

MODUL 6. METHOD

6.1 Tujuan

1. Mahasiswa mampu memahami konsep modularitas
2. Mahasiswa mampu mengimplementasikan konsep modularitas dalam bahasa pemrograman.

6.2 Alat dan Bahan

1. Visual Studio Code
2. JDK
3. Github

6.3 Dasar Teori

Program yang kalian tulis lama-kelamaan akan menjadi sangat panjang tentunya. Dan program yang kalian tulis bisa saja memiliki redundansi, atau kode-kode yang mirip satu sama lainnya. Ada, lho cara untuk memisah-misah kode panjang kalian menjadi lebih teratur, dan juga mengurangi redundansi. Salah satu caranya adalah menggunakan fungsi atau *method*.

Fungsi yang ada dalam bahasa pemrograman ini mirip dengan fungsi matematika, di mana ia terdiri dari parameter, proses, dan hasil akhirnya. Dalam konteks pemrograman, fungsi adalah sekelompok kode yang melakukan tugas tertentu dan dapat dipanggil berulang kali, sehingga akan membantu menata kode dengan lebih baik, mengurangi redundansi, dan mempermudah *maintenance* program.

6.3.1 Fungsi dan Method, apa bedanya?

Umumnya, pada bahasa pemrograman lain, kita tidak perlu menulis kelas utama seperti yang kita lakukan pada Java. Contohnya, pada bahasa pemrograman C++, kita hanya perlu menuliskan fungsi *main* atau fungsi utama saja. Namun, karena Java merupakan bahasa pemrograman berorientasi objek, jadi ia memang agak “spesial”, di mana kita harus melakukan semua hal di dalam sebuah kelas.

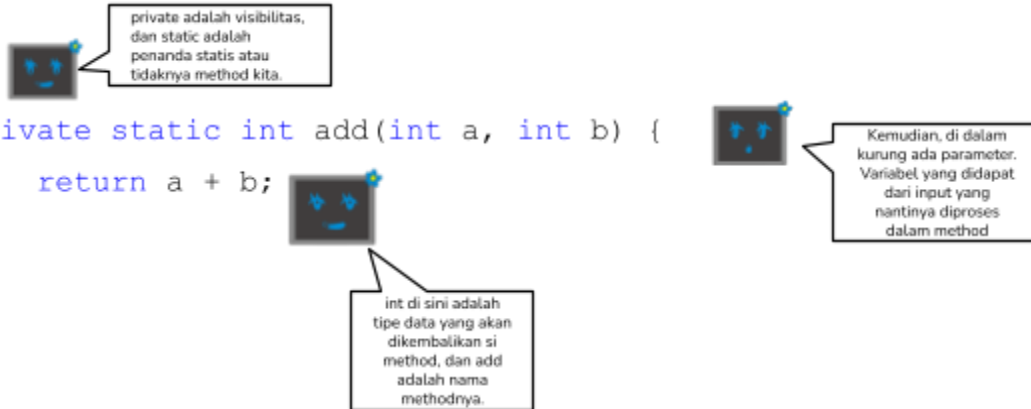
Nah, fungsi ini sebenarnya merupakan panggilan aslinya, yang diturunkan dari fungsi matematika. Di sisi lain, *method* atau metode sebenarnya merupakan fungsi juga, namun lebih khusus, yaitu fungsi yang berada di dalam sebuah kelas. Pada bahasa pemrograman lain, bila ingin membuat fungsi, kita bisa langsung membuatnya tanpa perlu memakai kelas terlebih dahulu. Namun di Java, kita harus membuatnya di dalam kelas sehingga seluruh fungsi di Java juga merupakan sebuah *method*.

Jadi perbedaannya hanyalah pada tempat pembuatannya saja. Dan khususnya di Java, **semua fungsi adalah *method***, jadi kita akan memanggilnya sebagai *method* untuk kejelasan di sini, ya!

6.3.2 Pembuatan Method

Kita pernah membahas *method main* pada bab awal-awal, bukan? Sekarang, kita akan membuat *method* kita sendiri. Kita membutuhkan beberapa hal untuk membuat sebuah *method*. Yang pertama ialah visibilitas, lalu penanda apakah *method* tersebut statis atau tidak, kemudian tipe data, selanjutnya nama *method*, dan terakhir parameter. Untuk lebih jelasnya, mari kita membuat sebuah *method* sederhana untuk menambahkan dua bilangan *integer*.

```
public class ProgramDira {  
    public static void main(String[] args) {  
    }  
    private static int add(int a, int b) {  
        return a + b;  
    }  
}
```



private adalah visibilitas, dan static adalah penanda statis atau tidaknya method kita.

int di sini adalah tipe data yang akan dikembalikan si method, dan add adalah nama methodnya.

Kemudian, di dalam kurung ada parameter. Variabel yang didapat dari input yang nantinya diproses dalam method

Untuk saat ini, kita akan terus menggunakan visibilitas *private* pada *method* yang kita buat sendiri. Karena visibilitas *public* baru akan berguna nanti disaat kita sudah membuat kelas sendiri. Kemudian, mengapa kita butuh *static*? Hal ini dikarenakan kita membuatnya di kelas utama, dan bukan kelas milik kita sendiri. Bila kita tidak menggunakan *static*, maka *method* harus dipanggil lewat objek! Kalian sudah tahu *method* “.nextInt” dan “.equals”, bukan? *Method* “.nextInt” dipanggil dari objek *Scanner* sedangkan *method* “.equals” dipanggil dari objek *String*. Nah, karena kita tidak akan membuat objek dari kelas utama kita, maka kita gunakan *static*. Nantinya kita bisa memanggil *method* kita secara langsung!

6.3.3 Pemanggilan Method

Setelah sebuah *method* dibuat, untuk kita pakai, ia harus dipanggil. Cara memanggilnya adalah dengan memakai nama *method* tersebut, disertai dengan kurung buka dan tutup. Bila ia memiliki parameter, maka harus diisi.

```
public class ProgramDira {  
    public static void main(String[] args) {  
        int angkaPertama = 10;  
        int angkaKedua = 20;  
    }  
}
```

```

    int hasil = add(angkaPertama, angkaKedua);

    System.out.println("Hasil penjumlahan "
        + angkaPertama
        + " dan "
        + angkaKedua
        + " adalah "
        + hasil);
}

private static int add(int a, int b) {
    return a + b;
}
}

```

Bila kode di atas dijalankan, maka kita akan mendapat output berikut.

```

>> "Linux Mint" - rahman @ furude-PC in ~/Projects/Java/Permulaan
● -> java ProgramDira
    Hasil penjumlahan 10 dan 20 adalah 30

```

Nah, kita bisa melihat bahwa pada kode di atas, ada sebuah variabel bernama *hasil* yang menyimpan pemanggilan *method* kita. Jadi, *method* ini memang hanya merupakan rangkaian proses dalam blok yang berbeda. Namun, output atau hasilnya tetap saja sebuah nilai yang kita kenal. Setelah proses selesai, ia akan mengembalikan *hasil*. Betul, mengembalikan. Karena itulah kita memakai keyword *return*, seperti yang kalian lihat. Dan untuk tipe data dari nilai yang dikembalikan, kita menuliskannya sebelum nama *method*. *Method* di atas mengembalikan tipe data *integer*.

Kita juga bisa, loh mengembalikan *String*! Tinggal ubah saja tipe data yang dikembalikannya.

```

private static String greet(String name) {
    return "Halo, " + name + "!";
}

```

Di atas adalah fungsi yang akan menyapa orang yang namanya diberikan pada parameternya.

6.3.3 Parameter vs Argument

Seperti yang kita tahu, parameter ini merupakan variabel yang ditulis dalam kurung *method* yang digunakan sebagai inputan yang akan diproses oleh *method*. Sedangkan Argumen merupakan value yang dikirimkan saat *method* dipanggil. Value/argumen ini nanti di-assign pada paramter didalam *method*.

Singkat cerita yang kita tulis saat **pembuatan method** disebut juga sebagai parameter, sedangkan yang kita masukkan ke dalam parameter saat kita **memanggil method** dipanggil juga sebagai argumen. Coba tebak manakah yang merupakan argumen dan mana yang merupakan parameter pada kode dibawah?

Yap! `int a` dan `int b` yang terletak pada kepala method `printMaximum` merupakan paramater. Sedangkan `num1` dan `num2` saat method `printMaximum` dipanggil merupakan argumen.

```
public class Main {
    // Method untuk mencari dan menampilkan nilai maksimum dari dua angka
    public static void printMaximum(int a, int b) {
        // Menggunakan struktur if-else untuk menentukan dan mencetak nilai
        // maksimum
        if (a > b) {
            System.out.println("Nilai maksimum adalah: " + a);
        } else {
            System.out.println("Nilai maksimum adalah: " + b);
        }
    }

    public static void main(String[] args) {
        // Contoh penggunaan method printMaximum
        int num1 = 5;
        int num2 = 6;

        // Memanggil method printMaximum
        printMaximum(num1, num2);
    }
}
```

6.3.4 Fungsi Void atau Prosedur

Penasaran apa itu *void* yang ada pada *method* utama kita? Harusnya itu kan tipe data seperti integer, String, atau sebagainya, ya kan? Nah, *void* ini merupakan sesuatu yang unik. Ia adalah sebuah tanda untuk sebuah *method*, kalau *method* tersebut **tidak akan mengembalikan apa-apa**. Jadi, *method* tersebut hanya akan dieksekusi untuk melakukan hal-hal yang tidak perlu ada nilai kembalinya, misal mengubah suatu nilai lewat argumen, atau hanya dibutuhkan untuk mencetak sesuatu dengan *println* saja. *Method* seperti ini disebut juga dengan prosedur.

Misal, kita ingin membuat sapaan "Halo, nama!". Berarti yang akan kita lakukan bisa jadi membuat *method* yang mengembalikan *string* seperti sebelumnya, bukan? Namun, bila kita tidak akan menggunakan *string* yang sudah dikembalikan tersebut untuk hal lain, dan hanya butuh untuk dicetak saja, maka penggunaan *void* akan tepat.

```
private static void printHalo(String name) {  
    System.out.println("Hello, " + name + "!");  
}
```

Lihat? Kita hanya butuh dia untuk mencetak. Tidak perlu dikembalikan nilainya.

Demikian, telah selesai bab tentang *method* ini! Menyenangkan, bukan? Kode panjang kalian bisa dipersingkat dengan ini. Nanti kalian akan menggunakan lebih banyak *method* lagi, lho! Sampai bertemu di bab selanjutnya!