

# METHOD DAN CONSTRUCTOR

---

## OBJEKTIF :

1. Mahasiswa Mampu Memahami *Method* dan *Constructor* pada Java.
  2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA untuk Membuat Program dengan *Method* dan *Constructor*.
- 

## 8.1 METHOD

*Method* adalah suatu blok dari program yang berisi kode dengan nama variabel dan nilai yang dapat digunakan kembali. Bentuk umum pendeklarasian dan pendefinisian *method* adalah :

1. Baris pertama deklarasi *method*, disebut *header* metode (*method header*),
2. Diikuti badan metode (*method body*), yang dimuat di dalam pasangan kurung kurawal ({ }).

```
Header_metode {  
    Badan_metode  
}
```

Berikut ini adalah sintaks umum *method* pada Java :

```
modifier typeOfMethod nameOfMethod (Parameter List) {  
    // method body  
}
```

Keterangan :

- *modifier* - *modifier* mendefinisikan jenis akses metode dan bersifat opsional untuk digunakan.
- *typeOfMethod* - *type* ini mendefinisikan apakah metode dapat mengembalikan nilai atau tidak.
- *nameOfMethod* - nama *method* yang akan dibuat
- Daftar Parameter - Daftar parameter adalah jenis tipe data dan variabel, urutan, serta banyaknya jumlah parameter *method*. Daftar parameter bersifat opsional, karena metode dapat berisi parameter nol.
- *method body* - badan metode mendefinisikan apa yang dilakukan metode dengan pernyataan.

Untuk menggunakan metode, metode tersebut harus dipanggil. Terdapat dua cara di mana sebuah metode dipanggil yaitu, metode tidak mengembalikan nilai (*void*) dan metode dapat mengembalikan nilai (*return*). Proses pemanggilan pada metode yaitu ketika sebuah program memanggil metode, kontrol program akan ditransfer ke metode yang dipanggil. Penjelasan beserta contohnya adalah sebagai berikut:

1. *Method* tidak mengembalikan nilai

Kata kunci *void* memungkinkan kita membuat *method* yang tidak mengembalikan nilai.

Contoh:

```
void nama_metode(){  
    System.out.println("Belajar Java");  
}
```

2. *Method* mengembalikan nilai

Jenis kedua adalah jika *method* diberi awalan sebuah tipe data maka *method* tersebut akan memberi nilai balik data yang bertipe data sama dengan *method* tersebut. Penggunaan perintah "*return*" dapat digunakan untuk mengevaluasi ekspresi, kemudian mengirim nilai yang dihasilkan ke pemanggilan *method*.

Contoh :

```
int jumlah(){  
    hasil = nilai1 + nilai2;  
    return hasil; // mengembalikan suatu nilai dari metode  
}
```

### 8.1.1 PEMANGGILAN METHOD

Menulis nama *method* diikuti argumen-argumennya di dalam pasangan tanda kurung ( ) untuk memanggil *method* yang dideklarasikan pada kelas yang sama.

Contoh :

```
hitungNilai();
```

Mengacu ke suatu object maupun suatu class, diikuti dengan titik (.) dan nama method.

Contoh :

```
mhs1.setName("Ani");
```

### 8.1.2 KARAKTERISTIK METHOD

Berikut adalah karakteristik dari *method* :

1. Dapat mengembalikan satu nilai atau tidak sama sekali
2. Dapat diterima beberapa parameter yang dibutuhkan atau tidak ada parameter sama sekali.
3. Setelah *method* telah selesai dieksekusi, dia akan kembali pada *method* yang memanggilnya.

### 8.1.3 PARAMETER PASSING

Saat bekerja di bawah proses pemanggilan, argumen harus diteruskan. Argumen harus dalam urutan yang sama dengan parameternya masing-masing dalam spesifikasi metode. Parameter dapat diteruskan dengan nilai atau referensi. Meneruskan parameter dengan nilai berarti memanggil metode dengan parameter. Melalui ini, nilai argumen diteruskan ke parameter.

### 8.1.4 OVERLOADING TERHADAP METHOD

*Overloading* terhadap metode merupakan fitur penting dan berguna di bahasa Java. *Overloading* metode digunakan untuk metode sama yang melakukan tugas-tugas serupa namun berbeda tipe argumennya.

```
class luasLingkaran {  
  
    // method menghitung luas dengan jari-jari  
    float luas(float r){  
        return (float) (Math.PI * r * r);  
    }  
  
    // method menghitung luas dengan diameter  
    double luas(double d){  
        return (double) (1/4 * Math.PI * d);  
    }  
}
```

### 8.1.5 KEYWORD THIS

*this* adalah kata kunci di Java yang digunakan sebagai referensi ke objek kelas saat ini, dengan *method instance* atau *constructor*. Dengan menggunakan *keyword this*, *programmer* dapat merujuk anggota kelas seperti *constructor*, variabel, dan *method*. Secara umum, kata kunci *this* digunakan untuk :

- Membedakan variabel *instance* dari variabel lokal jika memiliki nama yang sama dalam *constructor* atau *method*.

```
class Mahasiswa {  
    int umur;  
    Mahasiswa(int umur) {  
        this.umur = umur;  
    }  
}
```

- Memanggil satu jenis *constructor* (*constructor parametrized* atau *default*) dari yang lain dalam sebuah kelas. Ini dikenal sebagai *constructor* eksplisit.

```
class Student {  
    int age;  
    Student() {  
        this(20);  
    }  
    Student(int age) {  
        this.age = age;  
    }  
}
```

### 8.1.6 PENGGUNAAN MODIFIER FINAL DI PARAMETER

*Modifier* memberi dampak tertentu pada kelas, *interface*, *method* dan variabel. Java *modifier* terbagi menjadi kelompok berikut:

1. *Modifier* ketampakan (*visibility modifier*), disebut juga *modifier* pengaksesan (*access modifier*) berlaku untuk kelas, *interface*, *method* dan variabel yaitu default, public, protected, private.
2. *Final modifier* berlaku untuk kelas, variabel dan *method*, yaitu final.

Penggunaan *modifier* final di parameter dapat melindungi agar *object* di argumen dijamin tidak pernah diubah oleh *method* dengan memberi *modifier* final di deklarasi *method*.

3. *Static modifier* berlaku untuk variabel dan *method*, yaitu static.

*Modifier* static berarti variabel diasosiasikan dengan kelas dan dipakai bersama objek-objek kelas itu. Variabel static disebut variabel kelas (*class variable*). Kita memanggilnya pada kelas, bukan pada satu objek.

Serupa dengan data statis, kita juga dapat membuat *method* yang hanya bertindak pada kelas, jadi hanya dapat mengakses variabel static saja bukan data milik instan tertentu. Kita mendeklarasikan dengan memberi *modifier* static.

Terdapat kebutuhan variabel atau *method* yang *common* (dipakai bersama) untuk semua objek kelas tertentu. *Modifier* static menspesifikasikan bahwa variabel atau *method* sama untuk semua objek kelas itu. *Method* atau variabel yang memiliki *modifier* static adalah milik kelas. *Method* static hanya dapat mengakses variabel static. *Method* static yang public dapat diakses tanpa perlu menciptakan instan kelas.

4. *Abstract modifier* berlaku untuk kelas dan *method*, yaitu abstract.

*Modifier* abstract berarti mengidentifikasi *method* yang tidak dapat dijalankan dan harus didefinisikan subkelas tidak abstrak dari kelas yang dideklarasikan. *Method* abstract tidak mempunyai badan *method*, sehingga langsung diakhiri dengan titik koma (:).

5. *Synchronized modifier* berlaku untuk *method*, yaitu synchronized.

*Modifier* synchronized untuk menspesifikasikan bahwa metode adalah thread safe. Ini berarti hanya satu jalur eksekusi yang diizinkan di metode synchronized pada satu waktu. Pada lingkungan *multithreaded* seperti Java, dimungkinkan lebih dari satu jalur eksekusi berjalan di kode yang sama. *Modifier* synchronized mengubah aturan ini dengan hanya mengizinkan satu pengaksesan thread tunggal pada satu saat, memaksa thread-thread lain menunggu giliran.

6. *Native modifier* berlaku untuk *method*, yaitu native.
7. *Storage modifier* berlaku untuk variabel, yaitu transient.

## 8.2 CONSTRUCTOR

*Constructor* adalah *method* khusus yang akan dieksekusi pada saat pembuatan objek (*instance*). Biasanya *method* ini digunakan untuk inisialisasi atau mempersiapkan data untuk objek.

*Constructor* terutama digunakan untuk inisialisasi variabel-variabel *instance* kelas serta melakukan persiapan pada suatu objek agar objek itu dapat beroperasi dengan baik. Ketika objek suatu kelas diciptakan, *new* memanggil *constructor* kelas untuk melakukan inisialisasi. Terdapat hal penting mengenai *constructor*, yaitu:

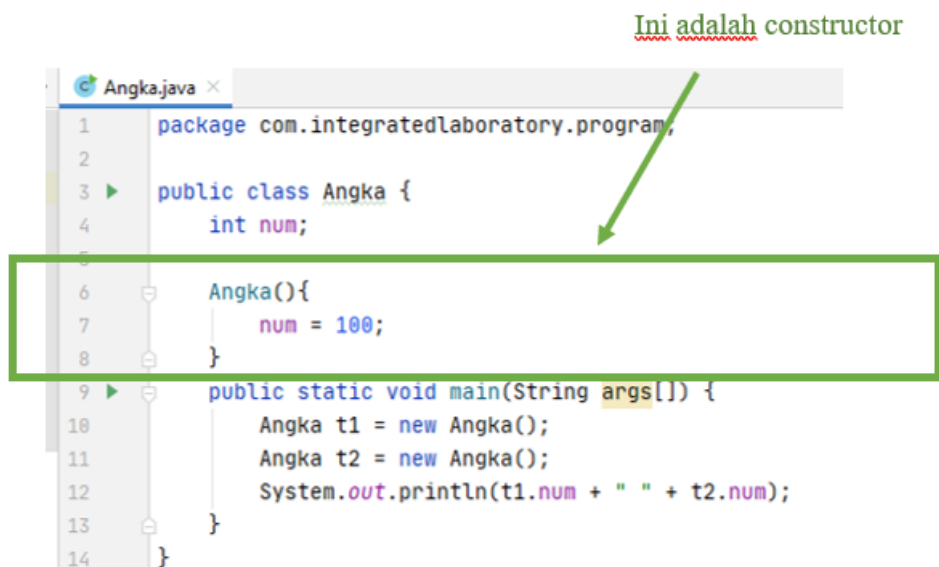
1. *Constructor* harus bernama sama dengan nama kelas (karena *case sensitive* maka nama harus sama dalam huruf kecil dan kapitalnya).
2. *Constructor* tidak menspesifikasikan tipe yang dikirim karena dapat dipastikan tipe yang dikirim bertipe kelas itu.
3. Kelas dapat mempunyai lebih dari satu *constructor*.
4. *Constructor* dapat mempunyai *access modifier*, biasanya *public* namun dapat berupa *private*.
5. *Constructor* dapat mempunyai nol, satu parameter atau lebih.
6. *Constructor* selalu dipanggil dengan operator *new*.

Ketika kelas melakukan inisialisasi, program dapat memberi nilai-nilai (berupa argumen-argumen di *constructor*) inisialisasi. Pada kelas, pasti memiliki sedikitnya satu *constructor*. Jika program tidak mendeklarasikan *constructor*, kompilator secara otomatis menciptakan *constructor* tanpa argumen. Kemudian melakukan inisialisasi variabel-variabel instan ke nilai-nilai inisial yang dideklarasikan atau ke nilai-nilai default (angka 0 untuk tipe-tipe numerik primitif, *false* untuk *boolean* dan *null* untuk *reference* ke objek). Berikut ini adalah sintaks *constructor* :

```
class ClassName {  
    ClassName() {  
    }  
}
```

Berikut ini contoh *constructor* tanpa argumen :

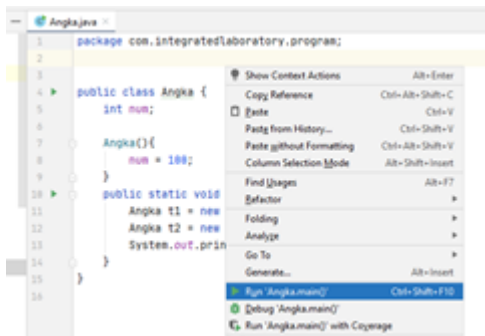
Ini adalah constructor



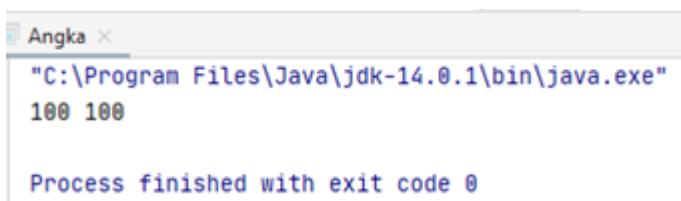
```
1 package com.integratedlaboratory.program;  
2  
3 public class Angka {  
4     int num;  
5  
6     Angka(){  
7         num = 100;  
8     }  
9  
10    public static void main(String args[]) {  
11        Angka t1 = new Angka();  
12        Angka t2 = new Angka();  
13        System.out.println(t1.num + " " + t2.num);  
14    }
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :



Contoh *constructor* yang memiliki parameter :

```
package com.integratedlaboratory.program;

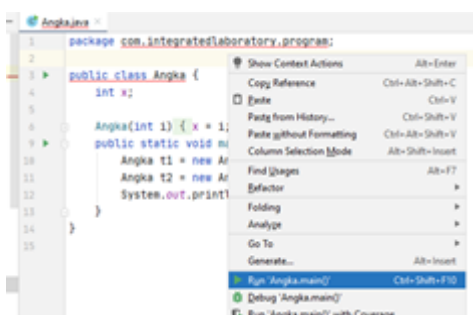
public class Angka {
    int x;

    Angka(int i){
        x = i;
    }


    public static void main(String args[]) {
        Angka t1 = new Angka(15);
        Angka t2 = new Angka(20);
        System.out.println(t1.x + " " + t2.x);
    }
}
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :



```
Angka x
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe"
15 20
Process finished with exit code 0
```

## 8.2.2 OVERLOADING TERHADAP CONSTRUCTOR

Di Java, kita dapat menerapkan *overloading* terhadap *constructor*, yaitu beberapa *constructor* dapat diterapkan pada satu kelas dengan ketentuan jumlah atau jenis argumen di *constructor-constructor* tersebut berbeda-beda.

## REFERENSI

- [1] Hariyanto, Bambang. 2010. Esensi-Esensi Bahasa Pemrograman Java Revisi Ketiga. Bandung: Informatika Bandung.
- [2] Muhardian, Ahmad. 2017. Belajar Java OOP: Mengenal Constructor & Destructor dalam Java. Diambil dari : <https://www.petanikode.com/java-oop-constructor/> . (22 Juli 2020)
- [3] Tutorialspoint. Java-Methods. Diambil dari : [https://www.tutorialspoint.com/java/java\\_method\\_s.htm](https://www.tutorialspoint.com/java/java_method_s.htm) . (23 Juli 2020)
- [4] Tutorialspoint. Java-Constructors. Diambil dari : [https://www.tutorialspoint.com/java/java\\_constructors.htm](https://www.tutorialspoint.com/java/java_constructors.htm) . (6 Agustus 2020)