MAKALAH

OVERLOADING DALAM BAHASA JAVA



Oleh :

Nasywa Deby Azanna

NIM 2022903430048

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEMAWE

2022/2023

# DAFTAR ISI

[DAFTAR ISI i](#_Toc136194624)

[BAB I PENDAHULUAN 1](#_Toc136194625)

[A. LATAR BELAKANG 1](#_Toc136194626)

[B. RUMUSAN MASALAH 3](#_Toc136194627)

[C. TUJUAN 3](#_Toc136194628)

[BAB II OVERLOADING 4](#_Toc136194629)

[A. PENGERTIAN OVERLOADING 4](#_Toc136194630)

[B. OVERLOADING METHOD 4](#_Toc136194631)

[C. OVERLOADING CONSTRUCTORS 7](#_Toc136194632)

[D. CONTOH PROGRAM OVERLOADING PADA JAVA 14](#_Toc136194633)

[E. PERBEDAAN OVERLOADING DAN OVERIDDING PADA JAVA 17](#_Toc136194634)

[BAB III PENENTU AKSES 20](#_Toc136194635)

[A. PENENTU AKSES 20](#_Toc136194636)

[B. PENENTU AKSES PRIVAT 21](#_Toc136194637)

[C. PENENTU AKSES PUBLIK 26](#_Toc136194638)

[D. PENENTU AKSES PROTECTED 32](#_Toc136194639)

[E. PENENTU AKSES DEFAULT 34](#_Toc136194640)

[BAB IV PENUTUP 36](#_Toc136194641)

[A. KESIMPULAN 36](#_Toc136194642)

# BAB I PENDAHULUAN

## LATAR BELAKANG

**Overloading**

Makalah ini membahas konsep overloading dalam bahasa pemrograman Java. Overloading adalah kemampuan untuk mendefinisikan beberapa metode dengan nama yang sama tetapi dengan parameter yang berbeda. Ini memungkinkan programmer untuk membuat kode yang lebih bersih, lebih fleksibel, dan lebih mudah dipahami. Makalah ini akan menjelaskan konsep overloading, aturan-aturan yang terkait, dan memberikan contoh-contoh implementasi overloading dalam bahasa pemrograman Java.

Overloading adalah deklarasi method maupun konstruktor lebih dari satu kali pada kelas yang sama. Overloading adalah salah satu cara penerapan dalam konsep polimorfisme.Jika kita tidak pernah pernah mengenal sebelumnya konsep ini kelihatan aneh, Tetapi metode inisebetulnya adalah hal yang sangat bermanfaat.

Overloading ada dua jenis, yaitu:

1. Overloading method
2. Overloading constructor

**Penentu Akses**

Makalah ini membahas tentang penentu akses dalam bahasa pemrograman Java. Penentu akses adalah mekanisme yang digunakan untuk mengatur tingkat aksesibilitas terhadap kelas, metode, dan variabel dalam suatu program Java. Pengetahuan tentang penentu akses sangat penting dalam pengembangan perangkat lunak berbasis Java, karena mereka memungkinkan pengembang untuk mengatur visibilitas dan perlindungan terhadap komponen-komponen program.

Makalah ini akan menjelaskan konsep penentu akses dalam Java, termasuk kata kunci yang digunakan untuk mengatur tingkat aksesibilitas, yaitu: public, private, protected, dan default. Kami akan menguraikan penggunaan dan batasan masing-masing penentu akses, serta memberikan contoh nyata untuk membantu pemahaman.

Selain itu, kami akan membahas pentingnya penentu akses dalam konteks keamanan dan pemeliharaan kode. Penggunaan yang tepat dari penentu akses dapat membantu mencegah akses yang tidak sah ke komponen-komponen penting, serta memfasilitasi perubahan dan evolusi kode dengan membatasi dampak perubahan pada bagian-bagian program yang tersembunyi.

Dalam makalah ini, kami juga akan membahas praktik terbaik dalam menggunakan penentu akses, termasuk pedoman desain dan penggunaan penentu akses yang konsisten dan logis dalam pengembangan perangkat lunak. Kami akan menyoroti beberapa kasus penggunaan khusus, seperti kapan menggunakan penentu akses protected atau default.

Makalah ini ditujukan untuk para pengembang perangkat lunak Java, baik yang baru memulai maupun yang berpengalaman. Diharapkan makalah ini dapat memberikan pemahaman yang kuat tentang konsep penentu akses dalam Java dan memberikan wawasan yang berharga dalam pengembangan perangkat lunak yang aman, mudah dipelihara, dan efisien.

Kata Kunci: Java, penentu akses, public, private, protected, default, pengembangan perangkat lunak.

Pendahuluan: Bahasa pemrograman Java menyediakan penentu akses sebagai mekanisme yang mengatur tingkat aksesibilitas terhadap komponen-komponen program seperti kelas, metode, dan variabel. Penentu akses memungkinkan pengembang untuk membatasi visibilitas dan perlindungan terhadap komponen-komponen ini, yang penting untuk menjaga integritas kode dan mengatur bagaimana komponen-komponen tersebut dapat diakses dan digunakan.

## RUMUSAN MASALAH

Dari uraian latar belakang diatas, penulis merumuskan beberapa masalah yang akan dijelaskan melalui makalah ini, yaitu sebagai berikut:

1. Bagaimana konsep pemrograman overloading dalam java.
2. Bagaimana konsep pemrograman penentu akses dalam java.
3. Bagaimana prinsip-prinsip dasar pemrograman overloading dalam Bahasa java.
4. Bagaimana prinsip-prinsip dasar pemrograman penentu akses dalam Bahasa java.

## TUJUAN

Dengan pembuatan makalah ini bertujuan agar pembaca dan penulis dapat:

1. Memahami konsep pemrograman overloading dalam Bahasa java.
2. Memahami konsep pemrograman penentu akses dalam Bahasa java.
3. Memahami prinsip-prinsip dasar pemrograman overloading.
4. Memahami prinsip-prinsip dasar pemrograman penentu akses.
5. Mengimpementasikan contoh penggunaan pemrograman overloading.
6. Mengimpementasikan contoh penggunaan pemrograman penentu akses.

# BAB II OVERLOADING

## PENGERTIAN OVERLOADING

Overloading adalah deklarasi method maupun konstruktor lebih dari satu kali pada kelas yang sama. Terdapat beberapa syarat overloading yaitu:

1. Memiliki jumlah parameter berbeda.
2. Jika jumlah parameter sama tipe data harus berbeda.

Overloading adalah salah satu cara penerapan dalam konsep polimorfisme.Jika kita tidak pernah pernah mengenal sebelumnya konsep ini kelihatan aneh, Tetapi metode inisebetulnya adalah hal yang sangat bermanfaat.

Overloading ada dua jenis, yaitu:

1. Overloading method
2. Overloading constructor

## OVERLOADING METHOD

Dalam Java memungkinkan untuk mendefinisikan dua atau lebih metode dalam kelas yang sama,menggunakan nama yang sama, tetapi parameter yang dideklarasikan harus berbeda (jumlah atau tipeparameternya). Jika jumlah parameternya sama maka Tipe harus berbeda semua, tetapi jika jumlahparameternya berbeda-beda maka boleh tipenya sama. Jenis-jenis metode overloading:

1. Jumlah parameter beda, tipe beda
2. Jumlah parameter sama, tipe beda
3. Jumlah parameter beda, tipe sama

Metode overloading ini, terjadi pada objek yang sama, tapi nilai argumennya berbeda-beda.Ketika metode overloaded dipanggil, Java mencari kesamaan antara argumen yang digunakan untuk memanggil metode dan parameter dari metode. Tetapi, kesamaan ini tidak mutlak. Hal ini karena pada java adanya otomatisasi konversi tipe yang berbeda.

**//PENERAPAN KONVERSI OTOMATIS PADA METODE OVERLOADING**

class OverloadMetode

{

//konversi tujuan bertipe integer

void test(int a)

{

System.out.println("Hasil konversi byte ke integer : " + a);

}

}

class OverloadKonversi

{

public static void main(String args[])

{

OverloadMetode ob = new OverloadMetode();

//variabel asal yang akan dikonversi bertipe bytebyte i= 88;

ob.test(i); // memanggil test(integer)

}

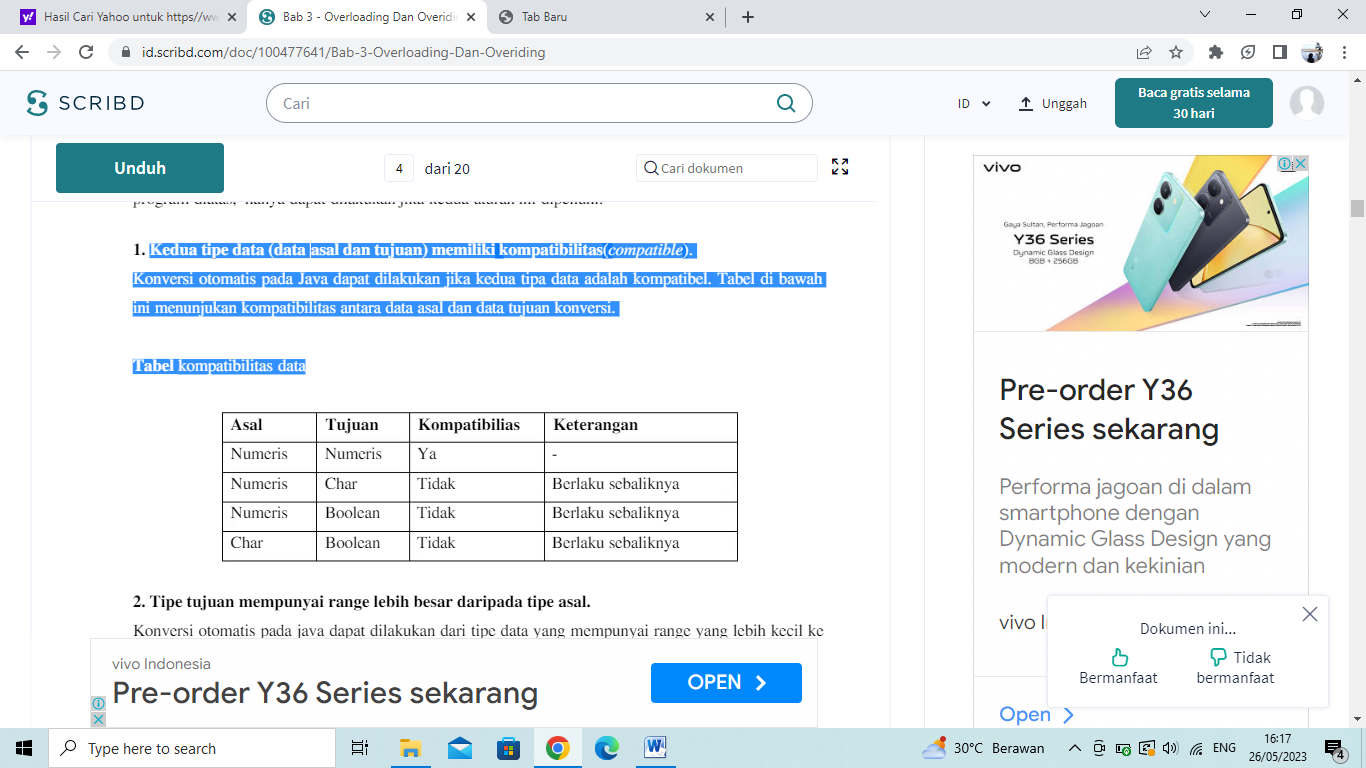
}

Seperti kita lihat, OverloadDemo tidak mendefinisikan test(byte). Maka dari itu,Ketika test( ) dipanggil dengan argument byte didalam OverloadKonversi, menyebabkanKetidaksamaan metode. Java dapat secara otomatis mengkonversi Byte ke dalam integer, dankonversi ini dapat digunakan untuk menyelesaikan masalah ketidaksamaan ini. Maka dari itu , sesudah test(byte) tidak ditemukan, Java mengubah tipe i kebentuk yang lebih tinggi ke integer dan kemudian memanggil test(int). Hal ini dilakukan hanya jika Java menemukan ketidaksamaan metode.Pada umumnya ketidaksamaan terjadi pada tipe– tipe numeris. Konversi otomatis pada java sepertiprogram diatas, hanya dapat dilakukan jika kedua aturan ini dipenuhi:

1. **Kedua tipe data (data asal dan tujuan) memiliki kompatibilitas (compatible).**

Konversi otomatis pada Java dapat dilakukan jika kedua tipa data adalah kompatibel. Tabel di bawahini menunjukan kompatibilitas antara data asal dan data tujuan konversi.

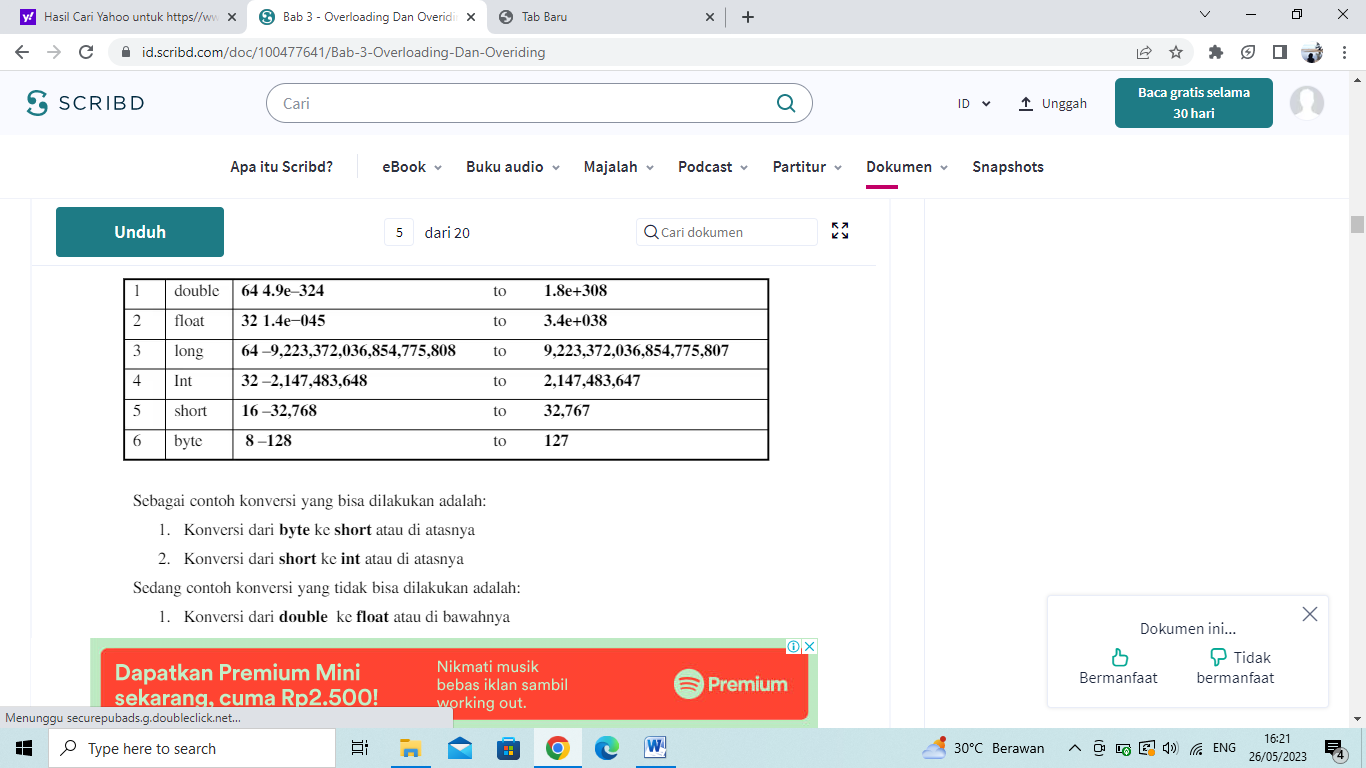
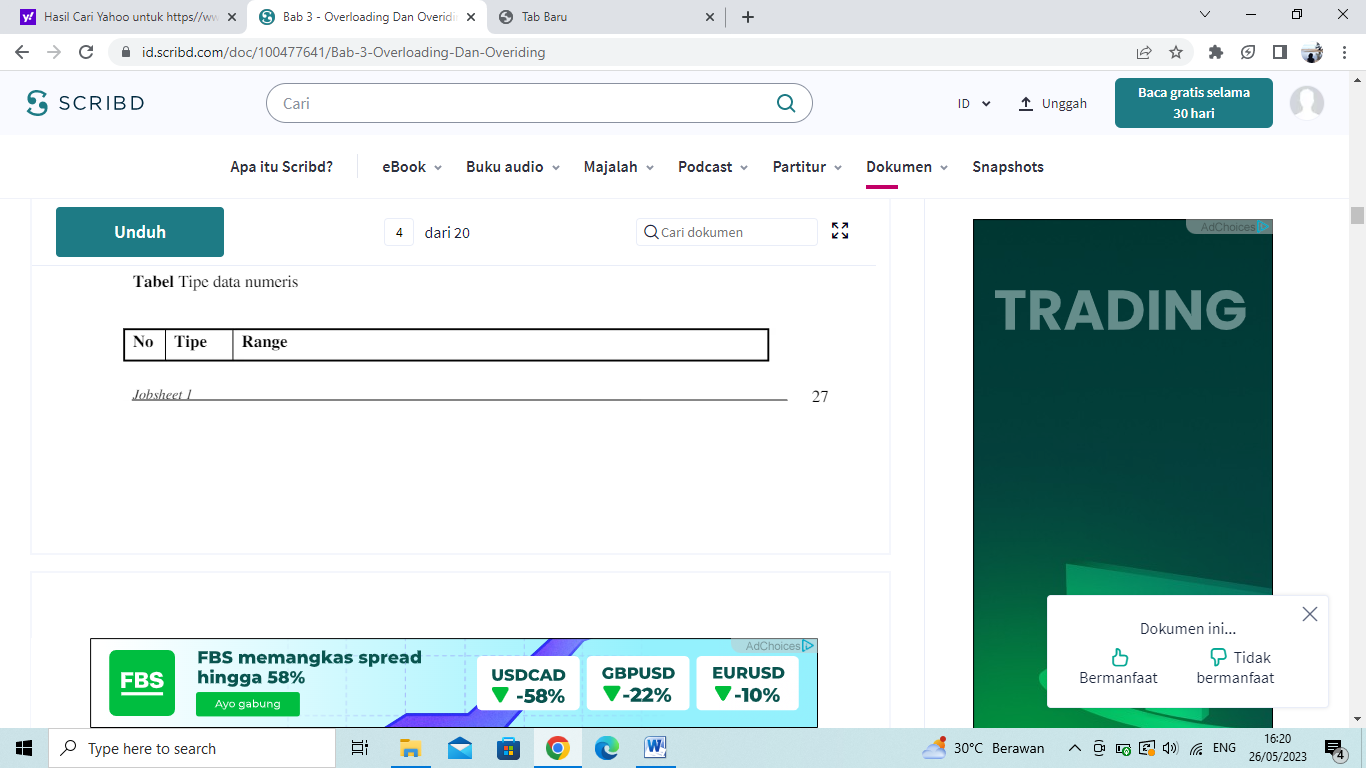
Tabel kompatibilitas data



1. **Tipe tujuan mempunyai range lebih besar daripada tipe asal.**

Konversi otomatis pada java dapat dilakukan dari tipe data yang mempunyai range yang lebih kecil kerange yang lebih besar, dan biasanya konversi terjadi pada tipe data numeris. Tabel dibawah inimenunjukan urutan range dari yang besar ke range yang kecil.

Tabel Tipe data numeris



Sebagai contoh konversi yang bisa dilakukan adalah:

1. Konversi dari **byte** ke **short** atau di atasnya
2. Konversi dari **short** ke **int** atau di atasnya.

Sedang contoh konversi yang tidak bisa dilakukan adalah:

1. Konversi dari **double** ke **float** atau di bawahnya
2. Konversi dari **Int** ke **short** atau di dibawahnya

Selain itu Java juga mengerjakan konversi secara otomatis ketika menyimpan konstanta integer literal menjadi variable bertipe **byte**, **short**, atau **long**.

## OVERLOADING CONSTRUCTORS

Overloading selain diterapkan pada **metode** juga diterapkan pada **konstuktor**. Bedanya overloading constructor terjadi pada objek yang berbeda-beda, nilai argument berbeda, sehingga metode bisa lebih dari satu. Penerapan overloading pada kelas Java akan menjadi suatu norma tanpa perkecualian.

**Kapan kita membutuhkan Constructor Overloading?**

Terkadang ada kebutuhan untuk menginisialisasi objek dengan cara yang berbeda. Ini dapat dilakukan dengan menggunakan overloading konstruktor. Misalnya, kelas Thread memiliki 8 jenis konstruktor. Jika kita tidak ingin menentukan apa pun tentang utas maka kita cukup menggunakan konstruktor default dari kelas Thread, namun jika kita perlu menentukan nama utas, maka kita dapat memanggil konstruktor berparameter dari kelas Thread dengan argumen String seperti ini:

Utas t= Utas baru (" Utas Saya ");

Mari kita ambil contoh untuk memahami kebutuhan overloading konstruktor. Pertimbangkan implementasi Box kelas berikut dengan hanya satu konstruktor yang mengambil tiga argumen.

// Contoh kelas untuk memahami kebutuhan

// kelebihan beban konstruktor.

Kotak kelas

{

lebar ganda, tinggi, kedalaman;

// konstruktor digunakan saat semua dimensi

// ditentukan

Kotak (w ganda, h ganda, d ganda)

{

lebar = w;

tinggi = h;

kedalaman = d;

}

// hitung dan kembalikan volume

volume ganda()

{

kembali lebar \* tinggi \* kedalaman;

}

}

Seperti yang dapat kita lihat bahwa konstruktor Box() membutuhkan tiga parameter. Ini berarti bahwa semua deklarasi objek Box harus meneruskan tiga argumen ke konstruktor Box(). Misalnya, pernyataan berikut saat ini tidak valid:

Kotak ob = Kotak baru();

Karena Box() memerlukan tiga argumen, merupakan kesalahan untuk memanggilnya tanpa argumen. Misalkan kita hanya menginginkan objek kotak tanpa dimensi awal, atau ingin menginisialisasi kubus dengan menentukan hanya satu nilai yang akan digunakan untuk ketiga dimensi tersebut. Dari implementasi kelas Box di atas, opsi ini tidak tersedia untuk kita. Jenis masalah dengan cara berbeda untuk menginisialisasi objek ini dapat diselesaikan dengan kelebihan beban konstruktor. Di bawah ini adalah versi Box kelas yang disempurnakan dengan kelebihan beban konstruktor.

// Java program to illustrate

// Constructor Overloading

class Box

{

double width, height, depth;

// constructor used when all dimensions

// specified

Box(double w, double h, double d)

{

width = w;

height = h;

depth = d;

}

// constructor used when no dimensions

// specified

Box()

{

width = height = depth = 0;

}

// constructor used when cube is created

Box(double len)

{

width = height = depth = len;

}

// compute and return volume

double volume()

{

return width \* height \* depth;

}

}

// Driver code

public class Test

{

public static void main(String args[])

{

// create boxes using the various

// constructors

Box mybox1 = new Box(10, 20, 15);

Box mybox2 = new Box();

Box mycube = new Box(7);

double vol;

// get volume of first box

vol = mybox1.volume();

System.out.println(" Volume of mybox1 is " + vol);

// get volume of second box

vol = mybox2.volume();

System.out.println(" Volume of mybox2 is " + vol);

// get volume of cube

vol = mycube.volume();

System.out.println(" Volume of mycube is " + vol);

}

}

Keluaran:

Volume mybox1 adalah 3000.0

Volume mybox2 adalah 0,0

Volume mycube adalah 343.0

**Menggunakan this() dalam overloading konstruktor**

[Referensi this()](https://www.geeksforgeeks.org/this-reference-in-java/) dapat digunakan selama overloading konstruktor untuk memanggil konstruktor default secara implisit dari konstruktor berparameter. Harap diperhatikan, this() harus menjadi pernyataan pertama di dalam konstruktor.

// Java program to illustrate role of this() in

// Constructor Overloading

public class Box

{

double width, height, depth;

int boxNo;

// constructor used when all dimensions and

// boxNo specified

Box(double w, double h, double d, int num)

{

width = w;

height = h;

depth = d;

boxNo = num;

}

// constructor used when no dimensions specified

Box()

{

// an empty box

width = height = depth = 0;

}

// constructor used when only boxNo specified

Box(int num)

{

// this() is used for calling the default

// constructor from parameterized constructor

this();

boxNo = num;

}

public static void main(String[] args)

{

// create box using only boxNo

Box box1 = new Box(1);

// getting initial width of box1

System.out.println(box1.width);

}

}

Keluaran:

0,0

Seperti yang dapat kita lihat pada program di atas bahwa kita memanggil konstruktor Box(int ​​num) selama pembuatan objek hanya menggunakan nomor kotak. Dengan menggunakan pernyataan this() di dalamnya, konstruktor(Box()) default secara implisit dipanggil darinya yang akan menginisialisasi dimensi Kotak dengan 0. Catatan: **Pemanggilan** konstruktor harus menjadi pernyataan pertama di badan konstruktor. Misalnya, fragmen berikut tidak valid dan memunculkan kesalahan waktu kompilasi.

Kotak(int num)

{

kotakNo = num;

/\* Panggilan konstruktor harus menjadi yang pertama

pernyataan dalam konstruktor \*/

ini(); /\*KESALAHAN\*/

}

**Poin penting yang harus diperhatikan saat melakukan Constructor Overloading :**

* Pemanggilan konstruktor harus menjadi pernyataan konstruktor **pertama di Jawa.**
* Jika kita telah mendefinisikan konstruktor berparameter apa pun, maka kompiler tidak akan membuat konstruktor default. dan sebaliknya jika kita tidak mendefinisikan konstruktor apa pun, kompiler membuat konstruktor default (juga dikenal sebagai konstruktor no-arg) secara default selama kompilasi
* Panggilan konstruktor rekursif tidak valid di java.

## CONTOH PROGRAM **OVERLOADING PADA JAVA**

1. **package** overloadconst;
3. **import** barang.Barang;
5. **public** **class** Main {
6. **static** Barang barang;
8. **public** **static** **void** main(String[] args) {
10. barang = **new** Barang("syarif soden", "Manga", 10000);
11. **int** total\_harga = barang.hitungHargaBeli(3);
12. System.out.println("Total Harga : "+ total\_harga);
13. **int** bayar = 50000;
14. **int** kembalian = barang.sisaKembalian(bayar);
15. System.out.println("Anda membayar : "+ bayar);
16. System.out.println("Kembalian : "+ kembalian);
18. }
20. }

Terakhir silahkan masukan kode dibawah pada Barang.java

1. **package** barang;
3. **public** **class** Barang {
4. String nama\_barang;
5. String nama\_pembeli;
6. **int** harga;
7. **int** total\_harga;
9. *//overloading*
10. **public** Barang(){
11. }
13. **public** Barang(String nama\_pembeli, String nama\_barang, **int** harga){
14. **this**.nama\_barang = nama\_barang;
15. **this**.nama\_pembeli = nama\_pembeli;
16. **this**.harga = harga;
17. infoBarang();
18. }
20. **private** **void** infoBarang() {
21. System.out.println("Barang berhasil dibuat dengan rincian sbb :**\n**"
22. + "Nama Barang : " + **this**.nama\_barang
23. + "**\n** Nama Pembeli : "+ **this**.nama\_pembeli
24. + "**\n** Harga Barang : "+ **this**.harga);
25. *//To change body of generated methods, choose Tools | Templates.*
26. }
28. **public** **int** hitungHargaBeli(**int** jumlah){
29. **this**.total\_harga = **this**.harga \* jumlah;
30. **return** **this**.total\_harga;
31. }
33. **public** **int** sisaKembalian(**int** nominalUang){
34. **return** nominalUang - **this**.total\_harga;
35. }
36. }

Output program:

Barang berhasil dibuat dengan rincian sbb:

Nama Barang : Mangga

Nama Pembeli : syarif sodden

Harga Barang : 10000

Total Harga : 20000

Anda membayar : 60000

Kembalian : 20000

BUILD SUCCESSFUL (total time : 1 second)

Penjelasan Program:

Dari contoh program di atas, variable barang = new Barang("syarif soden", "Manga", 10000);  pada class main.java akan menginputkan data ke class Barang.java pada package barang, selanjutnya dari data tersebut akan mengakses method hitungHargaBeli dan juga sisaKembalian yang ada pada class Barang.java di package barang. Setelah data di proses selanjutnya data akan ditampilkan ke layar monitor.

## PERBEDAAN OVERLOADING DAN OVERIDDING PADA JAVA

Dalam bahasa pemrograman Java, tidak ada istilah "overreading" yang digunakan dalam konteks pemrograman. Mungkin yang dimaksudkan adalah "overriding". Jadi, saya akan menjelaskan perbedaan antara "overloading" dan "overriding" dalam Java.

1. **Overloading (Pengisian Berlebihan):**

Overloading terjadi ketika terdapat beberapa metode dengan nama yang sama tetapi dengan parameter yang berbeda dalam suatu kelas. Metode-metode ini harus memiliki nama yang sama tetapi memiliki tipe, jumlah, atau urutan parameter yang berbeda. Overloading memungkinkan Anda menggunakan nama yang sama untuk operasi yang serupa tetapi dengan tipe atau jumlah parameter yang berbeda. Saat memanggil metode yang di-overload, kompilator Java akan memilih metode yang paling sesuai berdasarkan argumen yang diberikan.

Contoh overloading:

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

}

1. Overriding (Penggantian)

Overriding terjadi ketika sebuah subclass memiliki metode dengan nama yang sama, parameter yang sama, dan tipe pengembalian yang sama dengan metode yang diwarisi dari superclassnya. Dalam overriding, subclass memberikan implementasi ulang dari metode superclass yang telah ditentukan sebelumnya. Ini memungkinkan subclass untuk memiliki perilaku yang berbeda untuk metode yang sama.

Contoh overriding:

public class Animal {

public void makeSound() {

System.out.println("Animal makes a sound");

}

}

public class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Meow");

}

}

Dalam contoh ini, kelas **Cat** mewarisi metode **makeSound** dari kelas **Animal** dan memberikan implementasi ulangnya dengan perilaku yang berbeda. Ketika Anda memanggil metode **makeSound** pada objek **Cat**, maka akan mencetak "Meow" sebagai output.

Dengan demikian, perbedaan antara overloading dan overriding adalah bahwa overloading terjadi saat metode-metode memiliki nama yang sama tetapi parameter yang berbeda dalam suatu kelas, sementara overriding terjadi ketika metode subclass memiliki nama, parameter, dan tipe pengembalian yang sama dengan metode superclass dan memberikan implementasi ulang dari metode superclass tersebut.

# BAB III PENENTU AKSES

## PENENTU AKSES

Access specifier atau pengubah akses di java menentukan aksesibilitas (ruang lingkup) dari sebuah data anggota, metode, konstruktor atau kelas. Ini menentukan apakah sebuah data atau metode dalam sebuah kelas dapat digunakan atau dipanggil oleh kelas atau subkelas lain.

Pengubah akses dapat ditentukan secara terpisah untuk kelas, konstruktor,bidang, dan metode. Mereka juga disebut sebagai *Penentu akses Java* , tapi nama yang benar adalah **Pengubah akses Java** .

Access Modifier adalah terminologi berorientasi objek yang digunakan untuk mendefinisikan aksesibilitas kelas, metode, dan konstruktor. Menjadi pemrograman berorientasi objek, Java mendukung empat tingkat pengubah akses yang berbeda dan setiap pengubah akses beroperasi pada tingkat yang berbeda. Pengubah akses mungkin salah satu dari berikut ini, default, publik, pribadi, dan dilindungi. Pemilihan pengubah akses tergantung pada masalah yang akan Anda pecahkan. Panduan deskriptif ini menunjukkan fungsionalitas pengubah akses di Java.

Dalam Java, terdapat empat penentu akses yang umum digunakan: public, private, protected, dan default (juga dikenal sebagai "tidak ada penentu akses"). Masing-masing penentu akses memiliki penggunaan yang spesifik dan mempengaruhi tingkat aksesibilitas komponen tersebut.

1. Public: Penentu akses public mengizinkan akses penuh terhadap kelas, metode, atau variabel dari mana pun di dalam program Java. Jika suatu kelas atau anggota kelas dideklarasikan sebagai public, maka mereka dapat diakses oleh kelas lain di dalam program, bahkan jika kelas tersebut berada dalam paket yang berbeda.
2. Private: Penentu akses private membatasi aksesibilitas suatu kelas, metode, atau variabel hanya pada kelas di mana mereka dideklarasikan. Anggota kelas private tidak dapat diakses dari kelas lain, bahkan jika kelas tersebut berada dalam paket yang sama.
3. Protected: Penentu akses protected memungkinkan aksesibilitas terhadap kelas, metode, atau variabel hanya pada kelas di dalam paket yang sama atau pada subkelas dari kelas tersebut, terlepas dari apakah subkelas tersebut berada dalam paket yang sama atau berbeda. Penentuan akses protected sering digunakan dalam konteks pewarisan (inheritance) di Java.
4. Default: Penentu akses default (atau "tidak ada penentu akses") adalah tingkat akses yang diberikan saat tidak ada penentu akses yang dideklarasikan secara eksplisit. Jika tidak ada penentu akses yang ditentukan, maka komponen tersebut hanya dapat diakses oleh kelas-kelas yang berada dalam paket yang sama.

Masing-masing penentu akses memiliki batasan dan kegunaannya sendiri, yang harus dipahami oleh pengembang perangkat lunak Java. Dalam makalah ini, kami akan membahas contoh penggunaan dan batasan dari masing-masing penentu akses, serta memberikan panduan praktis dalam memilih penentu akses yang tepat dalam pengembangan perangkat lunak.

Contoh-contoh penggunaan penentu akses dalam Java akan diberikan dalam konteks pengembangan perangkat lunak untuk memberikan pemahaman yang lebih konkrit dan membantu pembaca menerapkan pengetahuan ini dalam proyek-proyek mereka.

## PENENTU AKSES PRIVAT

Private berarti bahwa pengaksesan suatu variabel instan atau metode hanya dapat dilakukan didalam kelas, tidak bisa diakses dari luar kelas. Jadi, penentu akses privat adalah penentu akses yang pengaksesannya hanya dapat dilakukan didalam kelas, tidak bisa diakses dari luar kelas. Penentu akses private membatasi aksesibilitas suatu kelas, metode, atau variabel hanya pada kelas di mana mereka dideklarasikan. Anggota kelas private tidak dapat diakses dari kelas lain, bahkan jika kelas tersebut berada dalam paket yang sama.

Penentuan akses privat merupakan aspek penting dalam pemrograman berorientasi objek untuk membatasi akses langsung terhadap variabel dan metode dalam suatu kelas. Dalam bahasa pemrograman Java, terdapat beberapa tingkatan akses yang dapat diterapkan pada elemen-elemen kelas, seperti variabel dan metode, termasuk akses privat.

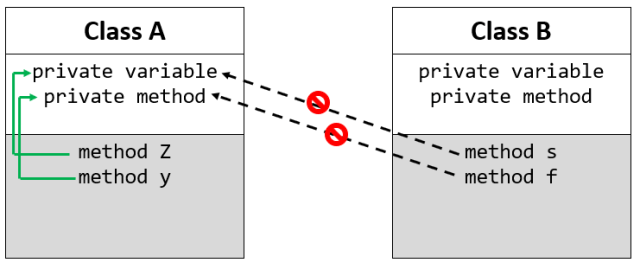
Sintaksis:

jam kelas publik {private long time = 0}

Sesuai dengan namanya, *private* atau *pribadi*. Artinya hanya bisa diakses oleh pemiliknya. Sehingga, sesuatu yang diberikan *access modifier* *private* hanya dapat diakses oleh *class*nya sendiri. Ada beberapa ketentuan khusus tentang penggunaan *access modifier*ini yakni:

* Data dan method *private*hanya bisa diakses di dalam *class* yang memilikinya
* *Class* dan *Interface* tidak dapat dideklarasikan sebagai *private*
* Jika sebuah *class* memiliki konstruktor *private* maka kita tidak dapat membuat *object* dari *class* tersebut di luar *class*tersebut*.*

Penggambaran untuk *access modifier* *private*adalah sebagai berikut.

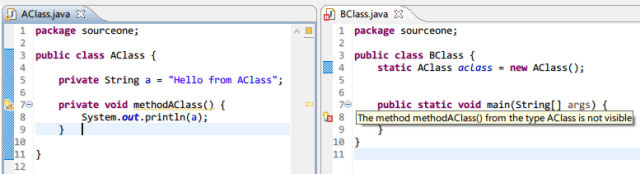


Ciri-ciri penentu akses privat:

* Metode atau anggota data yang dideklarasikan sebagai privat hanya dapat diakses di dalam kelas tempat mereka dideklarasikan.
* Kelas atau antarmuka tingkat atas tidak dapat dideklarasikan sebagai pribadi mengingat fakta itu
* Pribadi menandakan ' hanya terlihat di dalam kelas penutup '.
* Dilindungi menandakan ' j hanya terlihat di dalam kelas yang melampirkan dan setiap subkelas '.
* Jika kelas memilikiuntukkonstruktor pribadi maka Anda tidak dapat membuat objek kelas itu dari luar kelas.
* Kelas tidak dapat ditandai dengan pengubah akses pribadi .
* Mendenotasikan kelas dengan pengubah akses privat akan menyiratkan bahwa tidak ada kelas yang berbeda yang bisa mendapatkannya. Ini biasanya menyiratkan bahwa Anda tidak dapat memanfaatkan kelas dengan imajinasi apa pun. Dengan cara ini, pengubah akses pribadi tidak memperhitungkan kelas akun.

Penggambaran *access modifier private*

Dari penggambaran di atas, dapat kita lihat bahwa secara mudahnya apabila sebuah *class* memiliki *private variable* dan *private method*maka tidak dapat diakses dari *class*lain. Kemudian bagaimanakah implementasinya? mari kita perhatikan kodingan berikut ini.



Dapat kita lihat pada kodingan di atas, bahwa private variable pada class AClass dapat diakses oleh method yang ada di dalam class tersebut. Sedangkan method methodAClass sama sekali tidak dapat diakses dari BClass karena methodAClass memiliki access modifier private. Sehingga pada IDE memunculkan error: The method methodAClass() from the type AClass is not visible. Jadi itulah fungsi access modifier private, yakni membuat apapun yang diberikan access modifier private menjadi not visible sehingga tidak dapat diakses di luar class yang memilikinya.

Method akses privat:

public class StudentRecord

{

         //akses dasar terhadap variabel

         private int name;

          //akses dasar terhadap metode

          private String getName () {

                      return name;

           }

}

Pada contoh diatas, variabel name dan method getName() hanya dapat diakses oleh methode internal class tersebut.

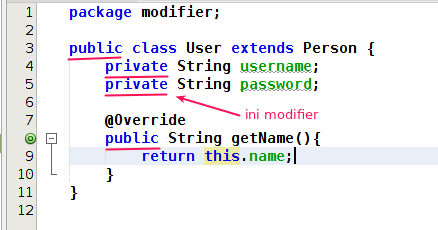
Perbedaan public, private, dan tanpa penentu akses pada variabel:

1. Tanpa penentu akses Variabel dapat diakses oleh kelas yang berada dalam paket yang sama.
2. Public Variabel dapat diakses pada semua mkelas yang menggunakan kelas yang berisi variable tersebut.
3. Private hanya dapat diakses oleh metode-metode dalam kelas itu sendiri.

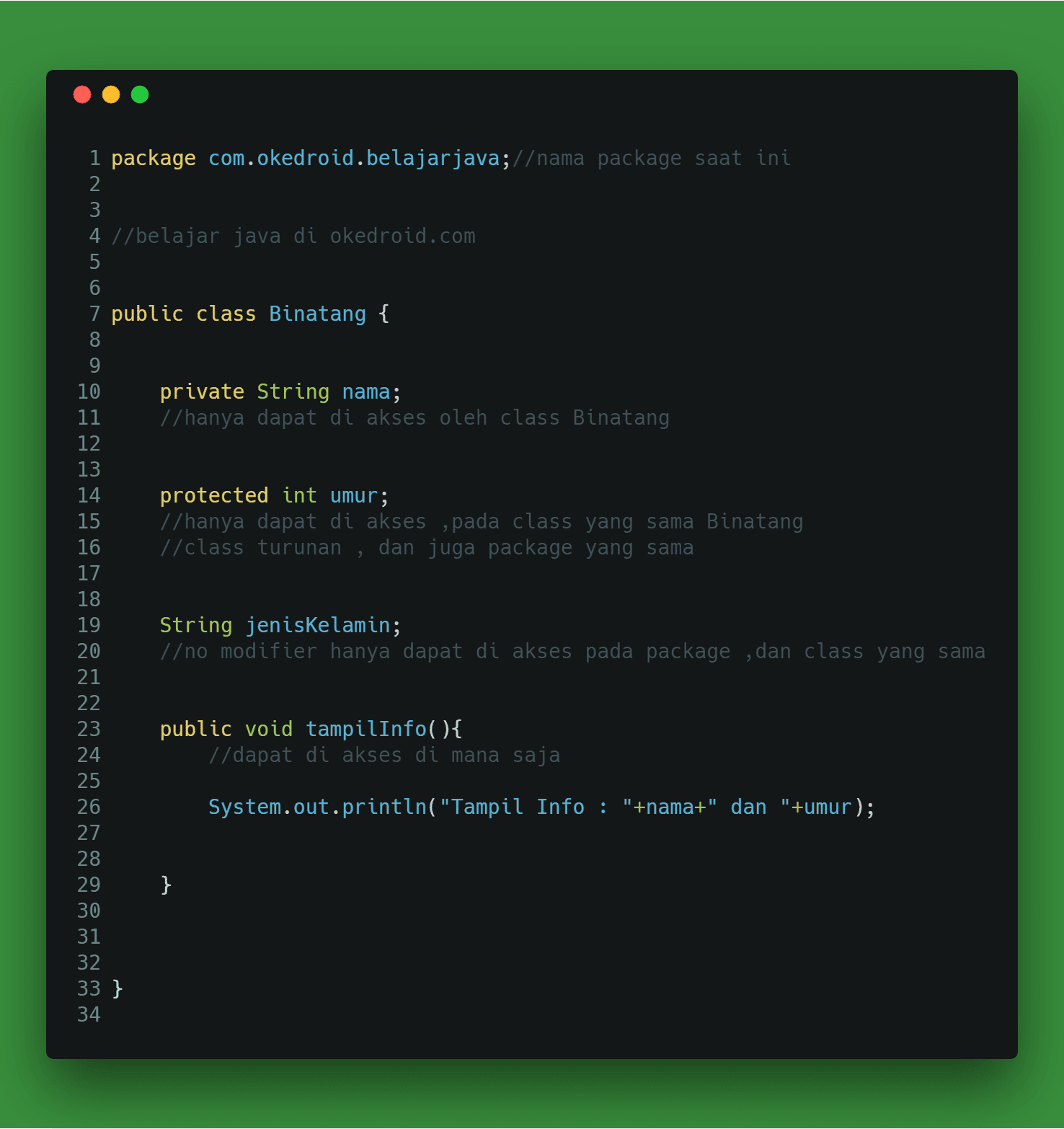
Perbedaan public, private, dan tanpa penentu akses pada method:

1. Tanpa penentu akses Metode hanya dapat diakses oleh metode dalam kelas itu sendiri ataupun metode pada kelas lain yang terletak pada paket yang sama.
2. Public Metode dapat diakses oleh sembarang kelas.
3. Private hanya dapat diakses oleh metode-metode dalam kelas yang sama.

Contoh program penentu akses privat:



Hanya dapat diakses oleh metode2 dalam kelas itu sendiri. Belajar Java Oop Memahami Tingkatan Akses Member Dan Class Modifier Contoh Coding Untuk Public Private Dan Protected.



Di mulai dengan hak akses publik. Taeganger Program Sederhana Menggunakan Modifier Public Private Dan Protected Contoh Coding Untuk Public Private Dan Protected.

## PENENTU AKSES PUBLIK

Suatu anggota kelas (class member) dideklarasikan sebagai publik, maka anggota tersebut akan dapat digunakan oleh siapa saja, yaitu baik oleh kelas yang sama, kelas lain maupun lingkungan luar kelas. Sehingga anggota kelas (class member) yang dideklarasikan sebagai public akan bisa diakses oleh sembarang objek lain. Public adalah kode akses yang bersifat umum. dengan kata lain, data maupun method dalam suatu kodingan tersebut dapat diakses oleh semua bagian di dalam program.

Penentu akses publik merupakan salah satu konsep penting dalam pemrograman berorientasi objek. Dalam bahasa pemrograman Java, terdapat beberapa tingkatan akses yang dapat diberikan pada elemen-elemen dalam sebuah kelas, seperti variabel, metode, dan konstruktor. Pengetahuan tentang penentu akses publik pada Java sangat penting bagi pengembang perangkat lunak untuk memastikan keamanan dan kegunaan program yang mereka tulis.

Dalam konteks desain perangkat lunak, terdapat beberapa prinsip desain yang berkaitan dengan penentu akses publik (public access determinants). Prinsip-prinsip ini bertujuan untuk menciptakan desain perangkat lunak yang mudah digunakan, terbuka untuk pengembangan, dan memberikan akses yang tepat kepada pengguna. Berikut adalah beberapa prinsip desain yang relevan dengan penentu akses publik dalam bahasa pemrograman Java:

1. Prinsip Enkapsulasi (Encapsulation): Prinsip ini menekankan pada pembungkusan data dan fungsi dalam sebuah objek, sehingga akses ke data dan fungsi tersebut hanya dapat dilakukan melalui antarmuka yang ditentukan. Dalam konteks penentu akses publik, Anda dapat menggunakan enkapsulasi untuk menyembunyikan implementasi internal dan mengizinkan akses terbatas ke komponen-komponen publik yang ditentukan.
2. Prinsip Pewarisan (Inheritance): Pewarisan memungkinkan Anda untuk membuat hierarki kelas di mana kelas-kelas anak mewarisi sifat dan perilaku dari kelas induk. Dalam konteks penentu akses publik, Anda dapat menggunakan pewarisan untuk mengatur tingkat akses kelas dan metode yang diwarisi oleh kelas-kelas anak. Misalnya, Anda dapat membuat kelas dasar dengan metode-metode publik yang dapat diakses oleh kelas-kelas anak.
3. Prinsip Polimorfisme (Polymorphism): Polimorfisme memungkinkan objek untuk mengambil banyak bentuk, di mana objek dari kelas yang berbeda dapat diperlakukan sebagai objek dari tipe yang sama. Dalam konteks penentu akses publik, Anda dapat menggunakan polimorfisme untuk menyediakan antarmuka yang konsisten untuk berinteraksi dengan objek-objek yang berbeda, tetapi masih mematuhi batasan-batasan akses yang telah ditentukan.
4. Prinsip Komposisi (Composition): Komposisi melibatkan pembuatan objek baru dari objek-objek yang lebih kecil. Dalam konteks penentu akses publik, Anda dapat menggunakan komposisi untuk membangun struktur perangkat lunak yang kompleks dengan memisahkan tanggung jawab antara objek-objek yang berbeda. Anda dapat mengatur tingkat akses untuk komponen-komponen tersebut sesuai kebutuhan.
5. Prinsip Ketergantungan Terbalik (Dependency Inversion): Prinsip ini mengusulkan agar ketergantungan antar kelas berdasarkan pada abstraksi daripada implementasi konkret. Dalam konteks penentu akses publik, ini berarti bahwa kode yang menggunakan komponen publik harus bergantung pada antarmuka atau abstraksi, bukan pada implementasi yang spesifik. Hal ini memungkinkan fleksibilitas dan penggantian implementasi tanpa mempengaruhi kode yang bergantung.
6. Prinsip Single Responsibility (SRP): Prinsip ini mengusulkan bahwa setiap kelas atau komponen harus memiliki satu dan hanya satu tanggung jawab. Dalam konteks penentu akses publik, ini berarti setiap kelas atau komponen harus fokus pada tugas-tugas yang berkaitan dengan penentuan akses publik saja. Dengan membatasi tanggung jawab, kode menjadi lebih terorganisir dan mudah dipahami.
7. Prinsip Interface Segregation (ISP): Prinsip ini menekankan pada pembagian antarmuka menjadi bagian-bagian yang lebih kecil dan spesifik. Dalam konteks penentu akses publik, Anda dapat menggunakan ISP untuk memisahkan antarmuka yang berhubungan dengan akses publik menjadi komponen-komponen yang lebih kecil, sehingga pengguna hanya perlu mengimplementasikan bagian yang relevan bagi mereka

Ciri-ciri penentu akses public:

* Pengubah akses publik ditentukan menggunakan kata kunci ***publik.***
* Pengubah akses publik memiliki cakupan yang luas di antara semua pengubah akses lainnya.
* metode atau anggota data yang dideklarasikan sebagai publik adalah dapat diakses di mana saja program. Tidak ada batasan pada ruang lingkup anggota data publik.

Sintaksis:

package [edureka.co](https://edureka.co/) public class PublicClassDemo {

// Di sini saya tidak menyebutkan pengubah apa pun sehingga bertindak sebagai pengubah default

public int myMethod (int x) {return x}

}

Sekarang, lihat contoh untuk mendapatkan gambaran yang jelas tentang pengubah akses publik ini.

Contoh:

paket p1 kelas publik A {tampilan void publik () {

System.out.println ('edureka!')}}

paket p2 impor p1. \* kelas B {public static void main (String args []) {A obj = baru A obj.display ()}}

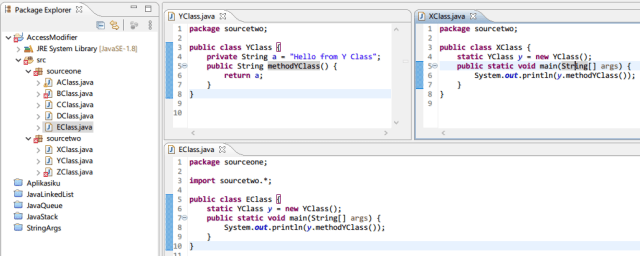
Keluaran:

edureka!

[Adalah Master Gelar Pascasarjana](https://id.ichlese.at/postgraduate-certificate-vs-master-s-degree)

Ini adalah segalanya tentang pengubah akses publik di Java.

Melihat dari judulnya, ya publik. Artinya memang publik alias dapat diakses oleh siapa saja. Entah itu dari *package* yang berbeda, semua dapat mengaksesnya. Bahkan, ketika sesuatu diberikan *access modifier public*, yang tadinya *private* pun menjadi *public*. Bingung? Mari kita perhatikan kodingan berikut ini.



Mari kita perhatikan kodingan di atas. Pada *class YClass* pada *package* *sourcetwo* memiliki sebuah *private variable*. Namun, memiliki sebuah method *public* yang dapat mengembalikan nilai dari *private variable tersebut*. Karena *method tersebut bersifat public* maka dapat diakses oleh siapa saja, contohnya *class XClass* dan *EClass* yang berasal dari *package sourceone*. Jadi, hati – hatilah menggunakan *access modifier public* karena dapat mematikan *access modifier* lainnya (contoh *private*dan *protected*).

Contoh penentu akses public:

public class Mahasiswa {

//variable instan

String nama;

//methode

void isiNama (String nama){

this.nama=nama;

}

//methode

String tampilkanNama (){

return this.nama;

}

}

**PenentuAkses.java**

public class PenentuAkses {

public static void main (String [] args){

Mahasiswa saya = new Mahasiswa ();

//mengisi variable instan

saya.isiNama(“Mafati Hurrohmah”);

//menampilkan nama melalui variable

System.out.println (saya.nama);

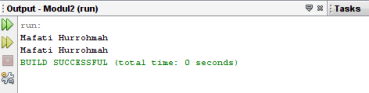
//menampilkan nama melalui pemanggilan methode

System.out.println (saya.tampilkanNama ());

}

}

**Hasil:**



Class Mahasiswa dan PenentuAkses merupakan Public jadi pengaksesan suatu variabel instan atau metode dapat dilakukan dari luar kelas.

## PENENTU AKSES PROTECTED

Dalam bahasa pemrograman Java, terdapat beberapa tingkat aksesibilitas yang digunakan untuk mengatur akses ke anggota kelas seperti variabel, metode, dan konstruktor. Salah satu tingkat aksesibilitas yang tersedia adalah "protected". Akses protected memungkinkan anggota kelas hanya dapat diakses oleh kelas yang sama, kelas turunan, atau kelas dalam paket yang sama.

Menggunakan tingkat akses **protected** dalam desain kelas memiliki beberapa keuntungan yang penting. Berikut adalah beberapa di antaranya:

1. Enkapsulasi: Tingkat akses **protected** membantu dalam menerapkan prinsip enkapsulasi dalam pemrograman berorientasi objek. Dengan menggunakan tingkat akses **protected**, Anda dapat membatasi akses ke anggota kelas hanya kepada kelas yang mengurus kelas tersebut dan kelas-kelas turunannya. Ini memungkinkan Anda untuk mempertahankan kontrol yang lebih baik terhadap bagaimana anggota kelas digunakan dan dimodifikasi di luar hierarki kelas yang relevan.
2. Pewarisan: Tingkat akses **protected** berperan penting dalam pewarisan. Ketika Anda mendeklarasikan anggota kelas sebagai **protected**, mereka dapat diakses oleh kelas-kelas turunannya. Ini memungkinkan kelas turunan untuk mewarisi perilaku dan data dari kelas dasar serta memodifikasinya atau menambahkan perilaku tambahan sesuai kebutuhan. Dengan tingkat akses ini, Anda dapat membangun hierarki kelas yang kuat dan fleksibel.
3. Fleksibilitas: Menggunakan tingkat akses **protected** memungkinkan Anda untuk memisahkan antara apa yang harus terlihat oleh klien atau pengguna kelas dan apa yang hanya relevan untuk implementasi internal kelas. Dengan membatasi akses langsung ke anggota kelas dengan menggunakan tingkat akses **protected**, Anda dapat membuat perubahan internal pada kelas tanpa mempengaruhi kode di luar kelas. Ini memungkinkan fleksibilitas dalam mengubah implementasi kelas tanpa merusak kode yang menggunakan kelas tersebut.
4. Pengendalian Versi: Tingkat akses **protected** dapat membantu dalam pengendalian versi (version control) dalam proyek perangkat lunak. Dengan membatasi akses langsung ke anggota kelas, Anda dapat membuat perubahan internal pada kelas tersebut tanpa mempengaruhi klien yang menggunakannya. Ini memungkinkan Anda untuk mengelola versi kelas secara lebih efektif dan mengurangi kemungkinan kerusakan pada kode yang ada.

Namun, penting untuk diingat bahwa menggunakan tingkat akses **protected** juga membutuhkan kebijaksanaan yang tepat. Anda harus mempertimbangkan dengan cermat desain kelas dan hubungan antara kelas-kelas terkait sebelum menggunakan tingkat akses ini. Pemahaman yang baik tentang prinsip-prinsip pemrograman berorientasi objek dan tujuan desain yang diinginkan akan membantu Anda mengambil keputusan yang tepat dalam menggunakan tingkat akses **protected**.

Berikut contoh program pada penentu akses Protected:

public class MyClass {

protected int protectedVariable;

protected void protectedMethod() {

// Kode method

}

}

## PENENTU AKSES DEFAULT

Akses default adalah jenis aksesibilitas yang diberikan secara implisit jika tidak ada modifikator akses yang dideklarasikan secara eksplisit. Anggota dengan akses default dapat diakses oleh kelas-kelas yang berada dalam paket yang sama, tetapi tidak dapat diakses oleh kelas-kelas di paket yang berbeda.

Aturan Akses Default:

* Anggota kelas (metode atau variabel) yang tidak diberi modifikator akses memiliki akses default.
* Anggota dengan akses default hanya dapat diakses oleh kelas-kelas dalam paket yang sama.
* Anggota dengan akses default tidak dapat diakses oleh kelas-kelas di paket yang berbeda, kecuali jika mereka merupakan kelas turunan (subclass).

Keuntungan Penggunaan Akses Default:

* Mengatur batasan aksesibilitas pada anggota-anggota yang tidak perlu diakses oleh kelas-kelas di luar paket.
* Membantu dalam menerapkan konsep enkapsulasi dengan menyembunyikan implementasi internal dari kelas atau paket.
* Memudahkan dalam mengorganisir dan memecah program menjadi modul yang lebih kecil dan terpisah.

Contoh Penggunaan Akses Default: Berikut adalah contoh penggunaan akses default dalam Java:

// Paket: com.example

class MyClass {

int defaultVar;

void defaultMethod() {

// Implementasi

}

}

// Paket: com.example

class AnotherClass {

void accessDefaultMember() {

MyClass myObj = new MyClass();

int x = myObj.defaultVar; // Akses variabel dengan akses default

myObj.defaultMethod(); // Akses metode dengan akses default

}

}

// Paket: com.anotherexample

class SomeClass {

void accessDefaultMember() {

MyClass myObj = new MyClass();

int x = myObj.defaultVar; // Tidak dapat mengakses variabel dengan akses default

myObj.defaultMethod(); // Tidak dapat mengakses metode dengan akses default

}

}

# BAB IV PENUTUP

## KESIMPULAN

* Makalah ini membahas konsep overloading dalam bahasa pemrograman Java. Overloading adalah kemampuan untuk mendefinisikan beberapa metode dengan nama yang sama tetapi dengan parameter yang berbeda. Ini memungkinkan programmer untuk membuat kode yang lebih bersih, lebih fleksibel, dan lebih mudah dipahami. Makalah ini akan menjelaskan konsep overloading, aturan-aturan yang terkait, dan memberikan contoh-contoh implementasi overloading dalam bahasa pemrograman Java.
* Dalam makalah ini, telah dijelaskan konsep penentu akses dalam bahasa pemrograman Java, yang merupakan mekanisme penting untuk mengatur tingkat aksesibilitas komponen-komponen program. Pengetahuan tentang penentu akses membantu pengembang untuk mengelola visibilitas dan perlindungan terhadap kelas, metode, dan variabel dalam program Java. Dengan menggunakan penentu akses yang tepat, pengembang dapat membangun perangkat lunak yang aman, mudah dipelihara, dan efisien.
* Makalah ini akan menjelaskan secara rinci tentang penentu akses privat dalam bahasa pemrograman Java. Ini mencakup konsep dasar pemrograman berorientasi objek, tingkatan akses dalam Java, definisi dan manfaat penentu akses privat, serta contoh penggunaannya dalam implementasi kelas.
* Dalam makalah ini, dijelaskan secara mendalam tentang penentu akses publik pada Java, dan bagaimana penggunaannya dapat mempengaruhi desain dan fungsionalitas program. Pengembang perangkat lunak yang memahami konsep ini akan dapat membuat kode yang lebih terstruktur, aman, dan mudah dikelola.
* Makalah ini akan membantu pembaca untuk memahami dengan lebih baik konsep penentu akses protected dalam Java dan memberikan contoh implementasinya.
* Akses default dalam Java memainkan peran penting dalam mengatur tingkat aksesibilitas anggota kelas, metode, dan variabel. Dengan menggunakan akses default, kita dapat mengatur batasan aksesibilitas pada anggota-anggota.