



**MODUL PEMROGRAMAN BERORIENTASI OBJEK
(CCC210)**

**MODUL 04
FUNGSI**

**DISUSUN OLEH
INDRIANI NOOR HAPSARI, ST, MT**

Universitas
Esa Unggul

**UNIVERSITAS ESA UNGGUL
2020**

MODUL 4 - FUNGSI

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan:

1. Mahasiswa memahami mengapa diperlukan fungsi
2. Mahasiswa memahami tipe fungsi dan cara penggunaannya
3. Mahasiswa dapat membuat fungsi secara tepat dalam bahasa pemrograman

B. Outline Topik

1. Pengertian Fungsi.....	2
2. Tipe Fungsi.....	3
3. Deklarasi dan Pemanggilan Fungsi pada Kelas.....	5
4. Parameter Fungsi.....	8
5. Fungsi Rekursif.....	9



C. Uraian

1. Pengertian Fungsi

Sebuah program dapat dibagi menjadi beberapa bagian sub-program. Setiap sub-program dapat kita beri nama dan dituliskan terpisah dalam sebuah fungsi. Fungsi berguna untuk menstrukturkan program agar lebih mudah dibaca dan dapat diguna ulang dengan memanggil nama fungsi tanpa harus menuliskan ulang kode program secara lengkap.

Dalam merancang sebuah program, kita perlu membagi algoritma ke dalam beberapa sub-bagian, kemudian mendekomposisikannya lagi ke sub-bagian yang lebih kecil, dan seterusnya. Metode ini dikenal dengan istilah perancangan “top-down”. Dengan menggunakan metode “top-down”, program dipecah menjadi beberapa sub-bagian yang lebih sederhana. Dengan cara ini, program menjadi lebih mudah untuk dibaca dan dipahami, lebih mudah diubah, lebih mudah ditulis, diuji, dan juga di-debug. Dengan menggunakan fungsi, kita dapat menstrukturkan program kita dengan cara yang lebih modular. **Fungsi adalah** sekumpulan *statements* yang dieksekusi ketika dipanggil di bagian tertentu dalam program. Berikut ini adalah cara mendeklarasikan sebuah fungsi dalam bahasa Java.

```
public type name ( parameter1, parameter2, ... )
{
    statements...;
}
```

Keterangan

type: tipe data yang dikembalikan oleh fungsi

name: identifier (nama) yang digunakan untuk memanggil fungsi.

parameters (as many as needed): variabel yang menerima nilai argumen. Setiap parameter terdiri atas sebuah tipe data yang diikuti dengan identifier, seperti halnya deklarasi variabel (misal: int x), dan merupakan variabel lokal dari sebuah fungsi. Parameter memungkinkan *passing* argumen ke dalam fungsi saat fungsi tersebut dipanggil. Parameter dapat lebih dari satu, dipisahkan dengan koma.

statements: implementasi/badan fungsi. *Statements* dituliskan di dalam braces { }

2. Tipe Fungsi

Terdapat dua buah tipe fungsi, yaitu

1. Fungsi yang mengembalikan nilai (*returning value*)
2. Fungsi yang tidak mengembalikan nilai (*non-returning value*)

1. Fungsi yang mengembalikan nilai (returning value)

Fungsi yang mengembalikan nilai akan mengirimkan nilai sebagai hasil operasi fungsi. Berlawanan dengan parameter yang merupakan input sebuah fungsi, nilai yang dikembalikan merupakan **output** dari sebuah fungsi. Parameter mengirimkan informasi dari pemanggil fungsi ke dalam fungsi. Nilai kembalian mengirimkan informasi keluar dari fungsi ke pemanggilnya. Pada fungsi yang mengembalikan nilai, pemanggilan terhadap fungsi dapat dituliskan sebagai bagian dari sebuah ekspresi.

Berikut adalah contoh deklarasi sebuah fungsi untuk menghitung luas segi empat. Fungsi `area` menerima dua buah parameter yaitu **pLong** dan **pWidth** yang masing-masing memiliki tipe data `float`. Pada contoh berikut, tipe fungsi adalah **float**, yang menunjukkan bahwa nilai kembalian dari fungsi (yaitu hasil perhitungan luas) bertipe `float`. *Keyword* “**static**” digunakan apabila kita ingin mendefinisikan fungsi yang dapat langsung dipanggil secara publik tanpa perlu menginstansiasi objek.

```
public static float area (float pLong, float pWidth)
{
    return pLong * pWidth;
}
```

Berikut ini adalah contoh pemanggilan fungsi di program utama. Pemanggilan fungsi dapat dituliskan sebagai bagian dari sebuah ekspresi, dimana hasil perhitungan luas akan dikembalikan ke pemanggilnya, dan nilai nya di-*assign* ke variabel **mLuas**. Pada akhir program, maka variabel **mLuas** akan bernilai 20.

```
public static void main(String[] args)
{
```

```
float mLuas = area(4,5);
```

2. Fungsi yang tidak mengembalikan nilai (void/non-returning value)

Fungsi yang tidak mengembalikan nilai dibuat dan digunakan seperti fungsi yang mengembalikan nilai, namun fungsi tersebut tidak mengembalikan nilai setelah dieksekusi. Fungsi yang tidak mengembalikan nilai menggunakan keyword “void”, dan dikenal juga dengan istilah fungsi void. Fungsi void menjalankan instruksi, dan kemudian kembali ke pemanggilnya setelah instruksi selesai dieksekusi. Walaupun tidak dituliskan *return* pada akhir badan fungsi, kontrol akan mengembalikan ke pemanggilnya secara otomatis di akhir fungsi. Fungsi void juga dapat menerima beberapa parameter seperti fungsi yang mengembalikan fungsi. Contoh umum penggunaan fungsi void adalah fungsi untuk mencetak teks ke layar. Berikut ini adalah contoh deklarasi fungsi print dalam bahasa Java untuk mencetak “hello world” ke layar.

```
public static void printHello(String pUsername)
{
    System.out.println("hello world, " + pUsername);
}
```

Karena fungsi void tidak mengembalikan nilai setelah dieksekusi, berbeda dengan fungsi yang mengembalikan nilai, fungsi void tidak dapat dipanggil sebagai bagian dari sebuah ekspresi. Pemanggilan terhadap fungsi void harus berupa statement sendiri yang lengkap. Berikut adalah contoh pemanggilan fungsi void pada program utama. Program berikut ini akan menampilkan “hello world, Lila” ke layar.

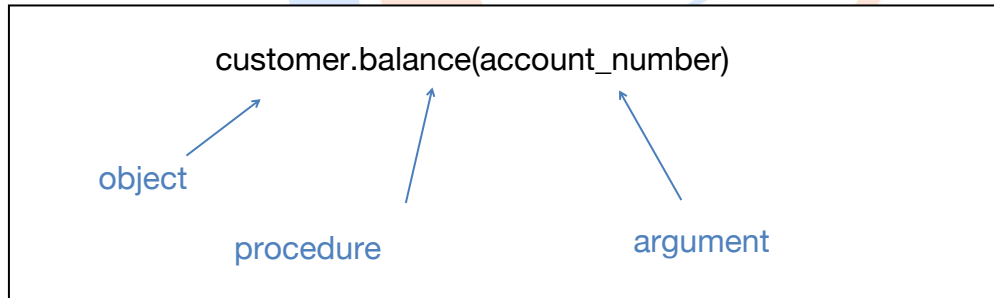
```
public static void main(String[] args)
{
    printHello("Lila");
}
```

3. Deklarasi dan Pemanggilan Fungsi pada Kelas

Paradigma pemrograman berorientasi objek dibangun diatas pemrograman terstruktur. Pada OOP fungsi-fungsi didefinisikan di dalam sebuah kelas bersama dengan datanya. Dengan demikian, deklarasi fungsi pada OOP ditulis di dalam sebuah kelas, dan dipanggil dengan mengacu pada objek yang memiliki definisi fungsi tersebut.

OOP terdiri atas sekumpulan objek yang saling berkomunikasi satu dengan lainnya. Komunikasi antar objek dilakukan dengan saling mengirimkan pesan yang dikenal dengan istilah “*message passing*”. Sebuah pesan untuk sebuah objek merupakan sebuah permintaan eksekusi dari prosedur, yang akan memanggil sebuah fungsi di objek penerima yang kemudian akan mengenerate hasil yang diharapkan.

Berikut ini adalah cara memanggil fungsi “*balance*” yang dimiliki oleh objek “*customer*” dengan memberikan informasi “*account_number*” sebagai argumen (nilai yang diberikan ke fungsi).



Sebagai contoh, didefinisikan sebuah kelas bernama **Rectangle** yang memiliki atribut **panjang** dan **lebar**. Kelas Rectangle membatasi akses data dari program eksternal dengan memberikan akses **private** pada datanya. Program eksternal hanya dapat mengakses fungsi-fungsi yang diset “**publik**”, misalkan, fungsi menghitung luas (**area**), dan mengeset atribut panjang dan lebarnya (**set_values**). Berdasarkan deskripsi tersebut, maka implementasi kelas Rectangle secara sederhana dalam bahasa pemrograman Java ditunjukkan pada kode program berikut ini.

```

public class Rectangle
{
    private float panjang;
    private float lebar;

    public Rectangle(float pLong, float pWidth)
    {
        set_values(pLong, pWidth);
    }

    public void set_values(float pLong, float pWidth)
    {
        panjang = pLong;
        lebar = pWidth;
    }

    public float area()
    {
        return panjang*lebar;
    }
};

```

Pada contoh di atas, kelas Rectangle memiliki dua buah fungsi, yaitu set_values dan area. Fungsi **set_values** merupakan fungsi void yang tidak mengembalikan nilai, yang berfungsi mengubah nilai **panjang** dan **lebar** dari kelas **Rectangle**. Sedangkan fungsi **area** merupakan fungsi yang mengembalikan nilai, dimana nilai yang dikembalikan yaitu hasil kalkulasi perhitungan luas. Adapun contoh pemanggilan fungsi yang dimiliki oleh kelas Rectangle di program utama adalah sebagai berikut.

```
public static void main(String[] args)
{
    Rectangle R1(4,5);
    float luas = R1.area();
    System.out.println( "Luas=" + luas); //nilai luas = 20
    R1.set_values(5,9);
    luas = R1.area();
    System.out.println( "Luas=" + luas); //nilai luas = 45
}
```



4. Parameter Fungsi

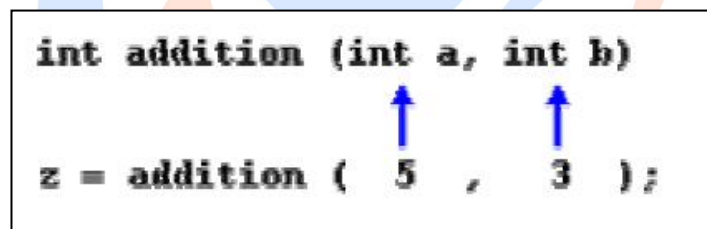
Terdapat dua tipe parameter fungsi, yaitu

1. **Value Parameter**: parameter formal menerima salinan nilai parameter aktual
2. **Reference Parameter**: parameter formal menerima lokasi (memory address) dari parameter aktual

1. Value Parameter

Pada *value parameter*, nilai dari parameter aktual disalin ke dalam parameter formal. *Value parameter* memiliki salinan data sendiri. Selama fungsi dijalankan, value parameter memanipulasi salinan data yang disimpan di ruang memori tersendiri. *Value parameter* dapat menerima ekspresi/ konstanta dari parameter aktual ketika fungsi dipanggil.

Berikut adalah contoh ilustrasi *passing parameter by value* dalam bahasa C++.



```
int addition (int a, int b)

      ↑           ↑
z = addition ( 5 , 3 );
```

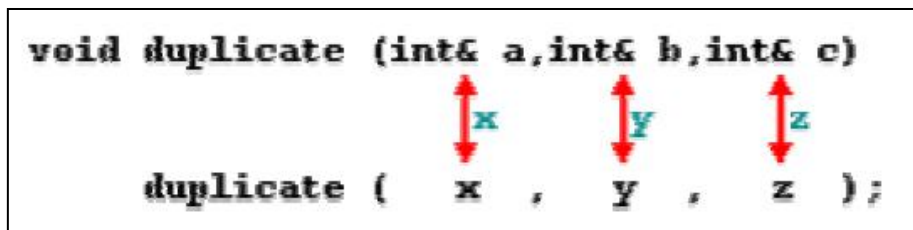
Ketika fungsi **addition** dipanggil, maka nilai dari variabel lokal **a** dan **b** menjadi **5** dan **3** berurutan. Modifikasi terhadap **a** dan **b** di dalam fungsi **addition** tidak akan mengubah nilai aktual **a** dan **b** di luar fungsi, karena variabel **a** dan **b** tidak di-*passing* ke fungsi, melainkan hanya salinan nilai dari variabel **a** dan **b** saja yang di-*passing* saat fungsi dipanggil.

2. Reference Parameter

Pada reference parameter, formal parameter menerima dan menyimpan alamat dari parameter aktual. Selama fungsi dijalankan, alamat yang disimpan di reference parameter dapat memodifikasi data dari parameter aktual, karena mengacu pada ruang memori yang sama. Reference parameter dapat mengubah nilai dari parameter aktual (bukan salinannya). Reference parameter berguna untuk tiga kondisi berikut ini

- a) Mengembalikan nilai lebih dari satu.
- b) Mengubah nilai dari parameter aktual
- c) Untuk *passing* data berukuran besar, sehingga *mem-passing* alamat parameter akan menghemat penggunaan memori daripada menyalin nilai seluruh parameter aktual ke parameter formal.

Reference parameter tidak dapat menerima ekspresi/ konstanta dari parameter aktual ketika fungsi dipanggil, melainkan **hanya menerima variabel saja**. Berikut adalah ilustrasi *passing parameter by reference* dalam bahasa C++. Pada Java, seluruh variabel bertipe data bentukan, *array*, dan *collection* akan di-*passing by reference*, sedangkan konstanta atau variabel bertipe data primitif akan di-*passing by value*.



```
void duplicate (int& a, int& b, int& c)
                x    y    z
duplicate (  x  ,  y  ,  z  );
```

5. Fungsi Rekursif

Fungsi rekursif adalah properti dimana fungsi harus dipanggil oleh dirinya sendiri. Rekursif sangat berguna untuk beberapa task, seperti mengurutkan, atau menghitung faktorial. Sebagai contoh, untuk mendapatkan hasil dari n faktorial (n!), maka rumus matematikanya adalah sebagai berikut

$$n! = n * (n-1) * (n-2) * (n-3) \dots * 1$$

Sehingga, untuk menghitung 5 faktorial, maka

$$5! = 5 * 4 * 3 * 2 * 1$$

Dan fungsi rekursif untuk melakukan perhitungan faktorial adalah sebagai berikut.

```
public static long factorial(long a)
{
    if (a > 1)
    {
        return (a * factorial(a-1));
    } else
    {
        return 1;
    }
}
```

Universitas
Esa Unggul

D. Latihan

1. Buat program Java sederhana dengan menggunakan fungsi untuk menghitung usia berdasarkan tahun lahir dan tahun sekarang

Jawaban

```
import java.util.*;

public class BasicJava{
    public static int hitungUsia(int thLahir, int thSkr)
    {
        return thSkr - thLahir;
    }
    public static void main(String[] args)
    {
        Scanner console = new Scanner(System.in);
        System.out.print("Masukkan tahun lahir
Kaila:");
        int tahunLahir = console.nextInt();
        int tahunSekarang = 2020;
        int usia = hitungUsia(tahunLahir,
tahunSekarang);
        System.out.print("Kaila lahir di tahun " +
tahunLahir + ". ");
        System.out.println("Sekarang sudah tahun " +
tahunSekarang);
        System.out.println("Berarti, usia Kaila
sekarang adalah " + usia + " tahun");
    }
}
```

2. Fungsi berguna untuk menstrukturkan program agar lebih mudah dibaca dan dapat diguna ulang dengan memanggil nama fungsi tanpa harus menuliskan ulang kode program secara lengkap.

- a) Benar
- b) Salah
- c) Tidak diketahui

3. Pemanggilan *returning value function* dapat dituliskan sebagai bagian dari sebuah ekspresi

- a) Benar
- b) Salah
- c) Tidak diketahui

4. Pemanggilan *non-returning value function* dapat dituliskan sebagai bagian dari sebuah ekspresi

- a) Benar
- b) Salah
- c) Tidak diketahui

5. *Non-returning value function* mengembalikan tipe void, sehingga dikenal juga dengan istilah fungsi void

- a) Benar
- b) Salah
- c) Tidak diketahui

6. *Non-returning value function* bisa digunakan untuk membuat fungsi mencetak teks ke layar

- a) Benar
- b) Salah
- c) Tidak diketahui

7. Pada Java, *passing parameter by value* terjadi ketika nilai konstanta di-*passing* ke sebuah *method*

- a) Benar
- b) Salah
- c) Tidak diketahui

8. Pada Java, *passing parameter by value* terjadi ketika variabel dengan tipe data primitif di-*passing* ke sebuah *method*

- a) Benar
- b) Salah
- c) Tidak diketahui

9. Pada Java, *passing parameter by value* terjadi ketika variabel tipe data bentukan di-*passing* ke sebuah *method*

- a) Benar
- b) Salah
- c) Tidak diketahui

10. Pada Java, *passing parameter by reference* terjadi ketika nilai konstanta di-*passing* ke sebuah *method*

- a) Benar
- b) Salah
- c) Tidak diketahui

11. Pada Java, *passing parameter by reference* terjadi ketika variabel dengan tipe data primitif di-*passing* ke sebuah *method*

- a) Benar
- b) Salah
- c) Tidak diketahui

12. Pada Java, *passing parameter by reference* terjadi ketika variabel tipe data bentukan di-*passing* ke sebuah *method*

- a) Benar
- b) Salah
- c) Tidak diketahui

13. Reference parameter berguna untuk mengembalikan nilai lebih dari satu.

- a) Benar
- b) Salah
- c) Tidak diketahui

14. Reference parameter berguna untuk mengubah nilai dari parameter aktual.

- a) Benar
- b) Salah
- c) Tidak diketahui

15. Reference parameter berguna untuk *passing* data berukuran besar, sehingga mem-*passing* alamat parameter akan menghemat penggunaan memori daripada menyalin nilai seluruh parameter aktual ke parameter formal.

- a) Benar
- b) Salah
- c) Tidak diketahui

E. Daftar Referensi

Walter Savitch, *Problem Solving with C++*, Pearson International Edition, 2006.

[http://www.mu.ac.in/myweb_test/MCA study material/](http://www.mu.ac.in/myweb_test/MCA_study_material/)

<http://www.cs.fsu.edu/~xyuan/cop3330/>

<https://www.programiz.com/cpp-programming>

buildingjavaprograms.com

