

# **LAPORAN PRAKTIKUM**

## **PEMROGRAMAN BERORIENTASI OBJEK**

### **PERTEMUAN KE-11**



**Disusun Oleh :**

**NAMA : Raden Isnawan Argi Aryasatya**

**NIM : 195410257**

**JURUSAN : Teknik Informatika**

**JENJANG : S1**

**KELAS : TI-5**

**Laboratorium Terpadu**  
**Sekolah Tinggi Manajemen Informatika Komputer**  
**AKAKOM**  
**YOGYAKARTA**

**2021**

## PERTEMUAN KE-11 (POLIMORFISME)

### TUJUAN

Dapat membuat aplikasi yang mengimplementasi compile-time (early binding) polimorfisme, dapat membuat aplikasi yang mengimplementasi runtime (late binding) polimorfisme

### DASAR TEORI

Polimorfisme adalah kemampuan sebuah variabel reference untuk merubah behavior sesuai dengan apa yang dimiliki 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).

---

### PRAKTIK 1: compile-time (early binding) polimorfisme dengan overloading method

```
Penjumlahan.java x
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
    }
}
```

output:

```
C:\Windows\system32\cmd.exe
7
16
13
Press any key to continue . . . _
```

penjelasan:

Polimorfisme adalah kemampuan suatu objek untuk mengungkapkan banyak hal melalui satu cara yang sama. Polimorfisme mengizinkan kelas induk untuk mendefinisikan sebuah method general (bersifat umum) untuk semua kelas turunannya, dan selanjutnya kelas-kelas turunan dapat memperbaharui implementasi dari method tersebut secara lebih spesifik sesuai dengan karakteristiknya masing-masing. Dalam program di atas, kita menggunakan konsep polimorfisme dengan overloading method. Overloading adalah suatu keadaan dimana beberapa method memiliki nama yang sama tetapi fungsionalitasnya berbeda.

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 parameter 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.

Untuk pembuatan programnya, kita awali dengan membuat class bernama Jumlah. Di dalamnya kita buat method bermodifier public bernama 'tambah' yang memiliki parameter int x dan y. Hal itu kita tulis dengan **public int tambah(int x, int y)** yang kemudian memproses return yaitu **return x + y** yang nantinya hasilnya adalah penjumlahan antara nilai x dan nilai y.

Setelah itu kita mengamplifikasikan konsep polimorfisme yaitu dengan membuat method overloading bernama 'tambah' juga tetapi dengan parameter, fungsionalitas, dan return value yang berbeda. Baris kode nya adalah:

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

kemudian kita buat lagi method overloading dengan baris kode:

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

selanjutnya kita buat program main method bernama public class Penjumlahan. Kita awali dengan menuliskan **public static void main (String [] args)** sebagai method utama dalam menjalankan program. Public, berarti metode ini dapat dipanggil dari luar class. Static, menunjukkan metode ini bersifat sama untuk semua class. Void, berarti metode ini tidak mengembalikan nilai. Argument args[] adalah array objek string argument baris-baris perintah.

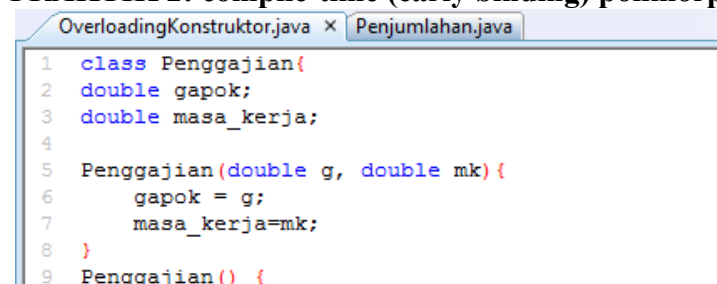
Lalu kita melakukan instansiasi yaitu proses pembuatan objek. Kita membuat objek milik class Jumlah bernama obj dengan menuliskan baris kode **Jumlah obj = new Jumlah();**. Terakhir, di dalam objek tersebut kita beri value atau nilai terhadap setiap method yang telah dibuat supaya menghasilkan perhitungan yang dikehendaki. Hal itu kita lakukan dengan menuliskan baris-baris kode berikut:

```
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
```

Sehingga hasilnya adalah:

```
7 (x+y)  
16 (x+y+z)  
13 ((int)pi + x)
```

## **PRAKTIK 2: compile-time (early binding) polimorfisme dengan overloading Konstruktor**



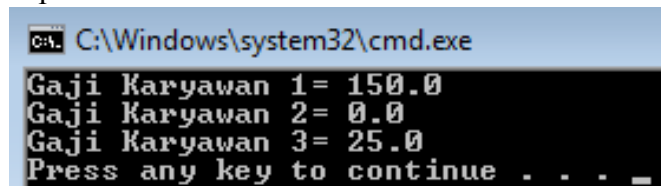
```
OverloadingKonstruktor.java x Penjumlahan.java  
1 class Penggajian{  
2     double gapok;  
3     double masa_kerja;  
4  
5     Penggajian(double g, double mk){  
6         gapok = g;  
7         masa_kerja=mk;  
8     }  
9     Penggajian() {
```

```

11  masa_kerja=0;
12  }
13
14  Penggajian(double lembur) {
15      gapok = masa_kerja = lembur;
16  }
17  double hitung_gaji() {
18      return gapok*masa_kerja;
19  }
20  }
21  }
22  public class OverloadingKonstruktor {
23  public static void main(String args[]) {
24      Penggajian Karyawan1 = new Penggajian(10,15);
25      Penggajian Karyawan2 = new Penggajian();
26      Penggajian Karyawan3 = new Penggajian(5);
27      double gaji;
28      gaji = Karyawan1.hitung_gaji();
29      System.out.println("Gaji Karyawan 1= " +gaji);
30      gaji = Karyawan2.hitung_gaji();
31      System.out.println("Gaji Karyawan 2= " +gaji);
32      gaji = Karyawan3.hitung_gaji();
33      System.out.println("Gaji Karyawan 3= " +gaji);
34  }
35  }

```

output:



```

C:\Windows\system32\cmd.exe
Gaji Karyawan 1= 150.0
Gaji Karyawan 2= 0.0
Gaji Karyawan 3= 25.0
Press any key to continue . . . _

```

penjelasan:

pada dasarnya program ini konsepnya sama dengan program sebelumnya yaitu membahas tentang overloading. Bedanya, disini yang overloading adalah konstruktor method. Konstruktor adalah method khusus yang akan dieksekusi pertama kali saat pembuatan (instansiasi) suatu Object (Class)). Konstruktor merupakan sebuah method dimana nama konstruktor harus sama persis dengan nama file dan Class-nya. Konstruktor merupakan komponen wajib yang harus ada dalam suatu class, jika tidak dituliskan dalam class maka java akan secara otomatis membuat satu constructor pada class tersebut. Ciri-ciri konstruktor yaitu, tidak memiliki return value. Konstruktor harus punya nama yang sama dengan nama Class-nya. Bisa memiliki parameter ataupun tidak. Konstruktor berfungsi untuk memberikan nilai awal (instans) pada sebuah class ketika class tersebut dibuat.

Untuk pembuatannya, kita awali dengan membuat class bernama Penggajian. Di dalamnya ada variabel gapok dan masa\_kerja yang memiliki tipe data double. double adalah tipe data 64 bit IEEE 754 floating point. Tipe data ini umumnya digunakan untuk tipe data desimal. Kemudian kita buat konstruktor bernama sama dengan nama class yaitu Penggajian yang memiliki parameter berupa variabel double g dan variabel double mk, kita tulis dengan **Penggajian(double g, double mk)**. Lalu di dalamnya kita akses variabel gapok dan masa kerja kemudian kita beri value kepada kedua variabel tersebut. Kita lakukan itu dengan menuliskan **gapok = g** dan **masa\_kerja=mk**.

Selanjutnya kita mempraktekkan polimorfisme dengan membuat konstruktor overloading. Kali ini, kita membuat konstruktor overloading dengan tidak menggunakan parameter yaitu Penggajian(). Di dalamnya kita memberikan nilai/argumen kepada kedua variabel yaitu dengan menuliskan kode **gapok = 0;** dan **masa\_kerja=0;**.

Konstruktor overloading selanjutnya adalah **Penggajian(double lembur)** yang di dalamnya kita

menuliskan **gapok = masa\_kerja = lembur;**

setelah itu kita membuat method baru bernama **hitung\_gaji()** yang merupakan method untuk menghitung gaji. Kita tuliskan dengan:

```
double hitung_gaji() {  
    return gapok*masa_kerja;
```

lalu kita buat main method bernama **OverloadingKonstruktor**. Eksekusi kita awali dengan menuliskan **public static void main(String args[])** sebagai method utama dalam menjalankan program. Public, berarti metode ini dapat dipanggil dari luar class. Static, menunjukkan metode ini bersifat sama untuk semua class. Void, berarti metode ini tidak mengembalikan nilai. Argument **args[]** adalah array objek string argument baris-baris perintah. Di dalamnya kita melakukan instansiasi 3 objek untuk 3 orang karyawan yaitu dengan menuliskan baris kode:

**Penggajian Karyawan1 = new Peggajian(10,15);** -> akan menghasilkan output  $10 \times 15 = 150.0$

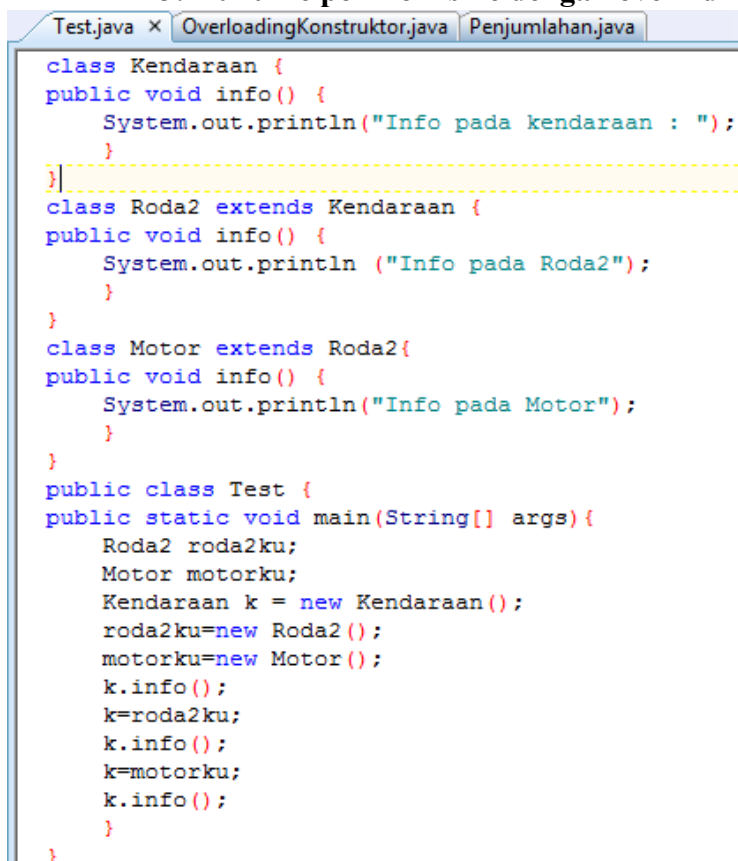
**Penggajian Karyawan2 = new Peggajian();** -> akan menghasilkan output  $0 \times 0 = 0.0$

**Penggajian Karyawan3 = new Peggajian(5);** -> akan menghasilkan output  $5 \times 5 = 25.0$

terakhir kita deklarasikan variabel gaji bertipe data double. Variabel gaji memiliki argumen berupa objek yang diinisialisasi dengan method **hitung\_gaji** supaya menghasilkan dan mencetak output berupa hasil perhitungan gaji masing-masing karyawan. Kita tulis baris-baris kode nya dengan:

```
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);
```

### PRAKTIK 3: Runtime polimorfisme dengan overriding method



```
Test.java x OverloadingKonstruktor.java Penjumlahan.java  
  
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();  
    }  
}
```

output:

```
C:\Windows\system32\cmd.exe
Info pada kendaraan :
Info pada Roda2
Info pada Motor
Press any key to continue . . . _
```

penjelasan:

Overriding method dilakukan dengan cara mendeklarasikan method yang sama persis dengan method yang ada pada superclassnya. Deklarasi dari method method harus sama persis, mulai dari access modifier, keyword yang digunakan, nama method, jumlah parameter dan letak parameter ataupun hingga deklarasi throws exception.

pada program di atas, superclass bernama Kendaraan yang memiliki sebuah method bernama info() yang memunculkan kalimat "Info pada kendaraan : ". Kemudian, kita membuat subclass bernama Roda2 yang mendapatkan warisan method (menggunakan extends) dari class Kendaraan. Di dalamnya kita mempraktekkan polimorfisme dengan mendeklarasikan kembali method info() tetapi dengan fungsionalitas yang berbeda yaitu memunculkan kalimat "Info pada Roda2". Selanjutnya kita buat subclass lagi bernama Motor yang extends dari class Roda2 yang di dalamnya ada method info juga yang override dari method info milik Roda2 (sebenarnya sama saja dengan method info milik Kendaraan). Method info di class Motor memunculkan kalimat "Info pada Motor". Terakhir, kita buat main method bernama Test. Di dalamnya kita buat 3 objek baru bernama k milik class Kendaraan, roda2ku milik class Roda2, dan motorku milik class Motor. Terakhir, kalimat - kalimat pada setiap method milik class-class tersebut kita munculkan menggunakan setiap objek.

#### PRAKTIK 4: Runtime polimorfisme dengan data member

```
Test2.java x Test.java OverloadingKonstruktor.java Penjumlahan.java
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();
    }
}
```

output:

```
C:\Windows\system32\cmd.exe
Nilai x = 5
Ini class Anak
Press any key to continue . . . _
```

penjelasan:

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() dari superclass.

Inti dari program ini adalah memanfaatkan konsep polimorfisme untuk menggabungkan dua data member dari dua class berbeda. Pada class Induk, kita memiliki variabel x = 5 dan method Info yang memunculkan kalimat "Ini class Induk". Pada subclass Anak, variabel x memiliki nilai 10 dan method Info memunculkan kalimat "ini class Anak". Lalu pada instansiasi objek, kita menggabungkan output kedua class dengan cara membuat objek tes dengan menuliskan **Induk tes=new Anak();** dari objek tersebut, kita panggil dan munculkan x dan method info dengan menuliskan **System.out.println("Nilai x = " + tes.x);** lalu **tes.Info();**.

### PRAKTIK 5: Runtime polimorfisme dengan passing parameter

```
BurungTest.java × Test2.java Test.java OverloadingKonstruktor.java Penjumlahan.java
1 class AlatGerak{
2 void bergerak(){
3 System.out.println(" Saya mampu bergerak");
4 }
5 }
6 class Sayap extends AlatGerak{
7 void bergerak(){
8 System.out.println(" Saya bisa terbang");
9 }
10 }
11 class Kaki extends AlatGerak{
12 void bergerak(){
13 System.out.println(" Saya bisa jalan-jalan");
14 }
15 }
16 class Burung {
17 private AlatGerak alatGerak=new AlatGerak();
18 Burung(){
19 System.out.println("Hai saya Burung");
20 }
21 public void bergerak() {
22 alatGerak.bergerak();
23 }
24 public void setAlatGerak(AlatGerak alatGerak) {
25 this.alatGerak = alatGerak;
26 System.out.println("Sekarang saya pakai " + alatGerak);
27 }
28 }
29 class BurungTest{
30 public static void main(String[] args){
31 Burung merpati=new Burung();
32 merpati.bergerak();
33 Sayap sayap=new Sayap();
34 Kaki kaki=new Kaki();
35 merpati.setAlatGerak(sayap);
36 merpati.bergerak();
37 merpati.setAlatGerak(kaki);
38 merpati.bergerak();
39 }
40 }
```

output:

```
ca. C:\Windows\system32\cmd.exe
Hai saya Burung
Saya mampu bergerak
Sekarang saya pakai Sayap@72ea2f77
Saya bisa terbang
Sekarang saya pakai Kaki@33c7353a
Saya bisa jalan-jalan
Press any key to continue . . .
```

penjelasan:

saya jelaskan terlebih dahulu apa itu passing parameter. Passing parameter merupakan suatu mekanisme mengirim dan atau mengembalikan suatu nilai kepada fungsi atau prosedur. Ada 2 jenis passing parameter, yang pertama adalah Pass by value (parameter nilai), jika di dalam fungsi atau prosedur dilakukan perubahan nilai parameter yang dilewatkan secara nilai, maka nilai parameter yang sebenarnya tidak ikut berubah hal ini dikarenakan parameter yang dilewatkan secara nilai akan di copy sebagai nilai lokal di prosedur atau fungsi yang bersangkutan. Yang kedua adalah Pass by reference (parameter variabel), jika di dalam prosedur atau fungsi dilakukan perubahan nilai parameter yang dilewatkan secara reference, maka nilai parameter yang sebenarnya juga akan berubah.

Pada program di atas, kita membuat class bernama class bernama AlatGerak yang memiliki method bergerak yang memunculkan kalimat "Saya mampu bergerak". Kemudian kita mempraktekkan polimorfisme dengan membuat dua class bernama Sayap dan Kaki yang extends dari kelas AlatGerak. Lalu ada class Burung juga yang di dalamnya ada instansiasi objek bernama alatGerak. Di dalam class Burung ada method bergerak dan satu method setter/mutator baru bernama setAlatGerak yang memiliki parameter AlatGerak dan alatGerak. Di dalamnya kita akses alatGerak dengan menuliskan **this.alatGerak = alatGerak;**. Terakhir kita buat main method, di dalamnya kita buat 3 objek dari 3 class tadi yaitu objek merpati milik class Burung, sayap milik class Sayap, dan kaki milik class Kaki. Lalu kita munculkan outputnya.

Jadi intinya adalah 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.

---

---

## KESIMPULAN

pada laporan/pertemuan ini, kita belajar tentang polimorfisme. Hal yang paling ditekankan pada laporan ini adalah bentuk-bentuk dari polimorfisme. Ada 2 bentuk polomorfisme yang kita praktekkan di laporan ini, yaitu:

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 kelas 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.

**Terima Kasih**