarisan dalam konsep PBO, yaitu:
 1. Single inheritance.
 2. Multilevel inheritance.
 3. Hierarchical inheritance.
 4. Multiple inheritance.
 5. Hybrid inheritance.
 6. Multi-path inheritance.



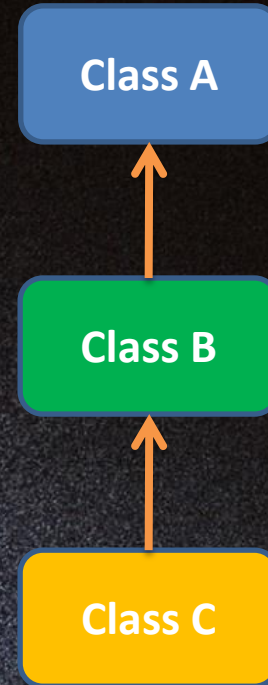
Pewarisan (Lanjutan)

- **Single Inheritance:** sebuah kelas turunan implementasi hanya dari satu kelas induk.
- Contoh: jika kelas B turunan dari kelas A, maka kelas B akan mewarisi semua yang ada pada kelas A.



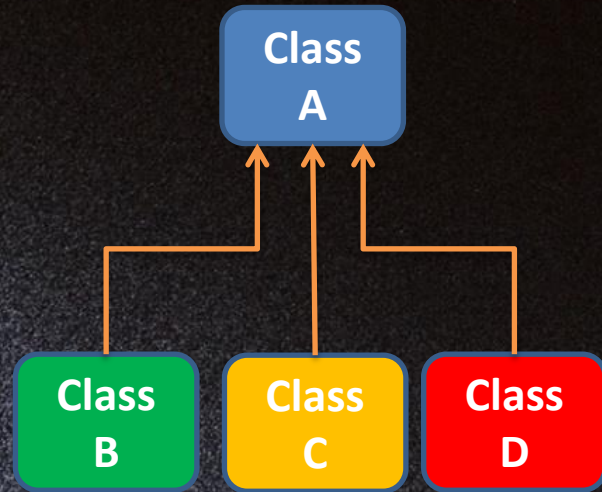
Pewarisan (Lanjutan)

- **Multilevel Inheritance:**
Jika kelas C inheritance dari kelas B, dan kelas B inherits dari kelas A, kelas C akan mewarisi semua member yang dideklarasikan di kelas B dan member yang dideklarasikan di kelas A.



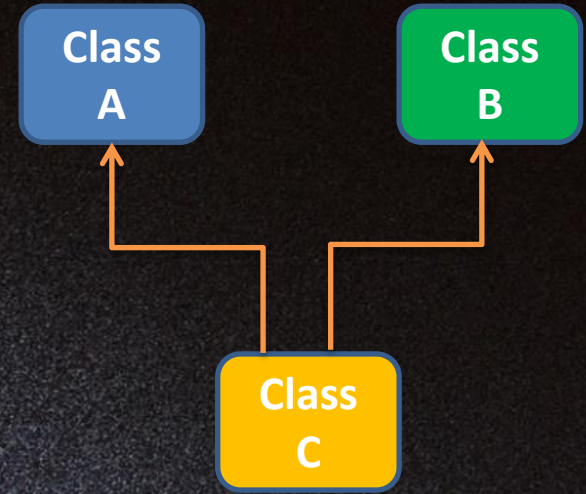
Pewarisan (Lanjutan)

- **Hierarchical Inheritance:** banyak memiliki sub kelas dari sebuah kelas induk. Bila kelas B, C, dan D inherit dari kelas A, maka kelas B, C, D akan mewariskan semua member yang dideklarasikan di kelas A.



Pewarisan (Lanjutan)

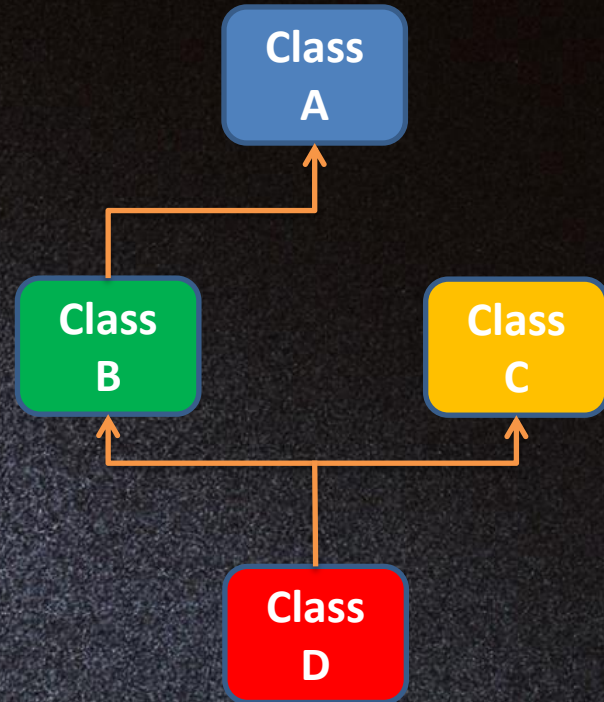
- **Multiple Inheritance:**
Jika kelas C inherits dari kelas A dan kelas B, maka kelas C akan acquire semua member yang dideklarasikan di kelas A dan kelas B. Kelas C mendapatkan semua member yang ada di kelas A dan kelas B.



*No Recommended
Di Java tidak bisa,

Pewarisan (Lanjutan)

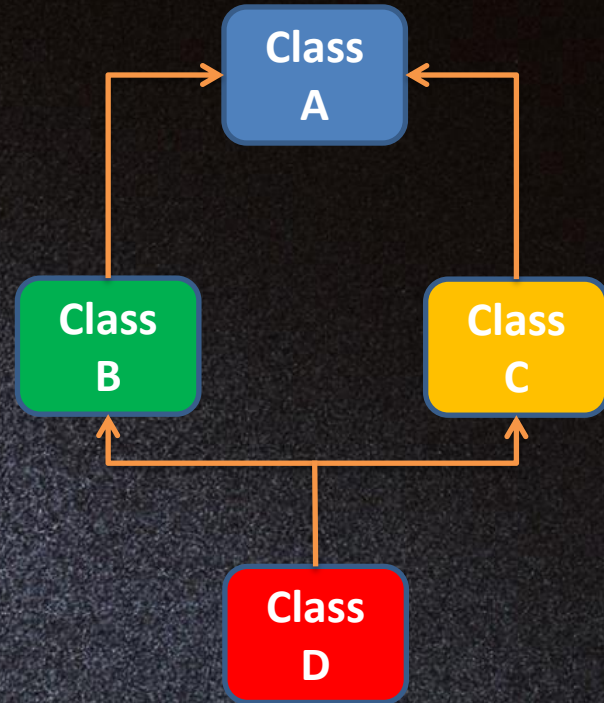
- **Hybrid Inheritance:** merupakan pewarisan yang menggabungkan beberapa jenis pewarisan.



*No Recommended

Pewarisan (Lanjutan)

- **Multipath inheritance:** merupakan pewarisan yang memiliki beberapa jalur pewarisan.



*No Recommended

Pewarisan (Lanjutan)

- **Catatan:** di dalam mengimplementasikan multiple, hybrid dan multipath harus didukung oleh bahasa pemrograman, bila tidak maka konsep ini tidak bisa dilakukan, dan sebaiknya menghindari jenis pewarisan tersebut (no recommended) karena mengandung ambiguitas.
- Dengan konsep pewarisan maka kita dapat mengimplementasikan konsep:
 - Overriding.

Pewarisan (Lanjutan)

- **Overriding:** kemampuan kelas turunan untuk memodifikasi atribut & method milik kelas induknya.
- Proses ini akan mengubah data dan method dari kedua class tersebut.
- Apabila atribut dan method dideklarasikan sebagai private atau final tidak dapat dilakukan overriding.
- Override method merupakan method yang sama persis dengan method yang sudah ada di super kelasnya, biasanya perbedaannya adalah pada implementasi (program body).
- Overriding tidak bisa dilakukan dalam kelas itu sendiri.
- Jadi Overriding erat kaitannya dengan inheritance (pewarisan).

Pewarisan (Lanjutan)

- Misalkan di kelas induk nama: "Wati", di kelas turunannya dengan override nama: "Dewi".
- **Overloading:** memungkinkan suatu kelas memiliki beberapa method dengan nama sama tetapi memiliki implementasi yang berbeda, sepanjang deklarasi dan parameternya berbeda disebut signaturenya berbeda.
- Hal ini dimungkinkan asalkan deklarasi method membuat penanda berbeda di satu kelas.



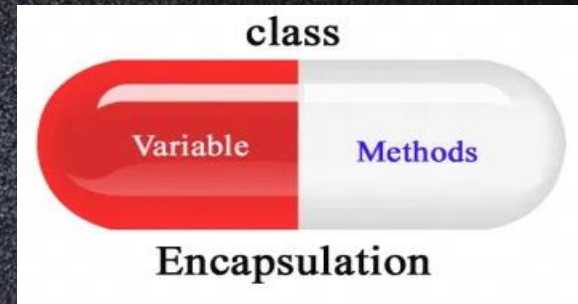
Pewarisan (Lanjutan)

- **Overloading:** memungkinkan suatu kelas memiliki beberapa method dengan nama sama tetapi memiliki implementasi atau argumen yang berbeda, sepanjang deklarasi dan parameternya berbeda disebut signaturenya berbeda.
- Hal ini dimungkinkan asalkan deklarasi method membuat penanda berbeda di satu kelas.

<u>return type</u>	<u>nama method</u>	<u>daftar parameter</u>
void	Coba	(int t1)
void	Coba	(int t1, int t2)
void	Coba	(int t1, int t2, int t3)
void	Coba	(int t1, int t2, int t3, int t4)
↓	↓	↓
sama	sama	berbeda

Pembungkusan

- **Pembungkusan:** adalah proses pemaketan data bersama method-nya, pembungkusan bermanfaat untuk menyembunyikan rincian-rincian implementasi dari pemakai.



Pembungkusan (Lanjutan)

- Pembungkusan dapat memberikan manfaat pada konsep pemrograman berorientasi objek, manfaat yang didapat dari pembungkusan ada 2 (dua) yaitu:
 - Penyembunyian informasi (*Information Hiding*).
 - Modularitas.
- Dalam mengimplementasikan konsep pembungkusan pada pemrograman berorientasi objek dapat diimplementasikan dengan cara:
 - Property method (getter dan setter).
 - Interface.

Pembungkusan (Lanjutan)

Diagram Kelas

Nama kelas	Kelas: Mahasiswa
Nama atribut	-nim : string -nama : string -uts : int -tugas : int -uas : int
Nama method	+total_nilai() : float +grade() : char



Instansiasi kelas
menjadi objek

Diagram Objek

Nama objek	Objek: Mahasiswa
Nama atribut	-nim: 202107001 -nama : Reny -uts : 90 -tugas : 90 -uas : 90
Nama method	+total_nilai: 90 +grade: A

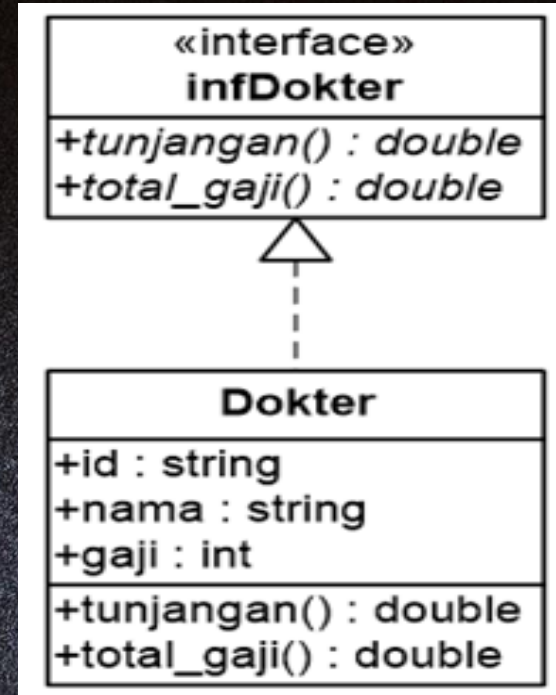
Pembungkusan (Lanjutan)

- Dengan property method, hal ini dikarenakan jangkauan nilai pada atribut adalah private.

Dokter
- id : String - nama : String - gaji : int + <<Property>>+id{read and write} : String + <<Property>>+nama{read and write} : String + <<Property>>+gaji{read and write} : int
+ tunjangan() : float + total_gaji() : float

Pembungkusan (Lanjutan)

- Dengan interface, kelas implementasi menyembunyikan proses sebenarnya dari fungsi yang ada dalam interface.



Banyak Bentuk

- Banyak bentuk dapat dimaknai sebagai modul yang memiliki nama sama, namun memiliki tingkah laku (behaviour) yang berbeda sehingga kode program implementasinya juga berbeda.
- Seperti kata “**bisa**”, dapat bermakna “racun” atau bermakna “mampu”.
- Banyak bentuk adalah: **pemakaian method dengan nama yang sama pada kelas-kelas yang berbeda dan memungkinkan method yang tepat dieksekusi berdasarkan konteks yang memangilnya.**
- Banyak bentuk memiliki arti "satu nama, banyak bentuk".

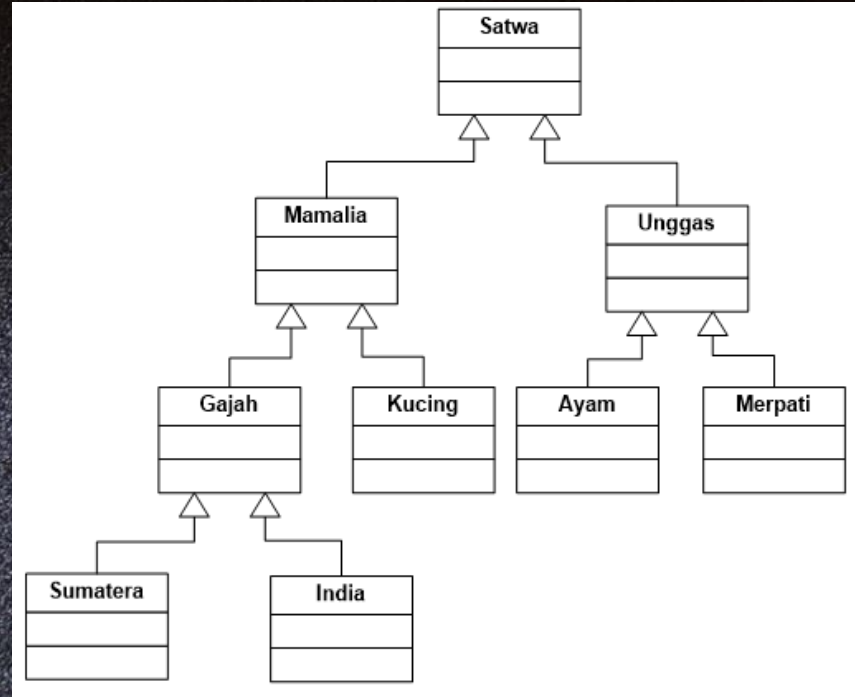
Banyak Bentuk (Lanjutan)



- Dalam mengimplementasikan banyak bentuk dapat:
 - berbasis pada **interface**.
 - berbasis **pewarisan** pada **pewarisan**.

Banyak Bentuk (Lanjutan)

- Semua objek yang memenuhi lebih dari satu relasi is-a maka objek tersebut dikatakan **polymorphic**.



Banyak Bentuk (Lanjutan)

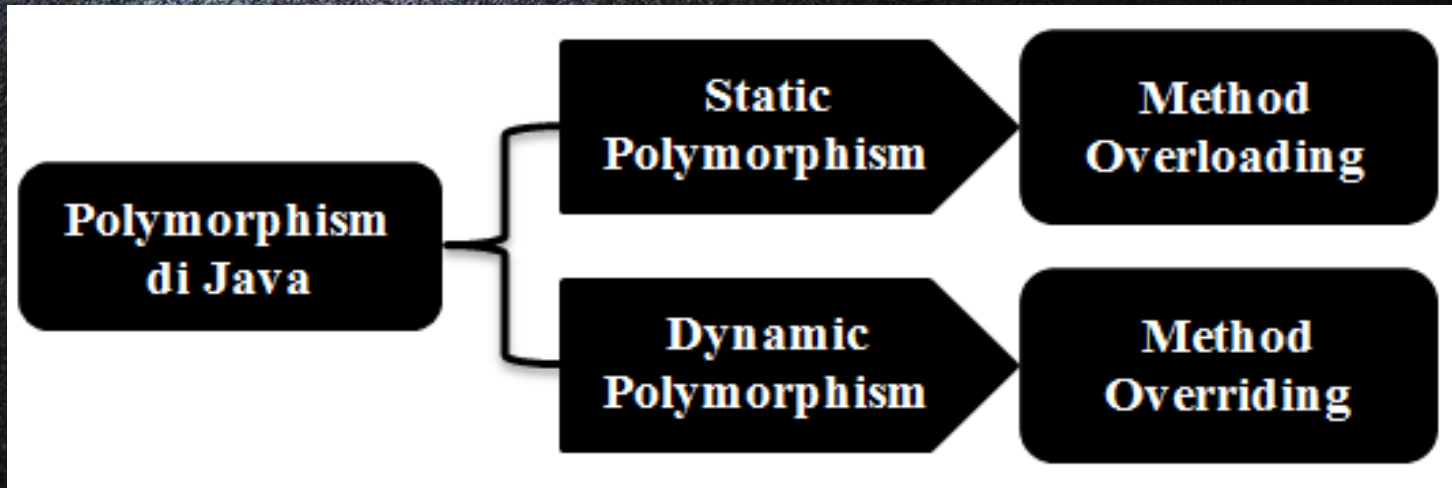
- Dalam mengimplementasikan polimorfisme harus memenuhi hal-hal berikut ini:
 - Method yang dipanggil harus melalui variabel dari basis kelas atau superkelas.
 - Method yang dipanggil juga harus menjadi method dari basis kelas.
 - Signature method harus sama, baik pada kelas induk maupun di kelas anak (subkelas).
 - Method access attribute pada subkelas tidak boleh lebih terbatas (sempit) dari basis kelas.

Banyak Bentuk (Lanjutan)

- Ada dua jenis polimorfisme di Java, yaitu:
 1. **Polimorfisme statis;** menggunakan method overloading, method yang sama hanya pada kelas yang sama.
 2. **Polymorphism dinamis;** menggunakan method overriding, method yang sama hadir pada kelas yang berbeda lewat pewarisan.

Banyak Bentuk (Lanjutan)

Bila diilustrasikan dalam gambar maka jenis polimorfisme seperti pada gambar



Banyak Bentuk (Lanjutan)

- Menerapkan polimorfisme statis dilakukan dengan cara:
 - Overloading method;
 - Overloading operator (di java tidak bisa melakukan overloading operator).

Pegawai
-id : string
-nama : string
-status : string
-gaji : int
+hitung() : double
+hitung() : double
+hitung() : double
+total() : double

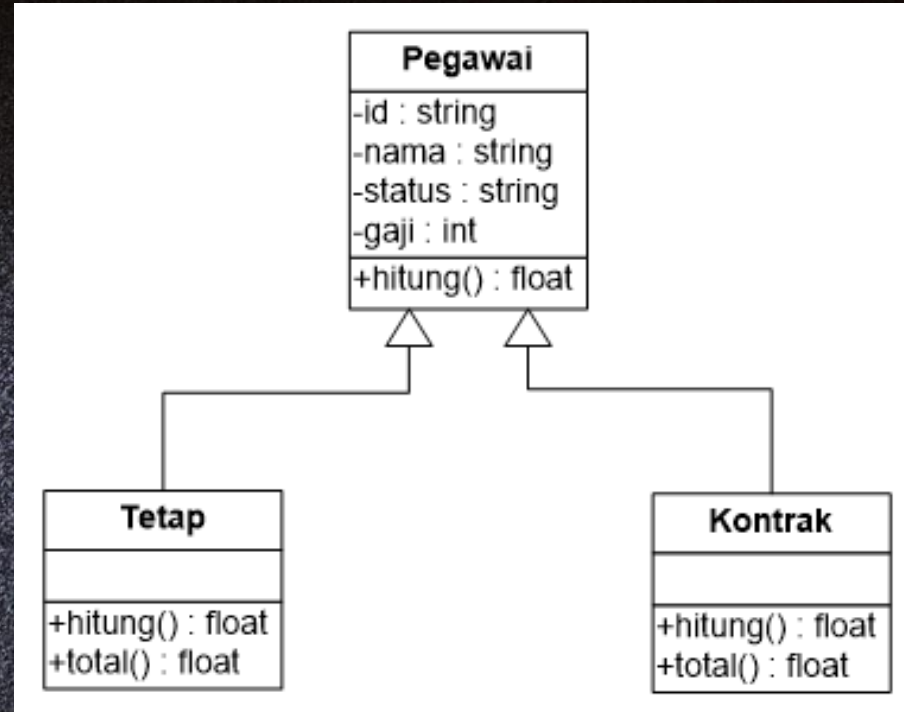
Overloading method

Banyak Bentuk (Lanjutan)

- Polimorfisme dinamis dikenal juga dengan sebutan run-time Polymorphism.
- Dalam mengimplementasikan polimorfisme dinamis menggunakan method overriding.
- Polimorfisme lebih sering diimplementasikan dengan **pewarisan**, **interface**, **kelas abstrak**, ada juga dengan **redefinisi method** namun jarang yang menggunakannya.

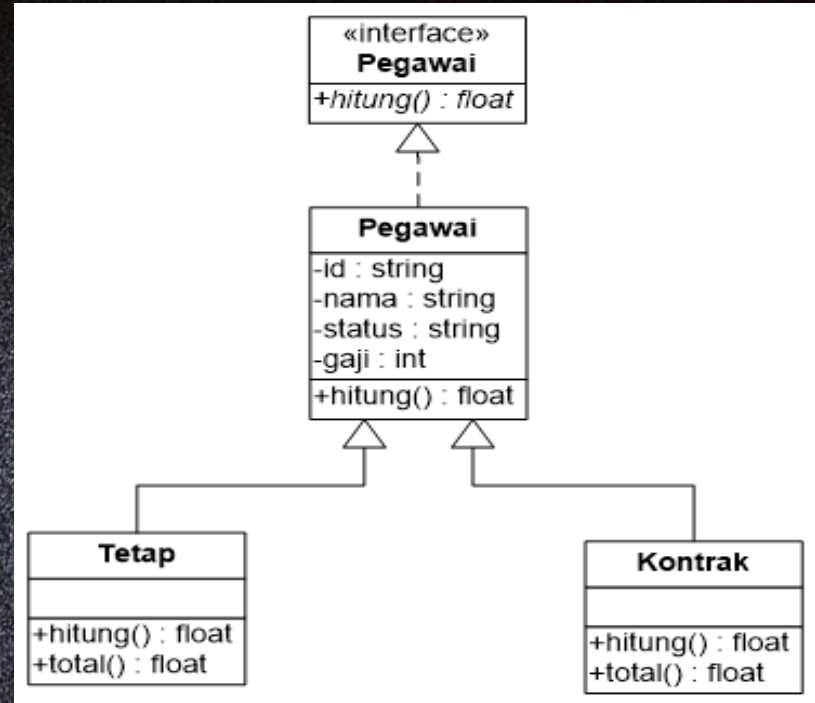
Banyak Bentuk (Lanjutan)

- Banyak bentuk dengan pewarisan melibatkan pendefinisian method di kelas induk dan mengoverride-nya dengan implementasi baru di kelas turunan.



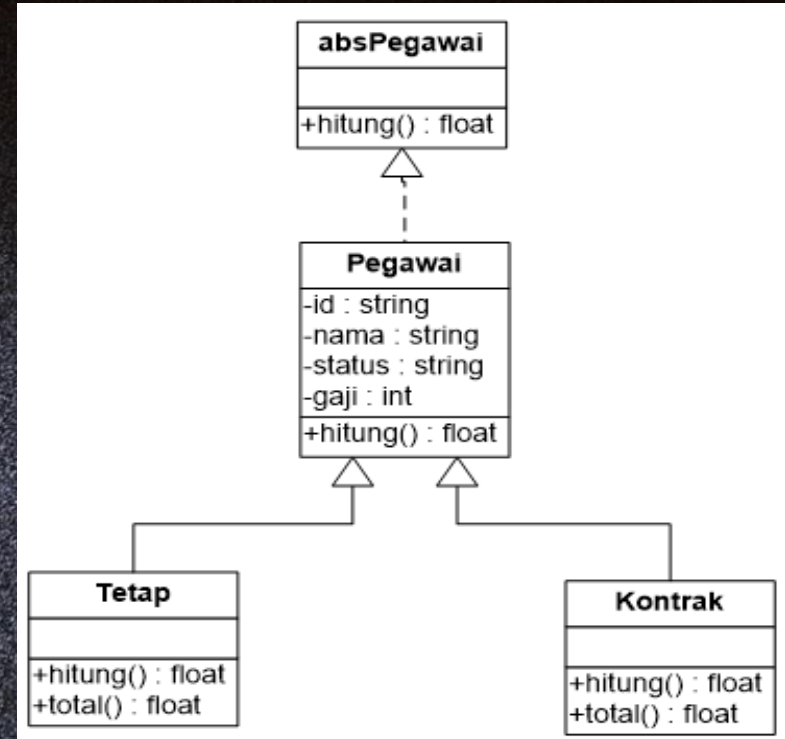
Banyak Bentuk (Lanjutan)

- Untuk implementasi banyak bentuk dengan interface, kita mengimplementasikan interface dengan cara berbeda di beberapa kelas.



Banyak Bentuk (Lanjutan)

- Banyak bentuk dengan kelas abstrak harus diwariskan di kelas konkret, yang merefer (mengacu) dari kelas abstrak.



Keuntungan PBO

- Keuntungan PBO adalah:
 - Natural.
 - Reliable.
 - Modularity.
 - Information-Hiding.
 - Maintainability.
 - Extensibility.
 - Reusability.
 - Pluggability & Debugging Ease.
 - Efisiensi.

Praktek dengan Java

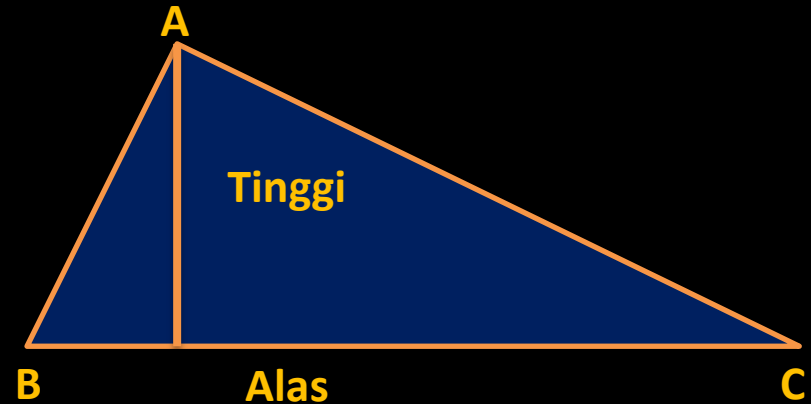


SEKIAN

Studi Kasus

- Pada studi kasus ini kita akan mengambil kasus objek segitiga.
 - Pendekatan prosedural
 - Pendekantan objek

Objek Segitiga



$$\text{Luas} = 0.5 * \text{Alas} * \text{Tinggi}$$

NON-OBJEK: Prosedural/Struktur

```
package luassegitiga;
import java.util.Scanner;
public class LuasSegitiga {

    //membuat fungsi dengan parameter
    static double LuasSegitiga(int mAlas, int mTinggi)
    {
        return 0.5 * mAlas * mTinggi;
    }

    public static void main(String[] args) {
        //mendeklarasikan variabel
        int Tinggi;
        int Alas;
        Double Luas;
```

```
        Scanner input = new Scanner (System.in);
        System.out.printf("Masukkan Alas : ");
        Alas = input.nextInt();
        System.out.printf("Masukkan Tinggi : ");
        Tinggi = input.nextInt();
        Luas = LuasSegitiga(Alas, Tinggi);
        System.out.println("Luas segitiga : "+ Luas);
    }
}
```

```
run:
Masukkan Alas : 10
Masukkan Tinggi : 20
Luas segitiga : 100.0
BUILD SUCCESSFUL (total time: 4 seconds)
```

1. ABSTRAKSI: Method dgn Prosedur

```
package latsegitiga;

public class clsSegitiga {
    int Alas;
    int Tinggi;

    //membuat method dengan prosedur
    void HitungLuas()
    {
        double Luas;
        Luas = 0.5 * Alas * Tinggi;
        System.out.printf("Luas Segitiga : %.2f\n", Luas);
    }
}
```

```
package latsegitiga;
import java.util.Scanner;
public class LatSegitiga {
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner input = new Scanner (System.in);
        //Instansiasi kelas menjadi objek
        clsSegitiga objSegitiga = new clsSegitiga();
        //masukkan nilai alas dan tinggi
        System.out.printf("Masukkan Alas : ");
        objSegitiga.Alas = input.nextInt();
        System.out.printf("Masukkan Tinggi : ");
        objSegitiga.Tinggi = input.nextInt();
        objSegitiga.HitungLuas();
    }
}
```


ABSTRAKSI: Method dgn Fungsi

```
package luassegitiga;
import java.util.Scanner;

//menciptakan kelas
public class clsSegitiga
{
    //membuat atribut, jangkauan nilai public
    int Alas;
    int Tinggi;
    //membuat method dengan fungsi tanpa parameter
    public double Luas()
    {
        return 0.5 * Alas * Tinggi;
    }
}
```

```
package luassegitiga;
import java.util.Scanner;
public class LuasSegitiga {
    public static void main(String[] args) {
        Scanner input = new Scanner (System.in);
        //membuat objek dari clsSegitiga
        clsSegitiga objSegitiga = new clsSegitiga();
        //masukkan nilai alas dan tinggi
        System.out.printf("Masukkan Alas : ");
        objSegitiga.Alas = input.nextInt();
        System.out.printf("Masukkan Tinggi : ");
        objSegitiga.Tinggi = input.nextInt();
        System.out.printf("Luas    segitiga    :    %.2f\n",
objSegitiga.Luas());
    }
}
```

ABSTRAKSI: Method dgn Fungsi

```
package luassegitiga;
import java.util.Scanner;
//menciptakan kelas
public class clsSegitiga
{
    //membuat atribut dgn jangkauan nilai public
    int Alas;
    int Tinggi;

    //membuat method dgn fungsi menggunakan parameter
    public double Luas(int mAlas, int mTinggi)
    {
        return 0.5 * mAlas * mTinggi;
    }
}
```

```
package luassegitiga;
import java.util.Scanner;
public class LuasSegitiga {

    public static void main(String[] args) {
        Scanner input = new Scanner (System.in);
        //membuat objek dari clsSegitiga
        clsSegitiga objSegitiga = new clsSegitiga();
        //masukkan nilai alas dan tinggi
        System.out.printf("Masukkan Alas : ");
        objSegitiga.Alas = input.nextInt();
        System.out.printf("Masukkan Tinggi : ");
        objSegitiga.Tinggi = input.nextInt();
        //Cetak luas segitiga
        System.out.printf("Luas      segitiga      :      %.2f\n",
objSegitiga.Luas(objSegitiga.Alas, objSegitiga.Tinggi));
    }
}
```


ABSTRAKSI: Property Method

```
package latsegitiga;

public class clsSegitiga {
    private int Alas;
    private int Tinggi;

    //property method alas
    public void setAlas(int newValue)
    {
        Alas = newValue;
    }
    public int getAlas()
    {
        return Alas;
    }
}
```

```
//property method tinggi
public void setTinggi(int newValue)
{
    Tinggi = newValue;
}
public int getTinggi()
{
    return Tinggi;
}

//membuat method tanpa parameter
public double Luas()
{
    return 0.5 * getAlas() * getTinggi();
}
}
```

ABSTRAKSI: Property Method

```
package latsegitiga;
import java.util.Scanner;

public class LatSegitiga {
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner input = new Scanner (System.in);
        //Instansiasi kelas menjadi objek
        clsSegitiga objSegitiga = new clsSegitiga();

        //masukkan nilai alas dan tinggi
        System.out.printf("Masukkan Alas : ");
        objSegitiga.setAlas(input.nextInt());
        System.out.printf("Masukkan Tinggi : ");
        objSegitiga.setTinggi(input.nextInt());
```

```
        System.out.printf("Luas      Segitiga      :      %.2f\n",
        objSegitiga.Luas());
    }
}
```

```
run:
Masukkan Alas : 4
Masukkan Tinggi : 4
Luas Segitiga : 8.00
BUILD SUCCESSFUL (total time: 6 seconds)
```


ABSTRAKSI: Property Method

```
package latsegitiga;

public class clsSegitiga {
    private int Alas;
    private int Tinggi;

    //property method alas
    public void setAlas(int newValue)
    {
        Alas = newValue;
    }
    public int getAlas()
    {
        return Alas;
    }
}
```

```
//property method tinggi
public void setTinggi(int newValue)
{
    Tinggi = newValue;
}
public int getTinggi()
{
    return Tinggi;
}

//membuat method dgn parameter
public double Luas(int mAlas, int mTinggi)
{
    return 0.5 * getAlas() * getTinggi();
}
}
```

ABSTRAKSI: Property Method

```
package latsegitiga;
import java.util.Scanner;

public class LatSegitiga {
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner input = new Scanner (System.in);
        //Instansiasi kelas menjadi objek
        clsSegitiga objSegitiga = new clsSegitiga();

        //masukkan nilai alas dan tinggi
        System.out.printf("Masukkan Alas : ");
        objSegitiga.setAlas(input.nextInt());
        System.out.printf("Masukkan Tinggi : ");
        objSegitiga.setTinggi(input.nextInt());
```

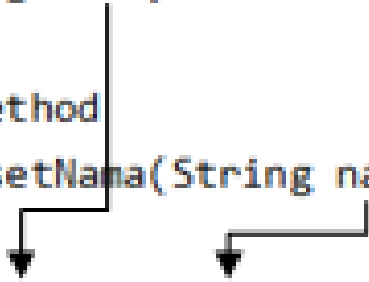
```
        System.out.printf("Luas      Segitiga      :      %.2f\n",
        objSegitiga.Luas(objSegitiga.getAlas(),
        objSegitiga.getTinggi()));
    }
}
```

```
run:
Masukkan Alas : 4
Masukkan Tinggi : 4
Luas Segitiga : 8.00
BUILD SUCCESSFUL (total time: 6 seconds)
```


ABSTRAKSI: Keyword this

Nama kelas: Dokter

```
1.  public class Dokter {  
2.      //deklarasi atribut  
3.      private String nama;  
4.  
5.      //property method  
6.      public void setNama(String nama)  
7.      {  
8.          this.nama = nama;  
9.      }  
...
```



ABSTRAKSI: Keyword super

Jika mendeklarasikan member (atribut atau method) dari subkelas dengan nama yang sama dengan yang dimiliki superkelas, maka hanya dapat mengakses member superkelas tersebut dengan menggunakan keyword super.

Keyword super menunjukkan bahwa kita ingin merefer superkelas dari kelas yang bersangkutan, seperti pada gambar.

ABSTRAKSI: Keyword super

```
class Induk
{
    String nama;
}

class Anak extends Induk {
    String nama;

    void detail()
    {
        super.nama = "Induk";
        nama = "Anak";
    }
}
```

The diagram illustrates the relationship between the `Induk` and `Anak` classes. A vertical line connects the `Induk` class definition to the `extends Induk` part of the `Anak` class definition. From this connection point, a horizontal line extends to the right, and then a vertical line goes down to the `super.nama = "Induk";` line inside the `detail()` method of the `Anak` class. Finally, a horizontal line goes left from there to an arrow pointing at the `String nama;` line in the `Induk` class, indicating that the `super` keyword refers to the superclass's attribute.

ABSTRAKSI: Overloading

Nama kelas: Hitung

```
1. package lat_hitung;
2. public class Hitung {
3.     //membuat method overloading
4.     public double hasil(int a, int b)
5.     {
6.         return a*b;
7.     }
8.
9.     public double hasil(double a, int b, int c)
10.    {
11.        return a+b*c;
12.    }
13.
14.    public double hasil(int a, double b, int c, int d)
15.    {
16.        return a+b+c+d;
17.    }
18. }
```


ABSTRAKSI: Overloading

Main program: Hitung

```
1. package lat_hitung;
2. public class Lat_Hitung {
3.     public static void main(String[] args) {
4.         // TODO code application logic here
5.         //instansiasi kelas menjadi objek
6.         Hitung objHitung = new Hitung();
7.
8.         System.out.println("Hasil a+b="+
9.             objHitung.hasil(10, 5));
10.        System.out.println("Hasil a+b*c="+
11.            objHitung.hasil(10, 5, 2));
12.        System.out.println("Hasil a+b+c+d="+
13.            objHitung.hasil(10, 5, 2, 4));
14.    }
15. }
```

Report Problems Window

iReport output

Notificati



run:

Hasil a+b=50.0

Hasil a+b*c=20.0

Hasil a+b+c+d=21.0

BUILD SUCCESSFUL (total time: 1 second)

ABSTRAKSI: Overriding

Nama kelas: GajiLama

```
1. package lat_overriding;
2. public class GajiLama {
3.     //method tunjangan dengan gaji lama
4.     public void tunjangan()
5.     {
6.         int gaji = 10000000;
7.         System.out.println("Tunjangan lama : "+(gaji/100)*10);
8.     }
9. }
```


ABSTRAKSI: Overriding





Nama kelas: GajiBaru

```
1.  package lat_overriding;
2.
3.  public class GajiBaru extends GajiLama{
4.      //method tunjangan dengan gaji baru
5.      @Override
6.      public void tunjangan()
7.      {
8.          int gaji = 15000000;
9.          System.out.println("Tunjangan lama : "+(gaji/100)*10);
10.     }
11. }
```

ABSTRAKSI: Overriding

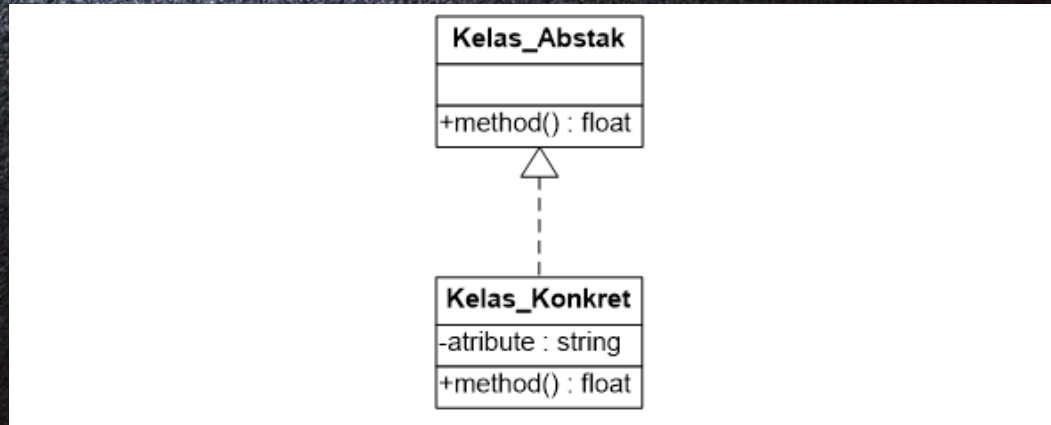
Main program: Lat_Overriding

```
1. package lat_overriding;
2. public class Lat_Overriding {
3.     public static void main(String[] args) {
4.         // TODO code application logic here
5.         //instansiasi kelas jadi objek
6.         GajiLama objLama = new GajiLama();
7.         objLama.tunjangan();
8.         //instansiasi kelas jadi objek
9.         GajiBaru objBaru = new GajiBaru();
10.        objBaru.tunjangan();
11.    }
12. }
```

Report Problems Window	iReport output	Notifications
	run:	
	Tunjangan lama : 1000000	
	Tunjangan lama : 1500000	
	BUILD SUCCESSFUL (total time: 1 second)	

ABSTRAKSI: Kelas Abstrak

Apakah yang dimaksud dengan kelas abstrak, kelas abstrak adalah: **kelas yang tidak dapat dibuat instannya.**



ABSTRAKSI: Kelas Abstrak

Kelas Abstrak: kelasAbstrak

```
package latkelasabstrak;  
  
public abstract class kelasAbstrak  
{  
    public abstract void Tunjangan();  
    public abstract void Bonus();  
}
```


ABSTRAKSI: Kelas Abstrak

Kelas: clsDosen

```
package latkelasabstrak;
public class clsDosen extends kelasAbstrak
{
    @Override
    public void Tunjangan()
    {
        System.out.println("Besar tunjangan: 1000000");
    }

    @Override
    public void Bonus() {
        System.out.println("Bonus: 10000000");
    }
}
```

ABSTRAKSI: Kelas Abstrak

Main Program

```
package latkelasabstrak;

public class LatKelasAbstrak {
    public static void main(String[] args)
    {
        // TODO code application logic here
        clsDosen objDosen = new clsDosen();

        objDosen.Tunjangan();
        objDosen.Bonus();
    }
}
```


ABSTRAKSI: Kelas Abstrak

Report Problems Window	iReport output	Output - LatK
	run:	
	Besar tunjangan: 1000000	
	Bonus: 10000000	
	BUILD SUCCESSFUL (total time: 0 seconds)	

ABSTRAKSI: Interface

- Interface merupakan kumpulan method-method abstrak.
- Sebuah kelas yang mengimplementasikan interface, mewarisi method-method abstrak dari interface tersebut.
- Persamaan interface dengan kelas:
 - ✓ Interface dapat memiliki banyak method.
 - ✓ Interface ditulis dalam file dengan ekstensi .java.

ABSTRAKSI: Interface

- Interface merupakan kumpulan method-method abstrak.
- Sebuah kelas yang mengimplementasikan interface, mewarisi method-method abstrak dari interface tersebut.

ABSTRAKSI: Interface

Interface: intDosen

```
package latihaninterface;
```

```
public interface intDosen
```

```
{
```

```
    void Tunjangan();
```

```
    void Bonus();
```

```
}
```


ABSTRAKSI: Interface

Kelas: clsDosen

```
package latihaninterface;
public class clsDosen implements intDosen
{
    @Override
    public void Tunjangan()
    {
        System.out.println("Tunjangan: 1000000");
    }
    @Override
    public void Bonus()
    {
        System.out.println("Bonus: 9000000");
    }
}
```

ABSTRAKSI: Interface

Min program

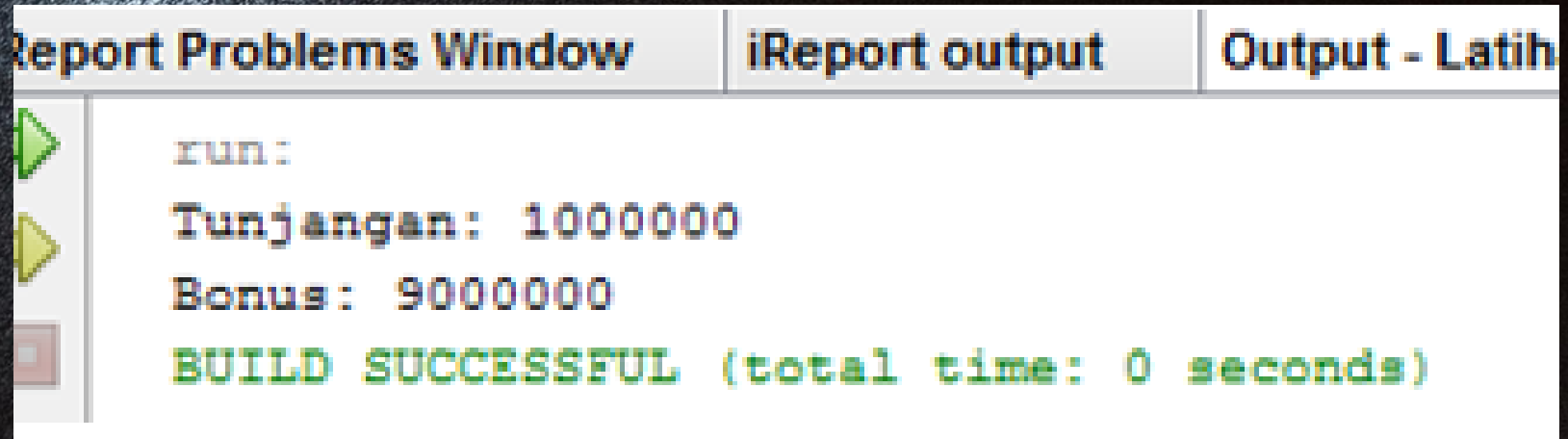
```
package latihaninterface;

public class LatihanInterface {

    public static void main(String[] args)
    {
        // TODO code application logic here
        clsDosen objDosen = new clsDosen();

        objDosen.Tunjangan();
        objDosen.Bonus();
    }
}
```


ABSTRAKSI: Interface



ABSTRAKSI: Interface

- Sebuah interface dapat meng-extends interface lainnya.
 - Misal ada sebuah interface A.
 - Interface A di extend oleh interface B.
 - Interface B baru di implement di kelas C.
 - Kelas C di instansiasi di main prorgam.

2. INHERITANCE

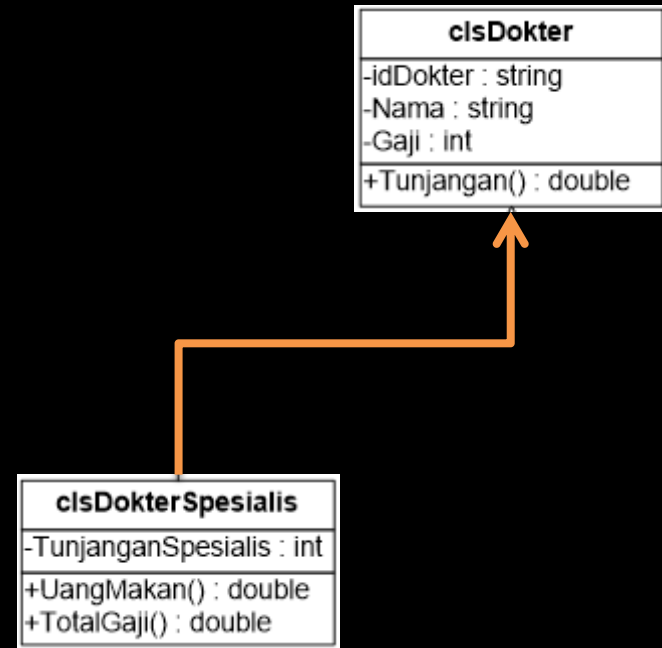
- Pada studi kasus ini kita akan mengambil kasus objek dokter, menerapkan inheritance.
 - Single Inheritance
 - Multilevel Inheritance
 - Hierarchical Inheritance



Dokter

SUB BAHASAN: Single Inheritance

- Pada studi kasus ini kita akan mengambil kasus objek dokter, menerapkan:
 - Single Inheritance



SUB BAHASAN: Single Inheritance

```
package dokter;
```

```
public class clsDokter
```

```
{
```

```
    String IdDokter;
```

```
    String Nama;
```

```
    int Gaji;
```

```
    public double Tunjangan (int mGaji)
```

```
    {
```

```
        return mGaji/100 * 10;
```

```
    }
```

```
}
```

```
package dokter;
```

```
public class clsDokterSpesialis extends clsDokter
```

```
{
```

```
    int TunjanganSpesialis;
```

```
    public double UangMakan(int mGaji)
```

```
    {
```

```
        return mGaji/100 * 10;
```

```
    }
```

```
    public double TotalGaji(int mGaji, double mTunjangan,  
double mTunSpesialis, double mUangMakan)
```

```
    {
```

```
        return mGaji + mTunjangan + mTunSpesialis +  
mUangMakan;
```

```
    }
```

```
}
```

SUB BAHASAN: Single Inheritance

```
package dokter;
import java.util.Scanner;
public class Dokter {

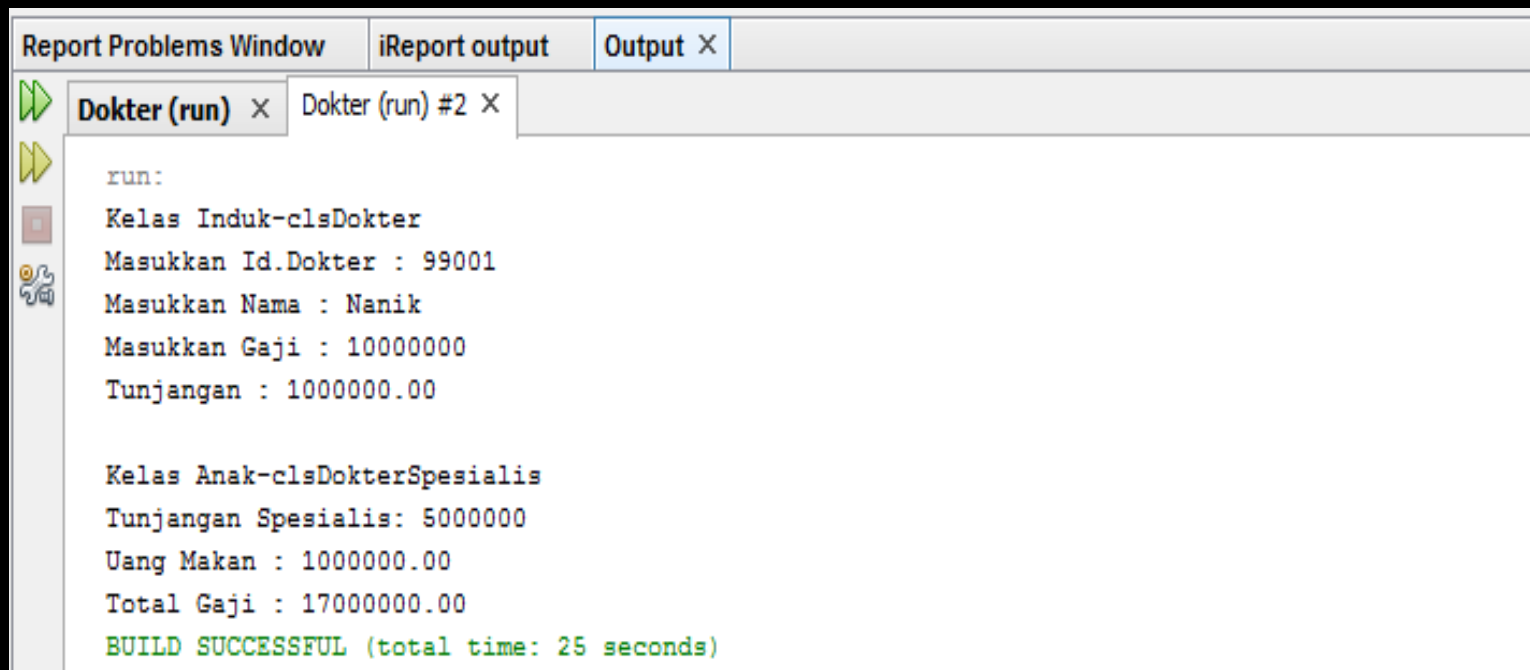
    public static void main(String[] args)
    {
        double Tunjangan;
        double UangMakan;
        Scanner input = new Scanner (System.in);

        // Membuat objek
        clsDokterSpesialis objDokSpesialis = new clsDokterSpesialis();

        System.out.println("Kelas Induk-clsDokter");
        System.out.printf("Masukkan Id.Dokter : ");
        objDokSpesialis.IdDokter = input.next();
        System.out.printf("Masukkan Nama : ");
```

```
        objDokSpesialis>Nama = input.next();
        System.out.printf("Masukkan Gaji : ");
        objDokSpesialis.Gaji = input.nextInt();
        Tunjangan = objDokSpesialis.Tunjangan(objDokSpesialis.Gaji);
        System.out.printf("Tunjangan : %.2f\n", Tunjangan);
        System.out.println();
        System.out.println("Kelas Anak-clsDokterSpesialis");
        System.out.printf("Tunjangan Spesialis: ");
        objDokSpesialis.TunjanganSpesialis = input.nextInt();
        UangMakan =
        objDokSpesialis.UangMakan(objDokSpesialis.Gaji);
        System.out.printf("Uang Makan : %.2f\n", UangMakan);
        System.out.printf("Total      Gaji      :      %.2f\n",
        objDokSpesialis.TotalGaji(objDokSpesialis.Gaji,
        Tunjangan,objDokSpesialis.TunjanganSpesialis ,UangMakan ));
    }
}
```


SUB BAHASAN: Single Inheritance



```
run:
Kelas Induk-clsDokter
Masukkan Id.Dokter : 99001
Masukkan Nama : Nanik
Masukkan Gaji : 10000000
Tunjangan : 1000000.00

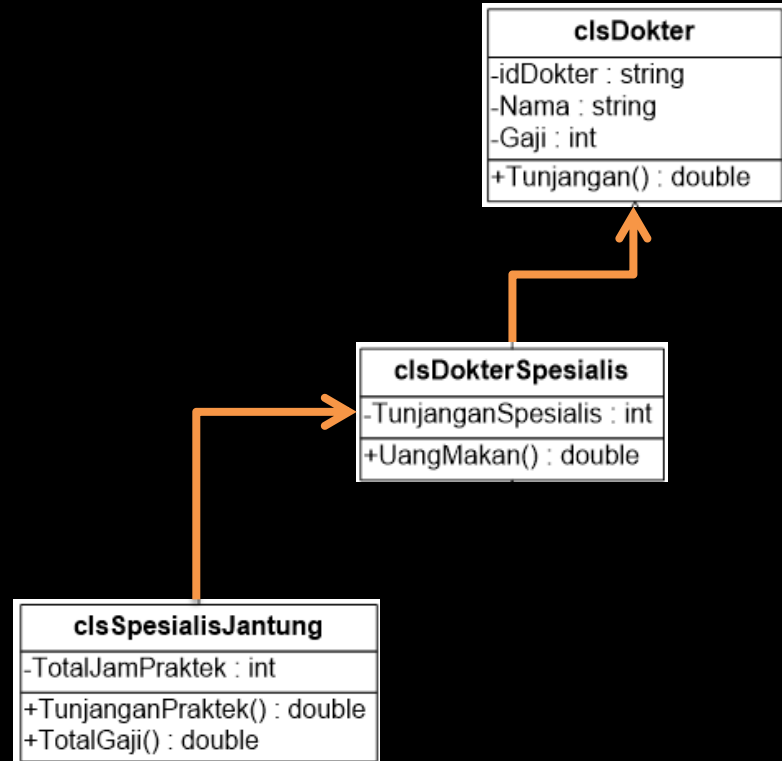
Kelas Anak-clsDokterSpesialis
Tunjangan Spesialis: 5000000
Uang Makan : 1000000.00
Total Gaji : 17000000.00
BUILD SUCCESSFUL (total time: 25 seconds)
```

SUB BAHASAN: Multilevel Inheritance

- Pada studi kasus ini kita akan mengambil kasus objek dokter, menerapkan:
 - Multilevel Inheritance

Silahkan
programnya

kerjakan

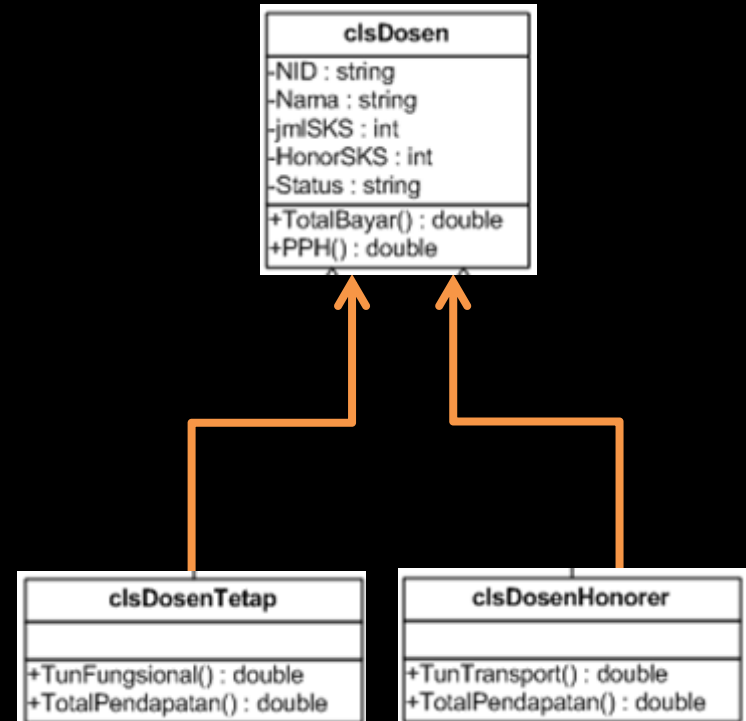


SUB BAHASAN: Hierarchical Inheritance

- Pada studi kasus ini kita akan mengambil kasus objek dokter, menerapkan:
 - Hierarchical Inheritance

Silahkan
programnya

kerjakan



3. ENCAPSULATION

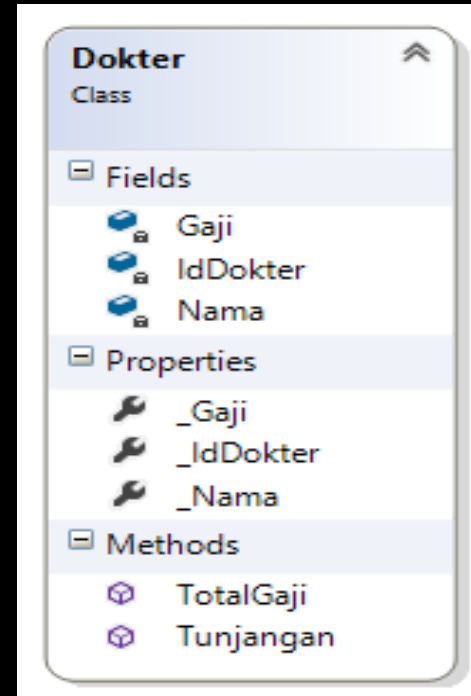
- Kita menggunakan kasus objek dokter.
- Dalam menerapkan encapsulation dapat menggunakan:
 - Property method
 - Interface



Dokter

SUB BAHASAN: Encapsulation – Property Method

- Pada studi kasus ini kita akan mengambil kasus objek dokter, menerapkan pembungkusan dengan:
 - Property method



SUB BAHASAN: Encapsulation – Property Method

```
package dokter;

public class clsDokter
{
    private String IdDokter;
    private String Nama;
    private int Gaji;

    public void setIdDokter(String newValue)
    {
        IdDokter = newValue;
    }

    public String getIdDokter()
    {
        return IdDokter;
    }
}
```

```
    public void setName(String newValue)
    {
        Nama = newValue;
    }

    public String getName()
    {
        return Nama;
    }

    public void setGaji(int newValue)
    {
        Gaji = newValue;
    }
}
```


SUB BAHASAN: Encapsulation – Property Method

```
public int getGaji()
{
    return Gaji;
}

public float Tunjangan()
{
    return Gaji/100*10;
}

public float TotalGaji()
{
    return Gaji+Tunjangan();
}
}
```

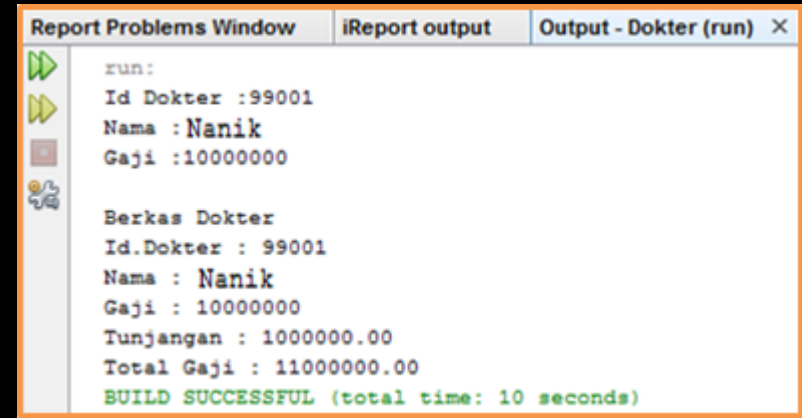
Bagian ini dibuat di program utama

```
package dokter;
import java.util.Scanner;
public class Dokter
{
    public static void main(String[] args)
    {
        // TODO code application logic here
        Scanner input = new Scanner(System.in);
        clsDokter objDokter = new clsDokter();

        System.out.print("Id Dokter :");
        objDokter.setIdDokter(input.nextLine());
        System.out.print("Nama :");
        objDokter.setNama(input.nextLine());
        System.out.print("Gaji :");
        objDokter.setGaji (input.nextInt());
    }
}
```

SUB BAHASAN: Encapsulation – Property Method

```
System.out.println();
    System.out.println("Berkas Dokter");
    System.out.println("Id.Dokter      :      "+"
objDokter.getIdDokter());
    System.out.println("Nama : " + objDokter.getNama());
    System.out.println("Gaji : " + objDokter.getGaji());
    System.out.printf("Tunjangan      :      %.2f\n",
objDokter.Tunjangan());
    System.out.printf("Total      Gaji      :      %.2f\n",
objDokter.TotalGaji());
    }
}
```



```
run:
Id Dokter :99001
Nama :Nanik
Gaji :10000000

Berkas Dokter
Id.Dokter : 99001
Nama : Nanik
Gaji : 10000000
Tunjangan : 1000000.00
Total Gaji : 11000000.00
BUILD SUCCESSFUL (total time: 10 seconds)
```


SUB BAHASAN: Encapsulation – Interface

- Pada studi kasus ini kita akan mengambil kasus objek dokter, menerapkan pembungkusan dengan:

– Interface

Silahkan dicoba secara mandiri



4. POLYMORPHISM

- Pada studi kasus ini kita akan mengambil banyak bentuk dengan, menerapkan:
 - Pewarisan
 - Interface



Segiempat dan segitiga sama-sama menggunakan perintah hitung untuk mendapatkan hasil luas. Maka hitung adalah banyak bentuk, karena digunakan di segiempat dan segitiga.

Maka kita membuat method hitung yang dapat digunakan di segiempat dan segitiga.

SUB BAHASAN: Polymorphism dgn Inheritance

```
package luasbangun;

public class clsBangun
{
    public double hitung(int x,int y)
    {
        return (0);
    }
}
```

```
package luasbangun;

public class clsSegiEmpat extends clsBangun
{

    public double hitung (int x, int y)
    {
        return( x*y);
    }
}
```

SUB BAHASAN: Polymorphism dgn Inheritance

```
package luasbangun;

public class clsSegiTiga extends clsBangun
{
    public double hitung (int x, int y)
    {
        return (0.5*x*y);
    }
}
```

```
package luasbangun;
import java.util.Scanner;

public class LuasBangun {

    public static void main(String[] args)
    {
        // TODO code application logic here
        Scanner input = new Scanner (System.in);

        clsBangun objBangun = new clsBangun();
        clsSegiEmpat objSegiEmpat = new clsSegiEmpat();
        clsSegiTiga objSegiTiga = new clsSegiTiga();

        int m,n;
```


SUB BAHASAN: Polymorphism dgn Inheritance

```
System.out.printf("Masukkan Nilai X: ");
```

```
m = input.nextInt();
```

```
System.out.printf("Masukkan Nilai Y: ");
```

```
n = input.nextInt();
```

```
objBangun.hitung(m, n);
```

```
System.out.println("Luas Bangun: ");
```

```
objBangun = objSegiEmpat;
```

```
System.out.println("Segi empat: "+objBangun.hitung(m, n));
```

```
objBangun = objSegiTiga;
```

```
System.out.println("Segi tiga: "+objBangun.hitung(m, n));
```

```
}
```

```
}
```

```
run:
```

```
Masukkan Nilai X: 3
```

```
Masukkan Nilai Y: 4
```

```
Luas bangun:
```

```
Segi empat: 12
```

```
Segi tiga: 6.0
```

```
BUILD SUCCESSFUL (total time: 7 seconds)
```

SUB BAHASAN: Polymorphism dgn Interface

- Polymorphism dengan interface:

- Silahkan dicoba secara mandiri

Sekian

Profile



Nama : I Gusti Ngurah Suryantara,S.Kom.,M.Kom

Dosen : Informatika – UBM

Pengampu : PBO, Analisis & Desain
Berorientasi objek,
Augmented Reality,
Pemrograman Game,
Pemrograman Multimedia,
RPL, Skripsi.

Penulis : Buku Komputer diterbitkan oleh Gramedia

Channel YouTube : I Gusti Ngurah Suryantara TV

IG : I Gusti Ngurah Suryantara

FB : Gusti Ngurah Suryantara