Bab 5

Method

Tujuan

1. Memahami Konsep Procedure, Function, dan Method
2. Meningkatkan *Readabilty Code* menggunakan Procedure, Function, dan Method
3. Memahami Penggunaan *method* dengan *Method Overloading*
4. Memahami Konsep dan Teknik *Recursive* dalam Pemrograman

Procedure, *Function*, ataupun *Method* merujuk pada hal yang sama, yaitu kumpulan *statement* yang disatukan menjadi sebuah subprogram. Umumnya ketika sebuah subprogram tidak mengembalikan nilai, maka dikenal sebagai *procedure* sedangkan subprogram yang mengembalikan nilai dikenal sebagai *function*, namun dalam bahasa pemrograman *Object Oriented*, subprogram lebih tepat disebut sebagai *method*.

Pada pemrograman Java, terdapat ***public static void main(String[] args) {}***, dalam kelas utama, bagian ini dikenal sebagai *main method* yang menjadi *entry point* sebuah program, *method-method* yang lain dapat didefinisikan di luar *main method* tetapi masih di dalam *class* yang sama.

5.1 Mendeklarasikan Method

```
modifier returnType namaMethod (paramater/argumen) {
    // body
    return value;
}
```

Drawing 24: Struktur Method Pada Java

- **modifier** merupakan tipe *method* yang menentukan penggunaannya, hal ini bersifat opsional dan akan lebih dalam dibahas dalam *Object Oriented Programming*
- **returnType** merupakan tipe nilai yang dikembalikan, jika *returnType*nya adalah ***void***, maka tidak perlu ada ***return value***
- **methodName** atau *signature* adalah nama *method* yang memiliki standarisasi penulisan seperti variabel

- **parameter/argumen** merupakan nilai-nilai yang dibutuhkan oleh sebuah *method*, bagian ini juga bersifat opsional
- **body** adalah bagian kode dimana *logic method* berjalan ketika *method* dipanggil
- **return value** adalah nilai yang dikembalikan jika *method* tersebut bukan *method void*, nilai ***value*** harus bertipe data ***returnType***

5.2 Memanggil Method

Untuk menggunakan *method*, cukup dengan memanggil nama *method*nya disertai dengan parameter/argumen jika ada.

```
class Main {
    public static void main (String[] args) {
          int[] arr = {7, 3, 9, 1};
          printArray(arr);
          System.out.println(getElement(arr, 2))
    }

    public static void printArray(int[] arr) {
          for (int i = 0; i < arr.length; i++) {
                System.out.print(arr[i] + " ");
          }
          System.out.println();
    }

    public static int getElement(int[] arr, int i) {
          if (i < 0 || i >= arr.length) {
                return 0;
          }
          return arr[i];
    }
}
```

Drawing 25: Penggunaan Method Untuk Memecah Fungsi Program

5.3 Method Overloading

```
class Main {
    public static void main (String[] args) {
        System.out.println(sum(3, 9));
        System.out.println(sum(1, 2, 3, 4, 5, 6, 7));
        System.out.println(sum(3.14, 2.8, 0.97));
    }

    public static int sum (int a, int b) {
        return a + b;
    }

    public static int sum (int ... nums) {
        int result = 0;
        for (int n : nums) {
            result += n;
        }
        return result;
    }

    public static double sum (double ... nums) {
        double result = 0;
        for (double n : nums) {
            result += n;
        }
        return result;
    }
}
```

Drawing 26: 3 Method sum() Dengan Jumlah Parameter dan Return Type yang berbeda

Method Overloading merupakan *method* dengan nama yang sama tetapi dengan jumlah, tipe parameter atau *return type* yang berbeda, hal ini meningkatkan fleksibilitas sebuah *method*.

5.4 Melempar Exception Pada Method

Pada bagian sebelumnya, telah dijelaskan bahwa untuk menangani *exception* terdapat beberapa cara, salah satunya dengan melempar *exception* pada *method*, cara ini berguna untuk menangani *Compile Time Exception* seperti *InterruptedException* dan *IOException* dimana program tidak dapat di*compile* karena *exception* tersebut, dengan melempar *exception* pada *method*, maka *JVM* dapat secara otomatis menangani *exception* tersebut.

```
class Main {
    public static void main (String[] args) {
          int[] arr = {7, 9, 8};
          try {
               System.out.println(getElements(arr, 2));
          } catch (Exception e) {
               System.out.println(e.getMessage());
          }
    }

    public static int getElements(int[] arr, int i)
throws InterruptedException {
          Thread.sleep(3000);
          return arr[i];
    }

}
```

Drawing 27: Method getElements() melempar InterruptedException

Tanpa *throws exception*, program di atas tidak akan bisa di*compile* karena *Thread.sleep()* akan menghasilkan *exception*. Sebuah *method* dapat melempar lebih dari satu *exception*, dengan menyatakan *throws exception* setelah parameter *method* dan dipisahkan dengan koma antara *exception* yang satu dengan *exception* lainnya, metode ini akan sering digunakan pada bab 8 nanti tentang *File I/O*.

5.5 Method Rekursif

Method rekursif merupakan method yang di dalam *body*nya memanggil dirinya sendiri, *method* rekursif biasanya digunakan untuk mempersingkat kode, menyederhanakan implementasi algoritma seperti *tree tarversal*, ataupun demi kode yang '*Elegant*', walaupun rekursif membutuhkan banyak memori dan lebih lambat. Dalam membuat *method* rekursif, ada dua hal yang perlu diperhatikan yaitu :

1. **Rekurens**, atau bagian dari program yang memanggil dirinya sendiri
2. **Basis**, atau nilai yang nantinya menjadi batas dari proses rekursi

```
class Main {
    public static void main(String[] args) {
          System.out.println(fact(8));
          System.out.println(factorial(8))
    }

    public static int fact(int n) {
          if (n == 0) {
                return 1;
          }
          int result = 1;
          for (int i = n; i > 0; i--) {
                result *= i;
          }
          return result;
    }

    // Method Rekursif
    public static int factorial(int n) {
          return n == 0 ? 1 : n * factorial(n – 1);
    }

}
```

Drawing 28: Menghitung Faktorial n menggunakan For Loop dan Rekursi