

Access Modifier

Access modifier di dalam Object Oriented Programming (OOP) akan menentukan apakah kelas lain dapat menggunakan field atau meng-invoke methods dari suatu kelas. Ada beberapa macam access modifier yang dapat digunakan yaitu private, default, protected, dan public.

Private

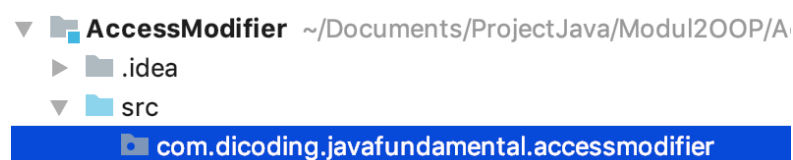
Access modifier private akan membatasi akses hanya di dalam *class*. Private biasanya digunakan sebagai modifier dari member dan metode suatu *class*.

Codelab Access Modifier Private

Mari kita coba access modifier private di dalam program.

1. Buatlah proyek baru dengan nama AccessModifier dengan nama package

`com.dicoding.javafundamental.accessmodifier` di dalamnya:



2. Buatlah sebuah kelas baru di dalamnya dengan nama `KelasA`, kemudian tambahkan kode berikut:

```
1. package com.dicoding.javafundamental.accessmodifier;
2.
3. public class KelasA {
4.
5.     private int memberA = 5;
6.
7.     private int functionA() {
8.         return memberA;
9.     }
10.
11.     int functionB() {
12.         // Pemanggilan private member dan private function
13.         int hasil = functionA() + memberA;
14.         return hasil;
15.     }
```

3. Buatlah kelas Main dan tambahkan kode berikut:

```
1. package com.dicoding.javafundamental.accessmodifier;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         KelasA kelasA = new KelasA();
6.
7.         System.out.println(kelasA.memberA);
8.
9.         System.out.println(kelasA.functionA());
10.
11.        System.out.println(kelasA.functionB());
12.    }
13. }
```

Jika diperhatikan, kode diatas akan terjadi eror karena `memberA` dan `functionA` dalam keadaan private, hal tersebut mengakibatkan tidak bisa diakses dari luar kelas.

4. Maka ubahlah kode di kelas Main menjadi seperti ini:

```
1. package com.dicoding.javafundamental.accessmodifier;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         KelasA kelasA = new KelasA();
6.         System.out.println(kelasA.functionB());
7.     }
8. }
```

5. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan output seperti ini.

10

Bedah Code Access Modifier Private

Perhatikan kode berikut:



```
1. public class KelasA {
2.
3.     private int memberA = 5;
4.
5.     private int functionA() {
6.         return memberA;
7.     }
8.
9.     int functionB() {
10.         // Pemanggilan private member dan private function
11.         int hasil = functionA() + memberA;
12.         return hasil;
13.     }
14. }
```

Dari kode di atas dapat dilihat bahwa ada kelas dengan nama `KelasA`. Di dalamnya ada 2 contoh kode yang menggunakan modifier private yaitu `memberA` dan `functionA`. Variabel `memberA` memiliki tipe data integer dengan nilainya 5. Sementara fungsi `functionA` memiliki nilai balik integer yang di ambil dari `memberA`, ini berarti `functionA` juga akan mengembalikan nilai 5. Kemudian yang perlu diperhatikan adalah pada fungsi `functionB`. Di dalamnya ada akses ke member private dan fungsi private. Jika dijalankan maka nilai yang didapatkan adalah 10 (5+5). Oleh karena itu, kode di atas merupakan contoh dari modifier private di mana akses ke variabel/fungsi hanya dari kelas tersebut.

Default

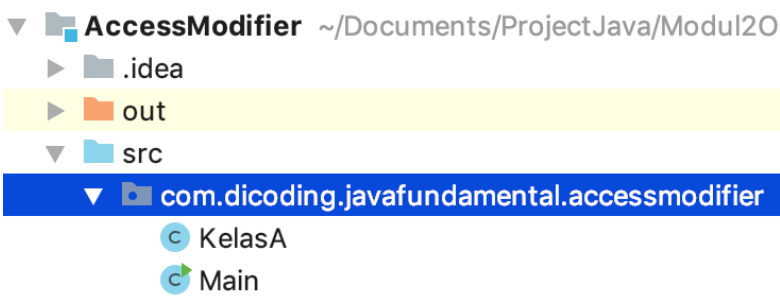
Default modifier berarti penulisan kodenya tanpa atribut modifier. Ini berlaku untuk semua kelas, member, atau fungsi yang kita tuliskan tanpa access modifier.

Modifier default bisa diakses selama masih dalam satu package.

Codelab Access Modifier Default

Mari kita tambahkan beberapa default modifier baru pada contoh sebelumnya yaitu `KelasA`.

1. Bukalah kembali proyek AccessModifier.



2. Masukkan kode berikut ke dalam kelas `kelasA` :

```

1. public class KelasA {
2.
3.     private int memberA = 5;
4.
5.     char memberB = 'A';
6.     double memberC = 1.5;
7.
8.     private int functionA() {
9.         return memberA;
10.    }
11.
12.    int functionB() {
13.        // Pemanggilan private member dan private function
14.        int hasil = functionA() + memberA;
15.        return hasil;

```

Perhatikan kode di atas. Ada 2 member baru yaitu `memberB` dan `memberC`. Keduanya tidak memiliki access modifier maka keduanya adalah termasuk default. Dan lihat juga `functionB`, fungsi ini juga termasuk default modifier. Terakhir, `KelasA` juga merupakan default modifier karena tidak memiliki access modifier.

3. Apa bedanya dengan access modifier private? Lihatlah contoh pemakaian dari `KelasA` di kelas `Main`.

Bukalah kelas `Main` dan tambahkan kode berikut:

```

1. package com.dicoding.javafundamental.accessmodifier;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         KelasA kelasA = new KelasA();
6.
7.         System.out.println(kelasA.functionB());
8.
9.         System.out.println(kelasA.memberB);
10.        System.out.println(kelasA.memberC);
11.    }
12. }

```

4. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

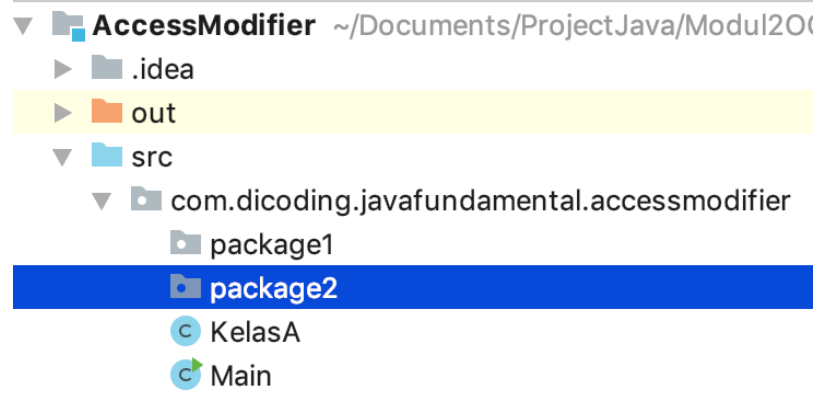
```

10
A
1.5

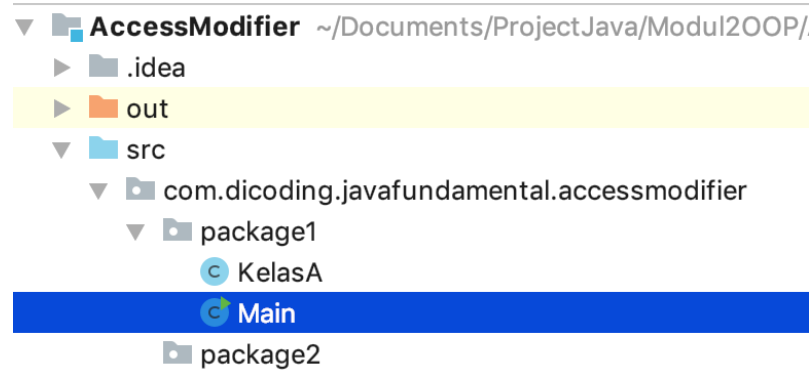
```

5. Pemanggilan `memberB`, `memberC`, dan `functionB` seperti di atas tidak akan menimbulkan kompiler error.

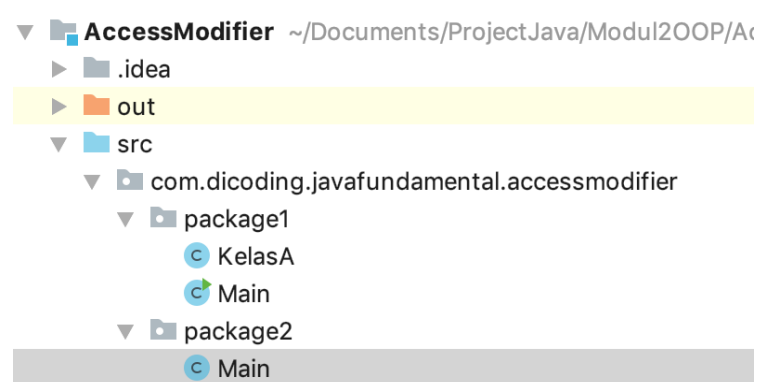
Tapi perlu diingat bahwa access modifier default tidak bisa diakses dari luar package. Sebagai contoh kita akan coba modifikasi dengan menambahkan package pada masing-masing kelas. Buatlah 2 package baru di dalamnya dengan nama `package1` dan `package2`.



6. Kemudian pindahkan `kelasA` dan `Main` ke dalam `package1`.



7. Buatlah kelas baru dengan nama `Main` di dalam `package2`.



8. Selanjutnya di kelas `Main` kita akan mencoba membuat objek `KelasA`. Masukkan kode berikut di dalam kelas `Main`.



Jika diperhatikan kode berikut pasti akan eror:

```
System.out.println(kelasA.functionB());
System.out.println(kelasA.memberB);
System.out.println(kelasA.memberC);
```

Ini terjadi karena `functionB`, `memberB` dan `memberC` tidak dalam keadaan public. Anda bisa beri komentar untuk ketiga kode tersebut menjadi seperti ini:

```
1. package com.dicoding.javafundamental.accessmodifier.package2;
2.
3. import com.dicoding.javafundamental.accessmodifier.package1.KelasA;
4.
5. public class Main {
6.     public static void main(String[] args) {
7.
8.         // Kode ini pasti akan mengalami kompiler error
9.
10.        KelasA kelasA = new KelasA();
11.
12.        //System.out.println(kelasA.functionB());
13.
14.        //System.out.println(kelasA.memberB);
15.        //System.out.println(kelasA.memberC);
```

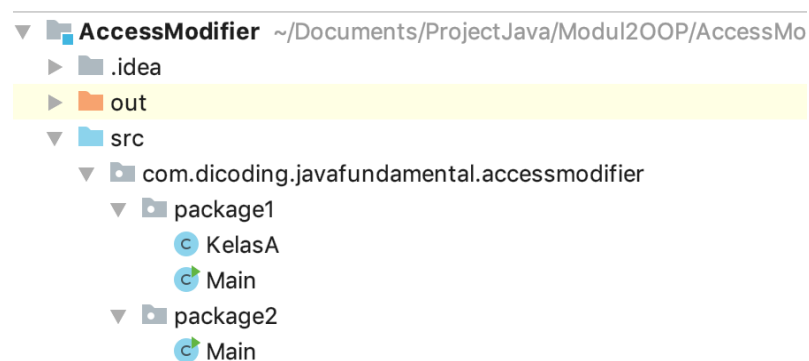
Protected

Access modifier protected bisa diakses selama masih dalam satu package. Protected memiliki sedikit perbedaan dengan default modifier. Perbedaannya adalah protected bisa diakses dari luar package. Akan tetapi, satu-satunya cara untuk akses dari luar package adalah kelas yang hendak mengakses, merupakan kelas turunannya.

Codelab Protected

Kita tambahkan access modifier public terlebih dahulu pada **KelasA** agar bisa di akses dari luar package. Kita juga menambahkan metode baru **methodC** dengan modifiernya sebagai protected.

1. Bukalah kembali proyek AccessModifier.



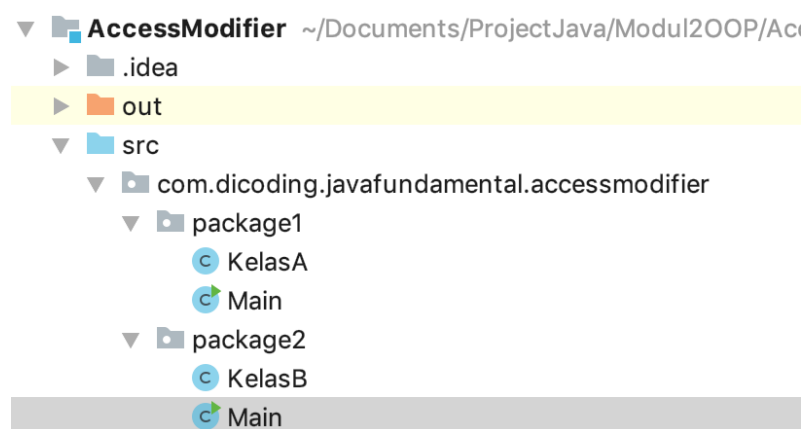
2. Tambahkan **methodC** di dalam kelas **kelasA**:

```

1. package com.dicoding.javafundamental.accessmodifier.package1;
2.
3. public class KelasA {
4.
5.     private int memberA = 5;
6.
7.     char memberB = 'A';
8.     double memberC = 1.5;
9.
10.    private int functionA() {
11.        return memberA;
12.    }
13.
14.    int functionB() {
15.        // Pemanggilan private member dan private function

```

3. Selanjutnya buatlah kelas baru dengan **KelasB** yang merupakan turunan dari **KelasA** di **package2**.



4. Masukkan kode berikut di dalam **KelasB**.

```

1. package com.dicoding.javafundamental.accessmodifier.package2;
2.
3. import com.dicoding.javafundamental.accessmodifier.package1.KelasA;
4.
5. public class KelasB extends KelasA {
6.
7.     @Override
8.     protected void methodC() {
9.         super.methodC();
10.
11.         System.out.println("Contoh pemanggilan protected dari package luar");
12.     }
13. }

```

Kode di atas adalah contoh kelas turunan yang bisa mengakses metode dengan modifier protected dari induknya.

5. Bukalah **Main** di package2 dan tambahkan kode berikut

1. package com.dicoding.javafundamental.accessmodifier.package2;

2.

3. import com.dicoding.javafundamental.accessmodifier.package1.KelasA;

4.

5. public class Main {

6. public static void main(String[] args) {

7.

8. // Kode ini pasti akan mengalami kompiler error

9.

10. KelasA kelasA = new KelasA();

11.

12. //System.out.println(kelasA.functionB());

13.

14. //System.out.println(kelasA.memberB);

15. //System.out.println(kelasA.memberC);

6. Jalankan kelas Main yang ada di package2 , maka hasilnya akan jadi seperti ini:

Percobaan access modifier!!!
Contoh pemanggilan protected dari package luar

Public

Access modifier public bisa kita sebut sebagai modifier global. Artinya bisa diakses dari manapun bahkan package yang berbeda.

Seperti pada contoh kode sebelumnya, KelasA ditambahkan modifier public. Karena modifiernya public maka bisa diakses dari package lainnya.

Ada 4 access modifier yang masing-masing memiliki batasan akses yang berbeda-beda. Berikut ini tabel perbandingan antara keempatnya.

Modifier	Class	Package	Subclass	World
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Dimulai dari private yang hanya memiliki batasan akses dari dalam kelas. Kemudian default modifier yang memiliki batasan dalam satu package. Protected memiliki akses dari luar package selama kelas tersebut merupakan turunannya. Dan terakhir, public, yang dapat diakses secara global.

Non Access Modifier

Java juga mengenal modifier lainnya yaitu tipe non access modifier. Ada beberapa non access modifier yaitu static, final, transient, dan synchronize.

Non access modifier memiliki fungsi yang beragam tergantung dari kebutuhannya. Mari kita bahas satu per satu.

Static

Yang pertama adalah static. Ia digunakan untuk mendeklarasikan variabel atau metode yang berdiri sendiri tanpa perlu instance dari suatu kelas. Ada dua macam non access modifier static yaitu static variable dan static methods.

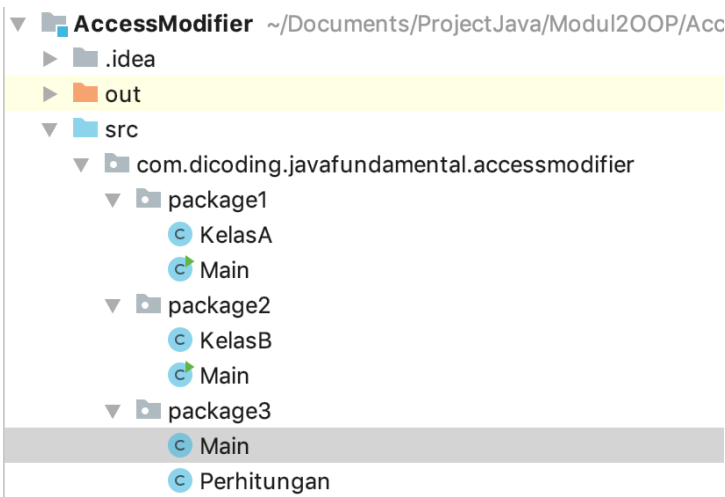
Static Variable

Variabel yang berdiri sendiri tanpa perlu instance dari kelas.

Codelab Static Variable

Kita akan mempraktikkan latihan untuk memperdalam materi satatic variable:

1. Bukalah kembali proyek AccessModifier.
2. Buatlah package baru dengan nama `package3` dan tambahkan kelas `Perhitungan` dan `Main` di dalamnya:



3. Tambahkan kode berikut di dalam kelas `Perhitungan` :

```
1. package com.dicoding.javafundamental.accessmodifier.package3;
2.
3. public class Perhitungan {
4.     public static int nilai = 0;
5. }
```

4. Untuk mengakses kelas perhitungan di atas, kita tidak perlu membuat objeknya di dalam kelas `Main` :

```
1. package com.dicoding.javafundamental.accessmodifier.package3;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         System.out.println("Nilainya adalah " + Perhitungan.nilai);
6.     }
7. }
```

5. Jalankan kelas Main yang ada di `package3`, maka hasilnya akan jadi seperti ini:

Nilainya adalah 0

6. Dan perlu diketahui juga bahwa static variable hanya berjumlah satu, tak peduli berapapun obyek yang dibuat. Sebagai contoh kita modifikasi kelas `Perhitungan` dengan menambahkan konstruktor, seperti di bawah ini.

```
1. package com.dicoding.javafundamental.accessmodifier.package3;
2.
3. public class Perhitungan {
4.     public static int nilai = 0;
5.
6.     Perhitungan() {
7.         nilai++;
8.     }
9. }
```

7. Kemudian tes pembuatan objek kelas perhitungan sebanyak 5 kali, seperti di bawah ini.

```
1. package com.dicoding.javafundamental.accessmodifier.package3;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         System.out.println("Nilainya adalah " + Perhitungan.nilai);
6.
7.         for (int x = 0 ; x < 5; x++){
8.             new Perhitungan();
9.         }
10.
11.         System.out.println("Sampai "+Perhitungan.nilai);
12.     }
13. }
```

8. Jalankan lagi kelas Main yang ada di `package3`, maka hasilnya akan jadi seperti ini:

Nilainya adalah 0

Sampai 5

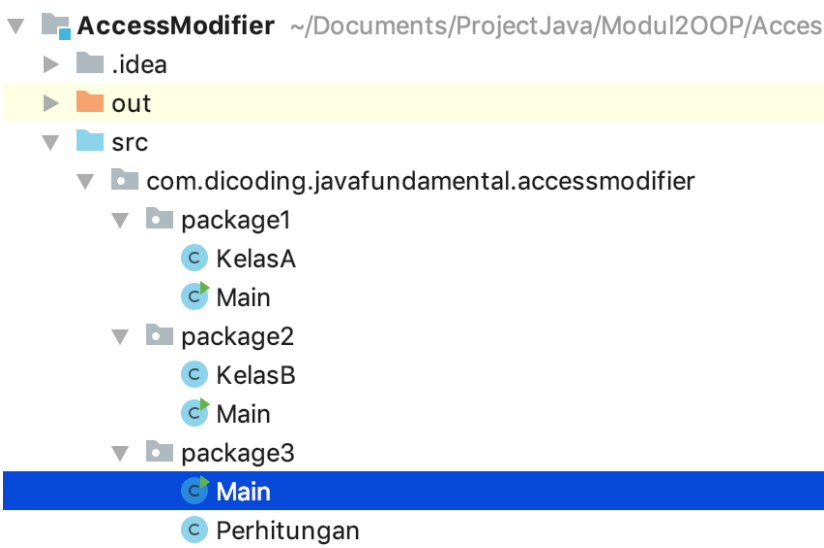
Static Methods

Metode yang berdiri sendiri tanpa perlu instance dari kelas.

Codelab Static Methods

Mari kita coba:

1. Bukalah kembali proyek AccessModifier.



2. Tambahkan kode berikut di dalam kelas **Perhitungan** :

```
1. package com.dicoding.javafundamental.accessmodifier.package3;
2.
3. public class Perhitungan {
4.     public static int nilai = 0;
5.
6.     protected static int getNilai(){
7.         return nilai;
8.     }
9.
10.    Perhitungan() {
11.        nilai++;
12.    }
13. }
```

3. Untuk mengakses kelas perhitungan di atas, kita tidak perlu membuat objeknya di dalam kelas **Main** :

```
1. package com.dicoding.javafundamental.accessmodifier.package3;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         System.out.println("Nilainya adalah " + Perhitungan.nilai);
6.
7.         for (int x = 0 ; x < 5; x++){
8.             new Perhitungan();
9.         }
10.
11.         System.out.println("Sampai "+Perhitungan.nilai);
12.
13.         System.out.println("getNilai memiliki nilai " + Perhitungan.getNilai());
14.     }
15. }
```

4. Jalankan kelas Main yang ada di `package3` , maka hasilnya akan jadi seperti ini:

```
Nilainya adalah 0
Sampai 5
Getnilai memiliki nilai 5
```

Final

Non access modifier kedua adalah final. Ada 3 macam penggunaan non access modifier final yaitu final variabel, final methods dan final class.

Final Variable

Final variabel hanya bisa diinisiasi sekali. Ini menandakan bahwa variabel yang sudah dideklarasikan final tidak dapat diganti dengan objek lainnya.

Perhatikan contoh kode di bawah ini:

```
1. package com.dicoding.javafundamental.accessmodifier.package4;
2.
3. public class Perhitungan {
4.     final int nilai = 5;
5.
6.     void ubahNilai(){
7.         // Kode di bawah akan menampilkan error
8.         nilai = 10;
9.     }
10. }
```

Kode di atas akan eror karena kita mencoba mengganti variabel nilai dengan objek baru.

```
1 package com.dicoding.javafundamental.accessmodifier.package4;
2
3 public class Perhitungan {
4     final int nilai = 5;
5
6     void ubahNilai(){
7         // Kode di bawah akan menampilkan error
8         nilai = 10;
9     }
10 }
```

Cannot assign a value to final variable 'nilai'

Biasanya final digunakan bersamaan dengan static untuk membuat suatu konstanta. Misalnya contoh kode konstanta nilai PI dari lingkaran, seperti di bawah ini.

```
1. package com.dicoding.javafundamental.accessmodifier.package4;
2.
3. public class Lingkaran {
4.     static final double PI = 3.141;
5. }
```

Final Methods

Metode yang dideklarasikan final maka tidak dapat di override oleh anak kelas. Ini akan berguna jika kita ingin membuat metode yang tidak dapat diubah. Contoh kodenya adalah seperti ini.

```
1. package com.dicoding.javafundamental.accessmodifier.package4;
2.
3. public class Lingkaran {
4.     static final double PI = 3.141;
5.
6.     int jari = 7;
7.
8.     final double getLuas() {
9.         return PI * (jari * jari);
10.     }
11. }
```

Final Class

Kelas yang dideklarasikan sebagai final maka tidak bisa dijadikan sebagai induk kelas.




```
1. final class Lingkaran {
2.
3. }
```

Transient

Transient memiliki hubungan dengan proses serialisasi (serializing). Serialisasi adalah proses konversi suatu objek menjadi byte agar dapat ditransmisikan.

Variabel yang dideklarasikan sebagai transient maka akan tidak dijaga nilainya di dalam proses serialisasi. Contoh penggunaannya adalah seperti ini.



```
1. class Test implements Serializable {
2.
3.     // Variabel ini tidak akan dijaga nilainya
4.     transient int nilaiA;
5.
6.     // Variabel ini akan dijaga nilainya
7.     double nilaiB;
8.     String nilaiC;
9.
10. }
```

Synchronized

Synchronized modifier digunakan untuk membatasi akses ke suatu variable/methods yang hanya boleh dilakukan oleh satu thread. Ketika ada 2 thread yang ingin mengakses synchronized variable/methods, maka prosesnya akan dilakukan secara serial (bergantian).

Contoh penggunaannya adalah seperti ini.



```
1. public synchronized void showData() {
2.
3. }
```



Dicoding Space
Jl. Batik Kumeli No.50, Sukaluyu,
Kec. Cibeunying Kaler, Kota Bandung
Jawa Barat 40123

Penghargaan



Decode Ideas
Discover Potential

➤ [Tentang Kami](#)

- [Blog](#)
- [Reward](#)
- [Showcase](#)
- [Hubungi Kami](#)
- [FAQ](#)