Modul 2 : Konsep Dasar Pemrograman Berorientasi Objek

2.1 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. **Mengerti konsep** dasar *object oriented programming*
2. Dapat **membandingkan** antara **pemrograman prosedural** dengan **pemrograman berorientasi object**.
3. **Memahami Class, Object, Method** dan **Constructor**, serta **mengimplementasikannya** dalam suatu program java

2.2 Pendahuluan

Sebelum membahas tentang kelas terlebih dahulu kita perlu mengetahui konsep pemrograman berorientasi objek. Suatu sistem yang dibangun dengan metode berorientasi objek adalah sebuah **sistem** yang komponennya di-enkapsulasi menjadi kelompok data dan fungsi, yang dapat mewarisi atribut dan sifat dari komponen lainnya dan komponen-komponen tersebut saling berinteraksi satu sama lain. [*Meyer,1997*]

Beberapa karakteristik utama dalam pemrograman berorientasi objek yaitu :

2.2.1 Abstraksi

Abstraksi pada dasarnya adalah menemukan **hal-hal** yang **esensial** pada suatu **objek** dan **mengabaikan** hal-hal yang sifatnya **insidental**. Kita menentukan apa **ciri-ciri(atribut)** yang dimiliki oleh objek tersebut serta apa saja yang bisa **dilakukan(fungsi)** oleh objek tersebut.

Contohnya adalah abstraksi pada manusia.
Ciri-ciri (atribut) : punya tangan, berat, tinggi, dll. (kata benda, atau kata sifat)
Fungsi (perilaku): makan, minum, berjalan, dll. (kata kerja)

2.2.2 Enkapsulasi

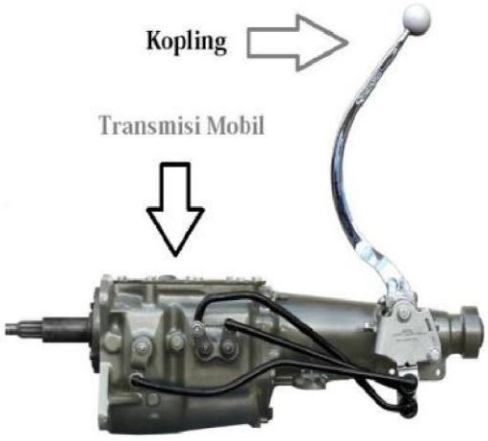
Pengkapsulan adalah proses **pemaketan** data **objek** bersama **method-methodnya**. **Manfaat** utama pengkapsulan adalah **penyembunyian** rincian-rincian **implementasi** dari **pemakai/objek lain** yang **tidak** berhak. Enkapsulasi **memprotek** suatu **proses** dari **kemungkinan interferensi** atau **penyalahgunaan** dari **luar sistem**.

Fungsi dari enkapsulasi adalah:

a. Penyembunyian data
   Penyembunyian data (data hiding) mengacu **perlindungan data internal objek**. Objek tersebut disusun dari antarmuka **public method** dan **private data**. Manfaat utama adalah bagian **internal** dapat **berubah tanpa mempengaruhi** bagian-bagian **program** yang **lain**.

b. Modularitas
   Modularitas (*modularity*) berarti **objek** dapat **dikelola** secara **independen**. Karena **kode** sumber bagian **internal** objek **dikelola** secara **terpisah** dari **antarmuka**, maka kita **bebas** melakukan **modifikasi** yang **tidak menyebabkan masalah** pada bagian-bagian lain. Manfaat ini **mempermudah mendistribusikan objek-objek** di **sistem.**

Contoh ilustrasi pada mobil.



Mobil memiliki sistem transmisi dan sistem ini menyembunyikan proses yang terjadi di dalamnya tentang bagaimana cara ia bekerja, mulai dari bagaimana cara ia mengatur percepatan dan apa yang dilakukan terhadap mesin untuk mendapat percepatan tersebut.

Kita sebagai pengguna hanya cukup memindah-mindahkan tongkat transmisi (kopling) untuk mendapatkan percepatan yang diinginkan. Tongkat transmisi adalah satu-satunya interface yang digunakan dalam mengatur sistem transmisi mobil tersebut. Kita tidak dapat menggunakan pedal rem untuk mengakses sistem transmisi tersebut dan sebaliknya dengan mengubah-ubah sistem transmisi kita tidak akan bisa menghidupkan radio mobil atau membuka pintu mobil. Konsep yang sama dapat pula kita terapkan dalam pemrograman orientasi objek.

2.2.3 Pewarisan (Inheritance)

Pewarisan adalah proses **penciptaan class baru** (yang disebut **subclass** atau **kelas turunan**) dengan **mewarisi karakteristik** dari **class yang telah ada** (**superclass** atau **kelas induk**), ditambah **karakteristik unik class baru** itu. **Karakteristik unik** tersebut bisa merupakan **perluasan** atau **spesialisasi** dari **superclass**.Kelas turunan akan **mewarisi** anggota-anggota suatu **kelas** yang berupa **data (atribut)** dan **fungsi (operasi)** dan pada **kelas turunan** memungkinkan **menambahkan data** serta **fungsi yang baru**. Secara praktis berarti bahwa jika superclass telah mendefinisikan perilaku yang kita perlukan, maka kita tidak perlu mendefinisi ulang perilaku itu, kita cukup membuat class yang merupakan subclass dari superclass yang dimaksud.

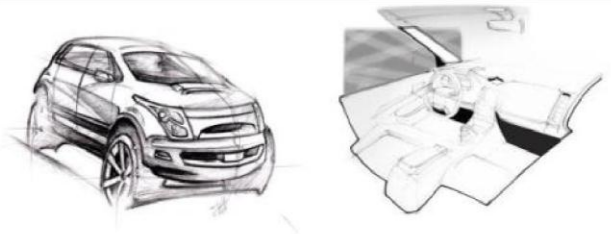
2.2.4 Reuseability

Reuseability adalah **kemampuan** untuk **menggunakan kembali** kelas yang sudah ada. Karakteristik ini dimiliki oleh OOP, sehingga kita tidak perlu membuat ulang definisi perilaku jika perilaku tersebut sudah ada di suatu class lain.

2.2.5 Polymorphisms

Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk. Konsep ini memungkinkan digunakannya **antarmuka** (interface) yang sama untuk **memerintah** suatu **objek** agar dapat **melakukan aksi** atau **tindakan** yang mungkin secara **prinsip sama** tapi secara **proses berbeda**. Seringkali ***Polymorphism*** disebut dengan "**satu interface banyak aksi**". Mekanisme *Polymorphism* dapat dilakukan dengan dengan beberapa cara, seperti **overloading method**, **overloading constructor**, maupun **overriding method**. Semua akan dibahas pada bab selanjutnya.

Contoh ilustrasi pada mobil. Mobil yang ada di pasaran terdiri atas berbagai tipe dan merek, tetapi semuanya memilki interface kemudi yang hampir sama seperti setir, tongkat transmisi, pedal gas, dll. Jika kita dapat mengemudikan satu jenis mobil saja dari satu merek tertentu, maka boleh dikatakan kita dapat mengemudikan hampir semua jenis mobil yang ada karena semua mobil tersebut menggunakan interface yang sama. Misal, ketika kita menekan

	pedal gas pada mobil A, kita telah berhasil menggerakkan mobil A tersebut dengan percepatan yang tinggi. Sebaliknya, ketika kita menggunakan mobil B dan sama-sama menekan pedal gas, mobil B ini bergerak agak lambat. Jadi, dapat disimpulkan dengan **sama-sama** kita **menekan pedal gas** pada dua mobil ini akan didapatkan **hasil** yang **berbeda** pada keduanya.

2.2.6 Message (Komunikasi antar objek)

Objek yang bertindak sendirian jarang berguna, kebanyakan objek memerlukan objek-objek lain untuk melakukan banyak hal. Oleh karena itu**, objek-objek** tersebut **saling berkomunikasi** dan **berinteraksi lewat *message***. Ketika berkomunikasi, suatu objek mengirim pesan (dapat berupa pemanggilan method) untuk memberitahu agar objek lain melakukan sesuatu yang diharapkan. Seringkali pengiriman *message* juga disertai informasi untuk memperjelas apa yang dikehendaki. Informasi yang dilewatkan beserta *message* adalah **parameter message.**

2.3 Class

Class adalah **cetak biru** (rancangan) dari **objek**. Ini berarti kita bisa membuat **banyak objek** dari **satu** macam **class. Class mendefiniskan** sebuah **tipe dari objek**. Di dalam **class** kita dapat **mendeklarasikan variabel** dan **menciptakan object(instansiasi)**. Sebuah class mempunyai **anggota(member)** yang terdiri atas **atribut** dan **method**. **Atribut** adalah semua **field identitas** yang kita berikan pada suatu class, misal class manusia memiliki field atribut berupa nama, maupun umur. **Method** dapat kita artikan sebagai semua **fungsi** ataupun **prosedur** yang merupakan **perilaku (behaviour)** dari suatu **class**. Dikatakan **fungsi** bila method tersebut melakukan suatu **proses** dan **mengembalikan suatu nilai** (return value), dan dikatakan **prosedur** bila method tersebut **hanya melakukan** suatu **proses** dan **tidak mengembalikan nilai (void).**

Contoh pendeklarasian kelas:

```
class <classname>
{
    //declaration of data member
    //declaration of methods
}
```

Kata **`class`** merupakan ***keyword*** pada Java yang digunakan untuk **mendeklarasikan kelas** dan `<classname>` digunakan untuk memberikan nama kelas. Sebagai contoh, dibawah ini terdapat pendeklarasian kelas Mahasiswa yang mempunyai field nama, nim dan method getNamaMahasiswa().

```
class Mahasiswa
{
    String nama;            //data anggota
    int nim;                //data anggota
    void getNamaMahasiswa() {};   //method
}
```

Setelah mengetahui apa itu kelas , selanjutnya kita perlu mengetahui **Object** dan **Java Modifier**

2.3.1 Object

Object (objek) secara lugas dapat diartkan sebagai **insatansiasi** atau **hasil ciptaan** dari suatu **class** yang telah dibuat sebelumnya. Dalam pengembangan program orientasi objek lebih lanjut, sebuah objek dapat dimungkinkan terdiri atas objek-objek lain. Seperti halnya objek mobil terdiri atas mesin, ban, kerangka mobil, pintu, karoseri dan lain-lain. Atau, bisa jadi sebuah objek merupakan turunan dari objek lain sehingga mewarisi sifat-sifat induknya. Misal motor dan mobil merupakan kendaraan bermotor, sehingga motor dan mobil mempunyai sifat-sifat yang dimiliki oleh class kendaraan bermotor dengan spesifikasi sifat-sifat tambahan sendiri.

Contoh object yang lain : Komputer, TV, mahasiswa, ponsel, Ibuku, dan lainnya.

Contoh pembuatan Objek dalam java:

```
Manusia objMns1 = new Manusia();//membuat objek manusia
```

2.3.2 Java Modifier

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

1. ***Access modifier*** berlaku untuk class, method dan variabel, meliputi *modifier* **public, protected, private,** dan default (tak ada modifier).
2. ***Final modifier*** berlaku class, method dan variabel, meliputi *modifier* **final.**
3. ***Static modifier*** berlaku untuk variabel dan method, meliputi *modifier* **static.**
4. ***Abstract modifier*** berlaku untuk class dan method, meliputi *modifier***abstract.**
5. ***Syncronized modifier*** berlaku untuk method, meliputi *modifier* **syncronized .**
6. ***Native modifier*** berlaku untuk method, meliputi *modifier* **native.**
7. ***Storage modifier*** berlaku untuk variabel, meliputi **transient** dan **volatile.**

Berikut ini modifier-modifier yang terdapat pada java :

Modifier	Class dan Interface	Method dan Variabel
Default (tak ada *modifier*) **Friendly**	Tampak di Paketnya	Diwarisi oleh subclassnya di paket yang sama dengan classnya. Dapat diakses oleh method-method di class-class yang sepaket.
Public	Tampak di manapun	Diwarisi oleh semua subclassnya. Dapat diakses dimanapun.
protected	Tidak dapat diterapkan	Diwarisi oleh semua subclassnya. Dapat diakses oleh method-method di class-class yang sepaket.
Private	Tidak dapat diterapkan	Tidak diwarisi oleh subclassnya Tidak dapat diakses oleh class lain.

Permitted Modifier:

Modifier	Class	Interface	Method	Variabel
Abstract	Class dapat berisi method abstract. Class tidak dapat diinstantiasi **Tidak mempunyai constructor**	Optional untuk diberikan di interface karena interface secara inheren adalah abstract.	Tidak ada badan method yang didefinisikan. Method memerlukan class kongkret yang merupakan subclass yang akan mengimplementasikan method abstract	Tidak dapat diterapkan.
Final	Class menjadi tidak dapat digunakan untuk menurunkan class yang baru.	Tidak dapat diterapkan.	Method tidak dapat ditimpa oleh method di subclass-subclassnya	Berperilaku sebagai konstanta
Static	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Mendefinisikan method (milik) class. Dengan demikian tidak memerlukan instant object untuk menjalankanya. Method ini tidak dapat menjalankan method yang bukan static serta tidak dapat mengacu	Mendefinisikan variable milik class. Tidak memerlukan instant object untuk mengacunya. Variabel ini dapat digunakan bersama oleh semua instant objek.

Modifier	Class	Interface	Method	Variabel
			variable yang bukan static.	
synchronized	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Eksekusi dari method adalah secara mutual exclusive diantara semua thread. Hanya satu thread pada satu saat yang dapat menjalankan method	Tidak dapat diterapkan pada deklarasi. Diterapkan pada instruksi untuk menjaga hanya satu thread yang mengacu variable pada satu saat.
native	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Tidak ada badan method yang diperlukan karena implementasi dilakukan dengan bahas lain.	Tidak dapat diterapkan.
transient	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Variable tidak akan diserialisasi
Volatile	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Variabel diubah secara asinkron. Kompilator tidak pernah melakukan optimasi atasnya.

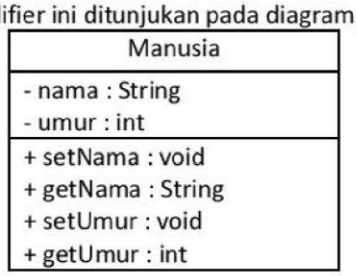
Deklarasi modifier dalam java :

```
[Modifier]  class/interface [nama class/interface] {}  /*deklarasi modifier di
class/interface*/

[Modifier][TypeData][nama Atribut];  //deklarasi modifier di atribut

[Modifier]{[TypeData]}[nama method](parameter_1,parameter_2,parameter_n) {}
//deklarasi modifier di method
```

Contoh :

```
public class Manusia {}  //contoh modifier di class
private int tinggi;      //contoh modifier di atribut
public int getTinggi(){} //contoh modifier di method
```

Contoh kelas yang mempunyai akses modifier ini ditunjukan pada diagram kelas Manusia berikut ini.



Keterangan : tanda - artinya memiliki access modifier **private**
\+ artinya memiliki access modifier **public**
\# artinya memiliki access modifier **protected**

Contoh implementasi di dalam program:

```
public class Manusia {

    //definisi atribut
    private String nama;
    private int umur;

    //definisi method
```

```
    public void setNama(String a){
        nama=a;
    }
    public String getNama(){
        return nama;
    }
    public void setUmur(int a){
        umur=a;
    }

    public int getUmur(){
        return umur;
    }
}
```

Kesepakatan umum penamaan dalam suatu Class :

1. **Nama Class** – gunakan **kata benda** dan **huruf pertama** dari tiap kata ditulis dengan **huruf besar** dan memiliki **access modifier public** : Manusia
2. Pada umumnya, **atribut** diberi **access modifier private**, dan **method** diberi **access modifier public.** Hal ini diterapkan untuk mendukung **konsep OO** yaitu **enkapsulasi** mengenai **data hiding**. Jadi, kita tidak langsung menembak data/atribut pada kelas tersebut, tetapi kita memberikan suatu antarmuka method yang akan mengakses data/atribut yang disembunyikan tersebut.
3. **Nama atribut** - gunakan **kata benda**, dan **diawali** dengan **huruf kecil** : nama, umur
4. **Nama Method** – gunakan **kata kerja**; **kecuali huruf** yang **pertama, huruf awal** tiap kata ditulis **kapital** : getNama(), getUmur()
5. Untuk **method** yang akan **memberikan** atau **mengubah nilai** dari **suatu atribut**, **nama method** kita tambahi dengan **kata kunci "set"**. : setUmur(int a), setNama(String a)
6. Untuk **method** yang akan **mengambil nilai** dari **atribut**, **nama method** kita tambahi dengan kata **kunci "get"**. : getUmur(), getNama()
7. **Konstanta** - **Semuanya ditulis** dengan **huruf besar**; **pemisah** antar **kata menggunakan garis bawah**: MAX_VALUE, DECIMAL_DIGIT_NUMBER

2.4 Field

Field adalah sebuah **atribut** .**Field** bisa berupa sebuah **variabel kelas**, **variabel objek**, **variabel method** objek atau **parameter** dari sebuah **fungsi. Field** merupakan **anggota** dari **kelas yang digunakan** untuk **menyimpan data.** Terdapat **dua field** dalam kelas , dibawah ini adalah contoh kelas yang terdapat 2 jenis field tersebut.

```
public class Circle {
  public static final double PI= 3.14159;
  public static double radiansToDegrees(double rads) {
    return rads * 180 / PI;
  }
  public double r;
  public double area() {
    return PI * r * r;
  }
  public double circumference() {
    return 2 * PI * r;
  }
}
```

- Class Field

Field yang **dikaitkan** dengan **class** di mana ia **didefinisikan, bukan** dengan sebuah **instance dari kelas**. Di bawah ini menyatakan pendeklarasian class field pada class `Circle`adalah :

```
  public static final double PI= 3.14159;
```

Static modifier diatas menyatakan bahwa **field** tersebut merupakan **class field**. Field ini berkaitan dengan kelas itu sendiri tidak dengan instance dari kelas. Berdasarkan kelas `Circle` maka jika terdapat method dari kelas `Circle` maka mengakses variabel `PI`dengan `Circle.PI` .

- Instance Field

Field apapun yang **dideklarasikan tanpa** menggunakan **static modifier** . Contoh instance field yang terdapat pada kelas `Circle` adalah:

```
public double r;
```

Field ini merupakan **field** yang **terkait dengan instance class** bukan dengan kelas itu sendiri. Berdasarkan kelas `Circle` , setiap objek `Circle` yang dibuat memiliki salinan sendiri field r. Contohnya r menyatakan jari-jari lingkaran. Jadi setiap objek `Circle` dapat memiliki jari-jari sendiri dari semua objek linkaran lainnya.

2.5 Method

Method dikenal juga sebagai suatu ***function*** dan ***procedure***. Dalam OOP, **method** digunakan untuk **memodularisasi program** melalui **pemisahan tugas** dalam suatu **class. Pemanggilan** method **menspesifikasikan nama** method dan **menyediakan informasi (parameter)** yang **diperlukan** untuk melaksanakan tugasnya.

Deklarasi method untuk yang mengembalikan nilai (fungsi)

```
[modifier] Type-data namaMethod(parameter1,parameter2,…parameterN)
{
    Deklarasi-deklarasi dan proses ;
    return nilai-kembalian
}
```

Type-data merupakan tipe data dari nilai yang dapat dikembalikan oleh method ini.

Deklarasi method untuk yang tidak mengembalikan nilai (prosedur)

```
[modifier] void namaMethod(parameter1,parameter2,…parameterN)
{
    Deklarasi-deklarasi dan proses ;
}
```

Contoh implementasi method :

```
public int jumlahAngka(int x, int y)
{
    int z=x+y;
      return z;
}
```

Ada dua cara melewatkan argumen ke method, yaitu:

1. **Melewatkan secara Nilai (Pass by Value)**
   Digunakan untuk **argumen** yang **mempunyai tipe data primitif** (byte, short, int, long, float, double, char, dan boolean). **Prosesnya** adalah **compiler** hanya **menyalin isi memori** (pengalokasian suatu variable), dan **kemudian menyampaikan salinan** tersebut **kepada method. Isi memory ini** merupakan **data "sesungguhnya"** yang akan **dioperasikan**. **Karena** hanya berupa **salinan isi memory**, maka **perubahan** yang terjadi pada variable **akibat proses** di dalam method **tidak** akan **berpengaruh** pada **nilai** variable **asalnya.**
2. **Melewatkan secara Referensi (Pass by Reference)**
   Digunakan pada **array** dan **objek**. **Prosesnya isi memory** pada **variable array** dan **objek** merupakan **penunjuk ke alamat memory** yang **mengandung data sesungguhnya** yang akan **dioperasikan**. Dengan kata lain, **variable array** atau **objek menyimpan alamat memory** bukan isi memory. Akibatnya, **setiap perubahan** variable di dalam method akan **mempengaruhi nilai** pada variable **asalnya.**

   Contoh program

```
class TestPass {
```

```
int i,j;
    TestPass(int a,int b) {
       i =a;
       j = b;
    }

//passed by value dengan parameter berupa tipe data primitif
void calculate(int m,int n) {
    m = m*10;
    n = n/2;
        }
//passed by reference dengan berupa tipe data class
void calculate(TestPass e) {
    e.i =  e.i*10;
    e.j =  e.j/2;
    }
}
```

```
class PassedByValue {
public static void main(String[] args) {
int x,y;
        TestPass z;
        z = new TestPass(50,100);
        x = 10;
        y = 20;

        System.out.println("Nilai sebelum passed by value : ");
        System.out.println("x = " + x);
        System.out.println("y = " + y);

        //passed by value
        z.calculate(x,y);
        System.out.println("Nilai sesudah passed by value : ");
        System.out.println("x = " + x);
        System.out.println("y = " + y);
        System.out.println("Nilai sebelum passed by reference : ");
        System.out.println("z.i = " + z.i);
        System.out.println("z.j = " + z.j);

    //passed by reference
        z.calculate(z);

        System.out.println("Nilai sesudah passed by reference : ");
        System.out.println("z.i = " + z.i);
        System.out.println("z.j = " + z.j);
   }
}
```

Output dari program diatas

```
Nilai sebelum passed by value :
x = 10
y = 20
Nilai sesudah passed by value :
x = 10
y = 20
Nilai sebelum passed by reference :
z.i = 50
z.j = 100
Nilai sesudah passed by reference :
z.i = 500
z.j = 50
```

Keterangan

Pada saat pemanggilan method *calculate()* dengan **metode pass by value**, hanya nilai dari variable *x* dan *y* saja yang dilewatkan ke variable *m* dan *n* , sehingga **perubahan** pada **variable *m*** dan ***n* tidak** akan **mengubah nilai** dari **variable *x*** dan ***y***. Sedangkan pada saat pemanggilan method *calculate()* dengan metode **pass by reference** yang menerima parameter bertipe class *Test*. Pada waktu kita memanggil method *calculate()* , nilai dari variable *z* yang berupa referensi ke obyek sesungguhnya dilewatkan ke variable *a* , sehingga variable *a* menunjukkan ke obyek yang sama dengan yang ditunjuk oleh variable *z* dan setiap **perubahan** pada objek tersebut dengan **menggunakan variable *a*** akan **terlihat efeknya** pada **variable *z*** yang terdapat pada kode yang memanngil method tersebut.

2.6 keyword "this"

Dalam Java terdapat suatu besaran **referensi khusus**, disebut **this,** yang **digunakan** dalam **method yang dirujuk** untuk objek yang sedang berlaku. Nilai this **merujuk** pada **objek** di mana **method yang** sedang **berjalan dipanggil.**

Contoh implementasi program:

```
class Lagu {
    private String pencipta;
    private String judul;

    public void IsiParam(String judul,String pencipta) {
        this.judul  = judul;
        this.pencipta = pencipta;
    }

    public void cetakKeLayar() {
        if(judul==null&& pencipta==null) return;
System.out.println("Judul : " + judul +", pencipta : " + pencipta);
    }
}

class DemoLagu {
publicstaticvoid main(String[] args) {
       Lagu a = new Lagu();
       a.IsiParam("God Will Make A Way","Don Moen ");
       a.cetakKeLayar();
    }
}
```

Keluaran dari program

```
Judul : God Will Make A Way, pencipta : Don Moen
```

keterangan

Perhatikan pada method *IsiParam()* di atas. Di sana kita mendeklarasikan nama variable yang menjadi parameternya sama dengan nama variable yang merupakan property dari class *Lagu* (*judul* dan *pencipta*). Dalam hal ini, kita perlu menggunakan keyword **this(**keyword ini merefer ke objek itu sendiri**)** agar dapat mengakses property *judul* dan *pencipta* di dalam method *IsiParam()* tersebut. Apa yang terjadi jika pada method *IsiParam()* keyword **this** dihilangkan ?

Keyword **this** juga dapat digunakan untuk **memanggil suatu konstruktor** dari konstruktor lainnya.

Contoh Program

```
public class Buku {
    private String pengarang;
    private String judul;

    private Buku() {
        this("Rumah Kita", "GoodBles"); // this ini digunakan untuk memanggil
konstruktor yang menerima dua parameter
    }
    private Buku(String judul,String pengarang)
    {
```

```
        this.judul = judul;
        this.pengarang = pengarang;
    }
    private void cetakKeLayar()
    {
        System.out.println("Judul : " + judul + " Pengarang : " + pengarang);
    }
    public static void main(String[] args)
    {
        Buku a,b ;
        a = new Buku("Jurassic Park", "Michael Chricton");
        b = new Buku();
        a.cetakKeLayar();
        b.cetakKeLayar();
    }
}
```

Keluaran dari program diatas

```
Judul : Jurassic Park Pengarang : Michael Chricton
Judul : Rumah Kita Pengarang : GoodBles
```

Pada contoh program di atas, kita juga dapat mengambil pelajaran bahwa penggunaan modifier private pada constructor mengakibatkan constructor tersebut hanya dapat dikenali dalam satu class saja, sehingga pembuatan objek hanya dapat dilakukan dalam lokal class tersebut.
Catatan penting lainnya : bahwa method **void** main(String[] args) haruslah bersifat **public static.**

2.7 Constructor

Constructor adalah **tipe khusus method** yang **digunakan** untuk **menginstansiasi** atau **menciptakan sebuah objek**. **Nama constructor** adalah **sama dengan nama kelasnya**. Selain itu, **constructor tidak bisa mengembalikan** suatu **nilai** (not return value) bahkan **void** sekalipun. Defaultnya, **bila** kita **tidak membuat constructor secara eksplisit,** maka **Java** akan **menambahkan constructor default** pada **program** yang kita buat **secara implisit**. **Constructor default** ini **tidak memiliki parameter masukan** sama sekali. Namun, **bila** kita telah **mendefinisikan minimal satu buah constructor,** maka **Java tidak** akan **menambah constructor default**.

Constructor juga dimanfaatkan untuk **membangun** suatu **objek** dengan **langsung mengeset atribut-atribut** yang **disandang** pada **objek** yang **dibuat** tersebut. Oleh karena itu, **constructor jenis ini** haruslah **memiliki parameter masukan** yang akan **digunakan** untuk **mengeset nilai** atribut.

Access modifier yang **dipaka**i pada **constructor** selayaknya adalah **public** karena **constructor** tersebut **akan diakses di luar classnya** (walaupun **kita** juga **bisa memberikan access modifier** pada constructor dengan **private**–artinya kita tidak bisa memanggil constructor tersebut di luar classnya). **Cara memanggil constructor** adalah dengan **menambahkan keyword new**. Keyword **new** dalam deklarasi ini artinya kita **mengalokasikan pada memory** sekian blok memory untuk **menampung objek** yang baru kita buat.

Deklarasi constructor :

```
[modifier] namaclass(parameter1){
    Body constructor;
}
[modifier] namaclass(parameter1,parameter2){
    Body constructor;
}
[modifier] namaclass(parameter1,parameter2,…,parameterN){
    Body constructor;
}
```

Contoh program sebelumnya dengan penambahan constructor eksplisit.

```
publicclass Manusia {
    private String nama;
    privateint umur;
```

```
//definisi constructor
    public Manusia(){ }   //constructor pertama = default tanpa parameter
    public Manusia(String a){     //constructor kedua
        nama=a;
    }
    public Manusia(String a, int b){     //constructor ketiga
        nama=a;
        umur=b;
    }

//definisi method
    public void setNama(String a){
        nama=a;
    }
    public String getNama(){
        return nama;
    }

    public void setUmur(int a){
        umur=a;
    }

    public int getUmur(){
        return umur;
    }
}

public class DemoManusia {
public static void main(String[] args) { //program utama
    Manusia arrMns[] = new Manusia[3]; //buat array of object
    Manusia objMns1 = new Manusia(); //constructor pertama

    objMns1.setNama("Markonah");
    objMns1.setUmur(76);

    Manusia objMns2 = new Manusia("Mat Conan"); //constructor kedua
    Manusia objMns3 = new Manusia("Bajuri", 13); //constructor ketiga

    arrMns[0] = objMns1;
    arrMns[1] = objMns2;
    arrMns[2] = objMns3;

for(int i=0; i<3; i++) {
        System.out.println("Nama : "+arrMns[i].getNama());
        System.out.println("Umur : "+arrMns[i].getUmur());
        System.out.println();
    }
}
}
```

Jika program di atas di-compile dan di running maka outputnya adalah sebagai berikut.

```
Nama : Markonah
Umur : 76

Nama : Mat Conan
Umur : 0

Nama : Bajuri
Umur : 13
```

Coba Anda analisis mengapa hasilnya seperti di atas !

Seperti source di atas, kita dapat mempunyai lebih dari satu *constructor*, masing-masing harus mempunyai parameter yang berbeda sebagai penandanya. Program secara otomatis akan memilih constructor mana yang

akan dijalankan sesuai dengan parameter masukan pada waktu pembuatan objek. Hal seperti ini disebut **overloading** terhadap *constructor*.

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

-Jangan pernah menggunakan kata-kata ini sebagai nama variabel, jika terjadi maka compile akan error -