PERTEMUAN KE – *10*
Polimorfisme

A. TUJUAN

- Dapat membuat aplikasi yang mengimplementasi compile-time (early binding) polimorphisme
- Dapat membuat aplikasi yang mengimplementasi runtime (late binding) polimorphisme

B. TEORI SINGKAT

Polimorfisme adalah kemampuan sebuah variabel reference untuk merubah behavior sesuai dengan apa yang dipunyai object.

Polimorfisme membuat objek-objek yang berasal dari subclass yang berbeda, diperlakukan sebagai objek-objek dari satu superclass. Hal ini terjadi ketika memilih method yang sesuai untuk diimplementasikan ke objek tertentu berdasarkan pada subclass yang memiliki method bersangkutan.

Polimorfisme dapat diterjemahkan pula sebagai “sebuah method yang sama namanya (homonim) tetapi mempunyai tingkah laku yang berbeda”. Komputer membedakan method berdasarkan signature method (saat compile) atau berdasarkan reference object (saat runtime).

Ada dua bentuk polimorfisme

1. Overloading (compile-time (early binding) polimorphisme)
   Overloading adalah salah satu cara penerapan dalam konsep polimorfisme. Overload merupakan pendefinisian ulang suatu metode dalam class yang sama. Syarat overload yaitu metode dan tipe parameter harus berbeda dalam kelass yang sama.

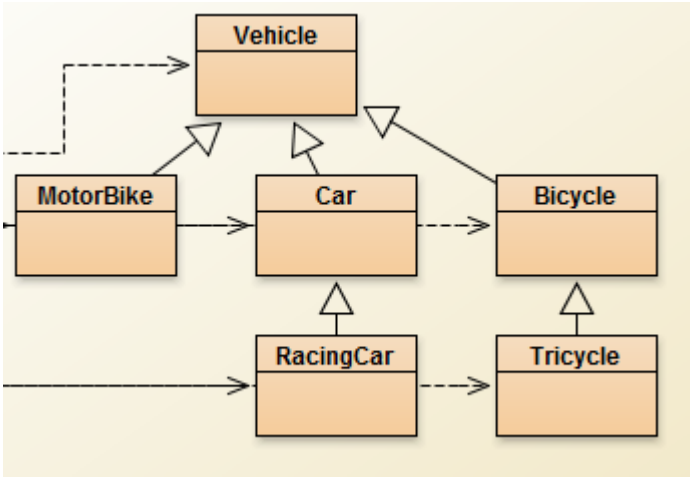
2. Overriding (runtime (late binding) polimorphisme)
   Override merupakan pendefinisian ulang suatu metode oleh subclass. Syarat Override yaitu metode, return type, dan parameter harus sama. Jika tidak sama maka bukan dianggap sebagai override tetapi metode yang baru pada subclass.

   Dikatakan **late-binding (run time) polymorphism** karena saat compile komputer tidak tahu method mana yang harus dipanggil, dan baru akan diketahui saat run-time. Run-time

polymorphism dapat diterapkan melalui **overridden method.** Run time polymorphism juga disebut dengan istilah **dynamic binding.**

Anggaplah subclass meng-override method tertentu pada superclass. Andai kita cipta objek dari subclass dan memasukkannya dalam superclass. Walau objek subclass telah dimasukkan pada superclass, sewaktu objek tersebut memanggil method yang dioverride, ia tetap memanggil method yang versi subclassnya, bukan versi superclass.

Contoh



- Class Vehicle memiliki method move().
- Class MotorBike meng-override method move().

  Java akan menunggu saat runtime untuk menentukan method move() yang mana akan dipanggil.

C. PRAKTIK

1. Praktik compile-time (early binding) polimorphisme dengan overloading method

```
class Jumlah{

   public int tambah(int x, int y){
   return x + y;
   }

   public int tambah(int x, int y, int z){
   return x + y + z;
   }

   public int tambah(double pi, int x){
   return (int)pi + x;
   }
```

```
}

public class Penjumlahan{

public static void main(String [] args)
{
Jumlah obj = new Jumlah();
System.out.println(obj.tambah(2,5)); // int, int
System.out.println(obj.tambah(2, 5, 9)); // int, int, int
System.out.println(obj.tambah(3.14159, 10)); // double, int
}
}
```

Pada praktik di atas polymorphism ditunjukkan dengan menggunakan berbagai method ***tambah***. Komputer membedakan mana method yang dipanggil berdasarkan signature dari method yaitu parameter inputnya (jumlah parameter input, type data parametr input, dan urutan type data).

Bentuk polymorphism ini dinamakan **early-binding (**atau **compile-time) polymorphism** karena komputer mengetahui mana method tambah yang dipanggil setelah mengcompile byte code . Jadi setelah proses compile ketika code sudah menjadi byte code, komputer akan "tahu" method mana yang akan dieksekusi Bentuk ini yang kita kenal dengan **overloaded method.**

2. Praktik compile-time (early binding) polimorphisme dengan overloading Konstruktor

```
class Penggajian{
 double gapok;
 double masa_kerja;
 Penggajian(double g, double mk)
 {
       gapok = g;
       masa_kerja=mk;
 }
 Penggajian()
 {
 gapok =0;
 masa_kerja=0;
 }
 Penggajian(double lembur)
 {
       gapok = masa_kerja = lembur;
 }
 double hitung_gaji()
 {
```

```
      return gapok*masa_kerja;
 }
}

class OverloadingKonstruktor
{
 public static void main(String args[])
 {
      Penggajian Karyawan1 = new Penggajian(10,15);
      Penggajian Karyawan2 = new Penggajian();
      Penggajian Karyawan3 = new Penggajian(5);
      double gaji;
      gaji = Karyawan1.hitung_gaji();
      System.out.println("Gaji Karyawan 1= " +gaji);
      gaji = Karyawan2.hitung_gaji();
      System.out.println("Gaji Karyawan 2= " +gaji);
      gaji = Karyawan3.hitung_gaji();
      System.out.println("Gaji Karyawan 3= " +gaji);
      }
 }
```

3. Runtime polimorfisme dengan overriding method

```
class Kendaraan {
public void info() {
    System.out.println("Info pada kendaraan : ");
    }
}

class Roda2 extends Kendaraan {
public void info() {
    System.out.println ("Info pada Roda2");
    }
}

class Motor extends Roda2
{
public void info() {
    System.out.println("Info pada Motor");
    }
}

public class Test {
public static void main(String[] args){
    Roda2 roda2ku;
    Motor motorku;
    Kendaraan k = new Kendaraan();
    roda2ku=new Roda2();
    motorku=new Motor();
    k.info();
    k=roda2ku;
```

```
    k.info();
    k=motorku;
    k.info();
}
}
```

4. Runtime polimorfisme dengan data member

```
class Induk {
int x = 5;
public void Info() {
System.out.println("Ini class Induk");
}
}
class Anak extends Induk {
int x = 10;
public void Info() {
System.out.println("Ini class Anak");
}
}
public class Test2 {
public static void main(String args[])
{
Induk tes=new Anak();
System.out.println("Nilai x = " + tes.x);
tes.Info();
}
}
```

Ketika program di atas dijalankan akan terlihat bahwa ketika diakses atribut x dari objek tes, maka yang muncul adalah nilai x dari super kelas, bukan atribut x dari subkelas. Akan tetapi ketika diakses method Info() dari objek tes, yang muncul adalah method Info() dari sub kelas, bukan method Info() dai superkelas.

5. Runtime polimorfisme dengan passing parameter

```
class AlatGerak{
      void bergerak(){
            System.out.println(" Saya mampu bergerak");
      }
}

class Sayap extends AlatGerak{
      void bergerak(){
            System.out.println(" Saya bisa terbang");
      }
}
```

```
class Kaki extends AlatGerak{
      void bergerak(){
            System.out.println(" Saya bisa jalan-jalan");
      }
}

class Burung {
      private AlatGerak alatGerak=new AlatGerak();
      Burung(){
            System.out.println("Hai saya Burung");
      }

public void bergerak() {
      alatGerak.bergerak();
}

public void setAlatGerak(AlatGerak alatGerak) {
      this.alatGerak = alatGerak;
      System.out.println("Sekarang saya pakai " + alatGerak);
      }
}

class BurungTest{
public static void main(String[] args){
Burung merpati=new Burung();
merpati.bergerak();
Sayap sayap=new Sayap();
Kaki kaki=new Kaki();
merpati.setAlatGerak(sayap);
merpati.bergerak();
merpati.setAlatGerak(kaki);
merpati.bergerak();
}
}
```

Method setAlatGerak dapat menerima berbagai type yaitu Sayap dan Kaki. Hal ini legal karena pada kenyataannya Sayap dan Kaki adalah AlatGerak. Saat method bergerak() milik object trutle dipanggil, method yang dipanggil adalah sesuai dengan method yang dimiliki oleh type yang diberikan.

D. LATIHAN

- *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
- *Dikerjakan di laboratorium pada jam praktikum.*

E. TUGAS

- *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
- *Dikerjakan di rumah dan dilampirkan pada laporan.*