PERTEMUAN 15

POLYMORPHISM(LANJUTAN)

Pengertian Polimorfisme

Polimorfisme terbagi menjadi dua suku kata yaitu, *Poly* yang berarti banyak dan *Morfisme* yang berarti bentuk. Dalam ilmu sains, Polimorfisme (*polymorphism*) adalah sebuah prinsip dalam biologi di mana organisme atau spesies memiliki banyak bentuk serta tahapan (*stages*). Prinsip tersebut diterapkan juga pada bahasa Java.

Polimorfisme dalam OOP merupakan sebuah konsep OOP di mana *class* memiliki banyak "bentuk" *method* yang berbeda, meskipun namanya sama. Maksud dari "bentuk" adalah isinya yang berbeda, namun tipe data dan parameternya berbeda.

Polimorfisme juga dapat diartikan sebagai teknik *programming* yang mengarahkan kamu untuk memprogram secara general daripada secara spesifik. Contohnya kita memiliki tiga class yang berbeda yaitu: "Kelinci",

“Kucing”, dan “Sapi”. Di mana ketiga *class* tersebut merupakan turunan dari *class* “Hewan”.

Sejalan dengan contoh yang diberikan, kamu diharapkan dapat mengerti dan memahami konsep polimorfisme itu sendiri.

Polimorfisme pada Java memiliki 2 macam yaitu diantaranya:

1. Static Polymorphism (Polimorfisme statis).
2. Dynamic Polymorphism (Polimorfisme dinamis).

Perbedaan keduanya terletak pada cara membuat polimorfisme. Polimorfisme statis menggunakan *method overloading,* sedangkan polimorfisme dinamis menggunakan *method overriding*.

Jika sebelumnya kamu belum tahu perbedaan antara *method overloading* dan *method overriding*, maka kita akan bahasa juga perbedaan dari keduanya.

Perbedaan Method Overloading dan Method Overriding

Baik teman-teman, *method overloading* terjadi pada sebuah *class* yang memiliki nama method yang sama tapi memiliki parameter dan tipe data yang berbeda.

Ingat! Intinya dalam sebuah class memiliki method yang sama, namun parameter dan tipe data yang berbeda.

Tujuan dari *method overloading* yaitu memudahkan penggunaan atau pemanggilan *method* dengan fungsionalitas yang mirip.

Aturan Method Overloading

- Nama method harus sama dengan method lainnya.
- Parameter haruslah berbeda.
- Return boleh sama, juga boleh berbeda.

Inilah contohnya.

Misal kamu membuat sebuah *class* dengan nama **Cetak.java**. Pada *class* ini mempunyai method maxNumber(). Perhatikan kode program dibawah ini.

Kode Lab:

```
1  public class Cetak {
2
3     // Method sama namun parameter berbeda
4     // Tipe data double
5     static double maxNumber(double a, double b) {
6        if (a < b) {
7           return a;
8        }else{
9           return b;
10       }
11    }
12
13    // Method sama, namun parameter berbeda
14    // Tipe data int
15    static int maxNumber(int a, int b) {
16       if (a < b){
17          return a;
18       }else {
```

```
19          return b;
20      }
21   }
22
23   public static void main(String[] argrs) {
24      System.out.println(maxNumber(5.5, 7.5));
25      System.out.println(maxNumber(10, 20));
26   }
27
28 }
```

Maka Outputnya:

5.5

10

Telusuri Kode

Coba perhatikan kode program di atas!

Pada *class* **Cetak.java** memiliki 2 *method* yang sama yaitu **maxNumber()**. Tapi parameter dan tipenya berbeda, yaitu:

- static double maxNumber(double a, double b)
- static int maxNumber(int a, int b)

Yang pertama memiliki parameter dan tipe data *double,* sedangkan satunya lagi memiliki parameter dan tipe data *int.* Hal ini jelas berbeda.

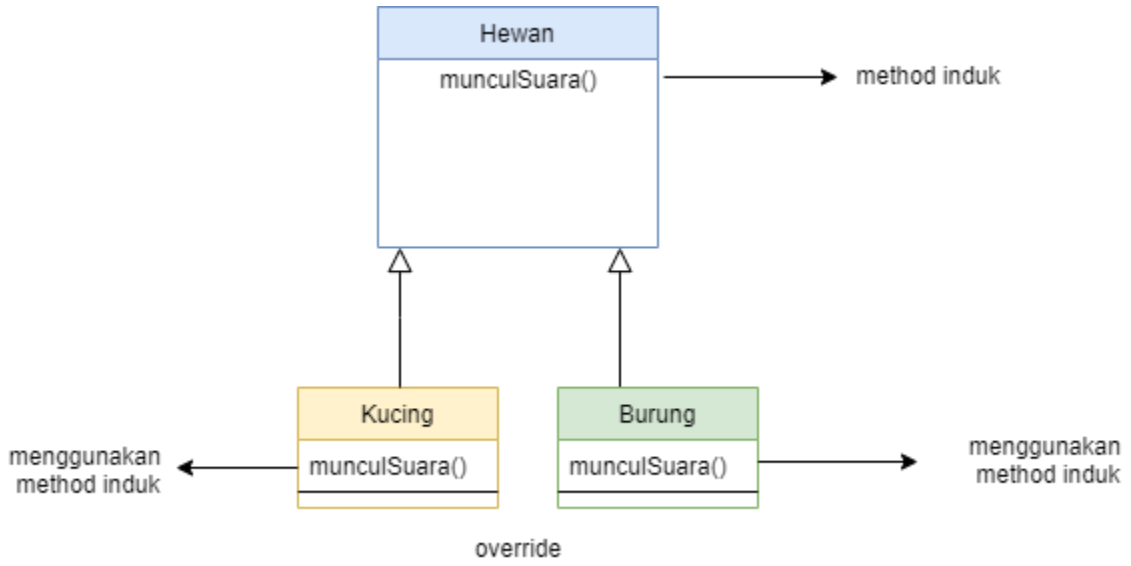
Lalu Polimorfisme yang dinamis seperti apa? Polimorfisme dinamis identik dengan menggunakan pewarisan (*inheritance*), implementasi *interface* bahkan abstrak *class*.

Untuk *inheritance* sendiri sudah dibahasa sebelumnya. Atau kamu bisa kunjungi linknya di awal tadi. Adapun pengertian dari *interface* dan abstrak.

Interface adalah *class* kosong yang berisi nama-nama *method* dan nantinya harus diimplementasikan pada *class* lain. Dalam pengimplementasiannya, tiap-tiap *class* akan mengimplementasikan secara berbeda dengan nama *method* yang sama.

Abstrak adalah *class* yang masih dalam bentuk bayangan. Ia tidak bisa dibuat langsung menjadi objek karena bentuknya masih bayangan atau abstrak. Tentunya abstrak ini induk dan jika ini konkrit, kamu mesti mengimplementasikan *method-method* tersebut. Ini bisa kamu lakukan dengan melakukan teknik pewarisan (*inheritance*).

Baik, sebagai contoh kita pilih salah satu *interface* atau abstrak. Saya coba dulu pilih menggunakan abstrak *class*. Perhatikan gambar tabel *class* diagram di bawah ini.



Kita akan coba membuat program sesuai dengan gambar tabel di atas.

Buatlah *class* **Hewan.java**. Berarti ini merupakan kelas induk dari semua kelas. Kemudian buat *class* **Kucing.java** dan **Burung.java** sebagai anaknya. Kamu bisa membuatnya dalam satu *class* maupun *class* yang berbeda.

```
1  abstract class Hewan {
2
3    // Mendeklrasikan class dan method tipe abstract
4    protected abstract void munculSuara();
5
6  }
7
8  class Kucing extends Hewan {
9
10   // Menggunakan method dari kelas induk abstrak
11   @Override
```

```
12    protected void munculSuara() {
13        System.out.println("Suara Kucing: Meow...meow..meow.");
14    }
15
16 }
17
18 class Burung extends Hewan {
19
20    // Menggunakan method dari kelas induk abstrak
21    @Override
22    protected void munculSuara() {
23        System.out.println("Suara Burung: Cit...cit..cit.");
24    }
25
26 }
27
28 public class Tampil {
29
30    public static void main (String[] args) {
31        Hewan kucing = new Kucing();
32        kucing.munculSuara();
33
34        Hewan burung = new Burung();
35        burung.munculSuara();
36    }
37
38 }
```

Maka Outputnya:

Suara Kucing: Meow…meow..meow.
Suara Burung: Cit…cit..cit.

Telusuri Kode

Class **Hewan.java** memiliki *method* utama yaitu *munculSuara()*. Kita menambahkan beberapa *subclass* anak kelas dari *Hewan.java* yakni **Kucing.java** dan **Burung.java**. Mereka memiliki *method* yang sama meskipun menampilkan *statements* argumen yang berbeda.

Kucing.java

```
1 // Menggunakan method dari kelas induk abstrak
2 @Override
```

```
3 protected void munculSuara() {
4     System.out.println("Suara Kucing: Meow...meow..meow.");
5 }
```

Burung.java

```
1 // Menggunakan method dari kelas induk abstrak
2 @Override
3 protected void munculSuara() {
4     System.out.println("Suara Burung: Cit..cit..cit");
5 }
```

Aturan Method Overriding

- Mode akses overriding method harus sama atau lebih luas daripada override method.
- Subclass hanya dapat dan boleh meng-override method superclass satu kali saja. Tidak boleh ada lebih dari satu method yang sama pada kelas.
- Soal aturan hak akses, setiap *subclass* tidak boleh mempunyai hak akses method overriding yang ketat dibandingkan dengan hak akses method pada superclass ataupun parent class.

Dapat disimpulkanpolimorfisme merupakan bagian dari teknik OOP, di mana sebuah *class* dapat memiliki method yang sama, namun bodynya berbeda-berbeda. Atau bisa kita artikan juga sebagai slogan berbeda-beda tapi namanya tetap sama.