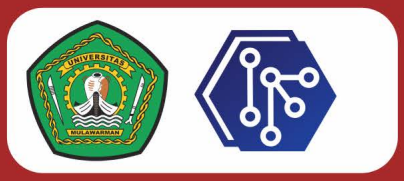


Modul Praktikum
PBO

5

MODUL 5 POLYMORPHISM

PENGERTIAN POLYMORPHISM

Polimorfisme (dalam inggris: *polymorphism*) terbagi menjadi dua suku kata yaitu, *Poly* yang berarti banyak dan *Morfisme* yang berarti bentuk. Dalam ilmu sains, Polimorfisme 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 sama.

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".

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*.

KELEBIHAN POLYMORPHISM

- Dapat memberikan *code reusability* (penggunaan kode secara berulang), sehingga dan
- Memudahkan programmer dalam melakukan perubahan terhadap kodingan tanpa mengubah keseluruhan program (melalui bentuk-bentuk class baru yang dibuat).
- Tidak banyak menggunakan memori dalam ukuran file maupun untuk compiling.

METHOD OVERLOADING (STATIC)

Method overloading terjadi pada sebuah *class* yang memiliki nama method yang sama tapi memiliki parameter dan tipe data yang berbeda. 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 yang di-overloading harus sama dengan method yang di-overwrite.
- Setiap method harus memiliki parameter yang berbeda.
- Setiap urutan atau tipe data parameter harus berbeda pada tiap method.
- Tipe data (*return type*) harus sama.

Perhatikan contoh pertama berikut:

```
class Printer {
  public void print(String nama) {
    System.out.println("Pesanan atas nama: " + nama);
  }

  public void print(int nomor) {
    System.out.println("Pesanan dengan nomor: " + nomor);
  }

  public void print(String nama, int nomor) {
    System.out.println(
        "Pesanan atas nama " + nama + " dengan nomor " + nomor
    );
  }

  static double maxNumber(double a, double b) {
    return a > b ? a : b;
  };

  static int maxNumber(int a, int b){
    return a > b ? a : b;
  };
}
```

```java
public class Main {
  public static void main(String[] args) {
    Printer printer = new Printer();
    printer.print("Gading");
    // Output:
    // Pesanan atas nama: Gading

    printer.print(31);
    // Output:
    // Pesanan dengan nomor: 31

    printer.print("Gading", 31);
    // Output:
    // Pesanan atas nama Gading dengan nomor 31

    System.out.println(Printer.maxNumber(5.5, 7.5));
    // Output:
    // 7.5

    System.out.println(Printer.maxNumber(10, 50));
    // Output:
    // 50
  }
}
```

Pada *class* **Printer** memiliki 2 *method* yang sama, yaitu **print()** dan **maxNumber()**. Akan tetapi, parameter dan tipenya berbeda, yaitu:

- public void print(String nama)
- public void print(int nomor)
- public void print(String nama, int nomor)

  dan

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

Yang pertama memiliki parameter dan tipe data *double*, sedangkan satunya lagi memiliki parameter dan tipe data *int.* Inilah yang disebut **Polimorfisme Statis**, yaitu setiap method memiliki bentuk yang berbeda-beda (yaitu tipe argumen dari parameter *method*-nya) setelah file java dibangun (*compiled*).

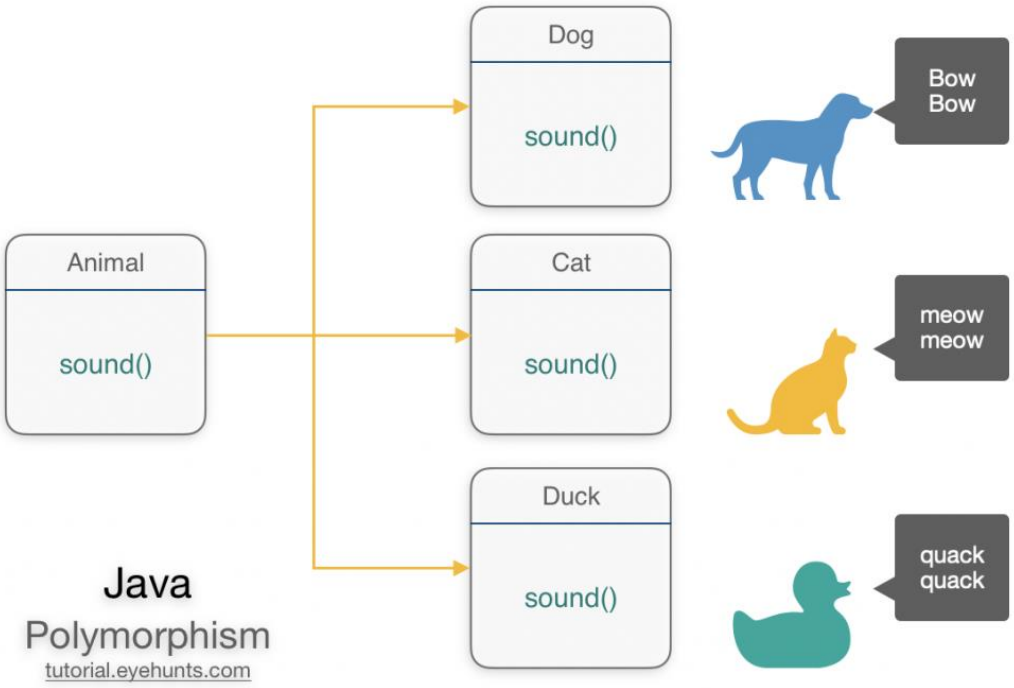
METHOD OVERRIDING (DYNAMIC)

Polimorfisme dinamis identik dengan menggunakan pewarisan (***inheritance***), implementasi ***interface***, bahkan ***abstract*** *class* yang akan diajarkan pada modul berikutnya.

Aturan Method Overriding

- Nama method yang di-overriding harus sama dengan method yang di-overwrite.
- Setiap method harus memiliki parameter yang sama.
- Tipe data method (*return type*) harus sama.
- Bukan merupakan method final.
- Tidak memiliki *throws Exception* pada metode yang
- Setiap *subclass* tidak boleh mempunyai *access modifier* (hak akses) method overriding yang lebih ketat dibandingkan dengan hak akses method pada superclass/induk-nya.

Perhatikan gambar tabel *class* diagram di bawah ini.



Pada *class* **Hewan** memiliki 1 *method*, yaitu **aksi()**. Method tersebut diwarisi oleh kelas-kelas lain, yaitu kelas Kucing, Kelinci, dan Kerbau. Akan tetapi, isinya berbeda-beda.

Apabila suatu method yang diwariskan tersebut memiliki sebuah parameter, maka ketika melakukan Override, harus disertakan dengan sama persis.

Inilah yang disebut **Polimorfisme Dinamis**, yaitu konsep tiap method akan dipanggil dan dibuat pada saat file Java berjalan (runtime), tidak saat kompilasi seperti jenis polimorfisme statis.

Contohnya, jika terdapat kelas hewan dan kelas anjing yang merupakan subclass dari kelas hewan, dan kelas kucing meng-override metode suara() dari kelas hewan, maka ketika sebuah objek kucing dipasangkan dengan referensi hewan, pemanggilan metode suara() akan memanggil implementasi dari metode tersebut di kelas kucing, bukan di kelas hewan.

```java
class Hewan {
  public void aksi() {
    System.out.println("Hewan ini beraksi");
  }
}

class Kucing extends Hewan {
  @Override
  public void aksi(){
    System.out.println("Kucing memburu tikus");
  }
}

class Kelinci extends Hewan {
  @Override
  public void aksi(){
    System.out.println("Kelinci memakan wortel");
  }
}

class Kerbau extends Hewan {
  @Override
  public void aksi(){
    System.out.println("Kerbau membajak sawah");
  }
}

public class Main {
  public static void main(String[] args) {
    Hewan hewan = new Hewan();
    Hewan kucing = new Kucing();
    Hewan kelinci = new Kelinci();
    Hewan kerbau = new Kerbau();

    hewan.aksi();   // Hewan ini beraksi
    kucing.aksi();  // Kucing memburu tikus
    kelinci.aksi(); // Kelinci memakan wortel
    kerbau.aksi();  // Kerbau membajak sawah
  }
}
```

Polimorfisme dinamis dapat memberikan fleksibilitas dan modularitas dalam kode, sehingga kita dapat menulis kode yang lebih *general* (umum) dan *reusable* (dapat digunakan kembali).

Hal tersebut disebabkan karena kelas-kelas yang berbeda dapat dipasangkan dengan referensi objek yang sama, sehingga memudahkan kita untuk memanipulasi objek dalam sebuah hirarki kelas yang lebih besar secara seragam dan efisien.

Referensi:

https://www.dicoding.com/blog/pengertian-polimorfisme-dalam-pemrograman-java/

https://www.geeksforgeeks.org/overriding-in-java/

https://tutorial.eyehunts.com/java/java-polymorphism-definition-type-example/

https://www.w3schools.com/java/java_polymorphism.asp