# JOBSHEET 9

# OVERLOADING DAN OVERRIDING

## 1. Kompetensi

Setelah menempuh pokok bahasan ini, mahasiswa mampu :

a. Memahami konsep overloading dan overriding,
b. Memahami perbedaan overloading dan overriding,
c. Ketepatan dalam mengidentifikasi method overriding dan overloading
d. Ketepatan dalam mempraktekkan instruksi pada jobsheet
e. Mengimplementasikan method overloading dan overriding.

## 2. Pendahuluan

### 2.1 Overloading

adalah menuliskan kembali method dengan nama yang sama pada suatu class. Tujuannya dapat memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang mirip. Untuk aturan pendeklarasian method Overloading sebagai berikut:

- Nama method harus sama.
- Daftar parameter harus berbeda.
- Return type boleh sama, juga boleh berbeda.

Ada beberapa daftar parameter pada overloading dapat dilihat sebagai berikut:

- Perbedaan daftar parameter bukan hanya terjadi pada perbedaan banyaknya parameter, tetapi juga urutan darai parameter tersebut.
- Misalnya saja dua buah parameter berikut ini:
  - Function_member (int x, string n)
  - Function_member (String n, int x)
- Dua parameter tersebut juga di anggap berbeda daftar parameternya.
- Daftar parameter tidak terkait dengan penamaan variabel yang ada dalam parameter.
- Misalnya saja 2 daftar parameter berikut :
  - function_member(int x)
  - function_member(int y)

- Dua daftar parameter diatas dianggap sama karena yang berbeda hanya penamaan variable parameternya saja.

Overloading juga bisa terjadi antara parent class dengan subclass-nya jika memenuhi ketiga syarat overload. Ada beberapa aturan overloading yaitu:

- Primitive widening conversion didahulukan dalam overloading dibandingkan boxing dan var args.
- Kita tidak dapat melakukan proses widening dari tipe wrapper ke tipe wrapper lainnya (mengubah Integer ke Long).
- Kita tidak dapat melakukan proses widening dilanjutkan boxing (dari int menjadi Long)
- Kita dapat melakukan boxing dilanjutkan dengan widening (int dapat menjadi Object melalui Integer)
- Kita dapat menggabungkan var args dengan salah satu yaitu widening atau boxing

## 2.2 Overriding

adalah Subclass yang berusaha memodifkasi tingkah laku yang diwarisi dari superclass. Tujuannya subclass dapat memiliki tingkah laku yang lebih spesifik sehingga dapat dilakukan dengan cara mendeklarasikan kembali method milik parent class di subclass.
Deklarasi method pada subclass harus sama dengan yang terdapat di super class. Kesamaan pada:

- Nama
- Return type (untuk return type : class A atau merupakan subclass dari class A)
- Daftar parameter (jumlah, tipe dan urutan)

Sehingga method pada parent class disebut overridden method dan method pada subclass disebut overriding method. Ada beberapa aturan method didalam overriding:

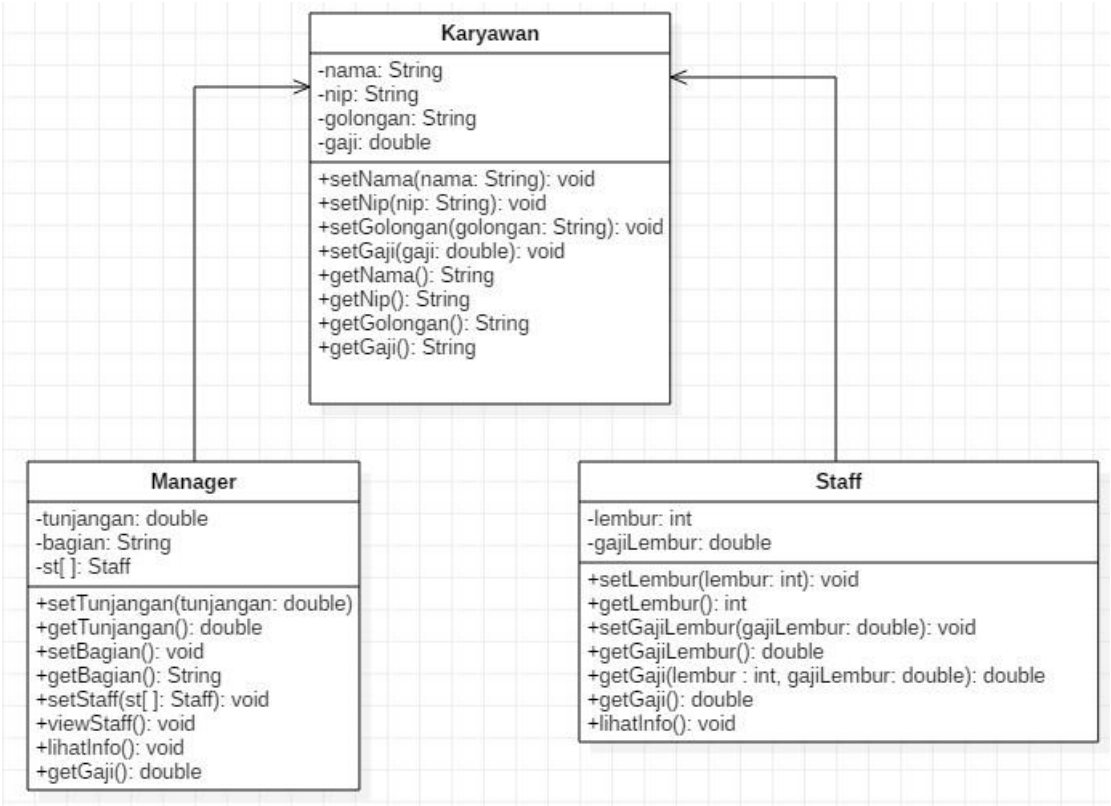
- Mode akses overriding method harus sama atau lebih luas dari pada overridden method.
- Subclass hanya boleh meng-override method superclass satu kali saja, tidak boleh ada lebih dari satu method pada kelas yang sama persis.

- Overriding method tidak boleh throw checked exceptions yang tidak dideklarasikan oleh overridden method.

## 3. Praktikum

### 3.1 Percobaan 1

Untuk kasus contoh berikut ini, terdapat tiga kelas, yaitu Karyawan, Manager, dan Staff. Class Karyawan merupakan superclass dari Manager dan Staff dimana subclass Manager dan Staff memiliki method untuk menghitung gaji yang berbeda.



### 3.2 Karyawan

```
public class Karyawan {

    /**
     * @param args the command line arguments
     */
  //  public static void main(String[] args) {
        // TODO code application logic here
 private String nama;
 private String nip;
 private String golongan;
 private double gaji;

 public void setNama(String nama)
 {
  this.nama=nama;
 }
 public void setNip(String nip)
 {
  this.nip=nip;
 }
 public void setGolongan(String golongan)
 {
  this.golongan=golongan;
   switch(golongan.charAt(0)){
    case '1':this.gaji=5000000;
      break;
    case '2':this.gaji=3000000;
      break;
    case '3':this.gaji=2000000;
      break;
    case '4':this.gaji=1000000;
      break;
    case '5':this.gaji=750000;
      break;
   }
 }
 public void setGaji(double gaji)
 {
  this.gaji=gaji;
 }
 public String getNama()
 {
  return nama;
 }
 public String getNip()
 {
  return nip;
 }
 public String getGolongan()
 {
  return golongan;
 }
 public double getGaji()
 {
  return gaji;
 }
}
```

### 3.3 Staff

```
 public class Staff extends Karyawan {
private int lembur;
private double gajiLembur;

public void setLembur(int lembur)
{
 this.lembur=lembur;
}
public int getLembur()
{
 return lembur;
}
public void setGajiLembur(double gajiLembur)
{
 this.gajiLembur=gajiLembur;
}
public double getGajiLembur()
{
 return gajiLembur;
}
public double getGaji(int lembur,double gajiLembur)
{
 return super.getGaji()+lembur*gajiLembur;
}
public double getGaji()
{
 return super.getGaji()+lembur*gajiLembur;
}
public void lihatInfo()
{
 System.out.println("NIP  :"+this.getNip());
 System.out.println("Nama  :"+this.getNama());
 System.out.println("Golongan :"+this.getGolongan());
 System.out.println("Jml Lembur :"+this.getLembur());
 System.out.printf("Gaji Lembur :%.0f\n", this.getGajiLembur());
 System.out.printf("Gaji  :%.0f\n",this.getGaji());
}
}
```

Overloading

Overriding

### 3.4 Manager

```
public class Manager extends Karyawan {
private double tunjangan;
private String bagian;
private Staff st[];

public void setTunjangan(double tunjangan)
{
 this.tunjangan=tunjangan;
}
public double getTunjangan()
{
 return tunjangan;
}
public void setBagian(String bagian)
{
 this.bagian=bagian;
}
public String getBagian()
{
 return bagian;
}
public void setStaff(Staff st[])
{
 this.st=st;
}

public void viewStaff()
{
 int i;
 System.out.println("--------------------");
 for(i=0;i<st.length;i++)
 {
  st[i].lihatInfo();
 }
 System.out.println("--------------------");
}
public void lihatInfo()
{
 System.out.println("Manager  :"+this.getBagian());
 System.out.println("NIP  :"+this.getNip());
 System.out.println("Nama  :"+this.getNama());
 System.out.println("Golongan :"+this.getGolongan());
 System.out.printf("Tunjangan :%.0f\n",this.getTunjangan());
 System.out.printf("Gaji  :%.0f\n",this.getGaji());
 System.out.println("Bagian  :"+this.getBagian());
 this.viewStaff();
}
public double getGaji()
{
 return super.getGaji()+tunjangan;
}
}
```

### 3.5 Utama

```
public class Utama {
public static void main(String[] args)
{
 System.out.println("Program Testing Class Manager & Staff");
 Manager man[]=new Manager[2];
 Staff staff1[]=new Staff[2];
 Staff staff2[]=new Staff[3];

 //pembuatan manager

 man[0]=new Manager();
 man[0].setNama("Tedjo");
 man[0].setNip("101");
 man[0].setGolongan("1");
 man[0].setTunjangan(5000000);
 man[0].setBagian("Administrasi");

 man[1]=new Manager();
 man[1].setNama("Atika");
 man[1].setNip("102");
 man[1].setGolongan("1");
 man[1].setTunjangan(2500000);
 man[1].setBagian("Pemasaran");
```

```
staff1[0]=new Staff();
staff1[0].setNama("Usman");
staff1[0].setNip("0003");
staff1[0].setGolongan("2");
staff1[0].setLembur(10);
staff1[0].setGajiLembur(10000);

staff1[1]=new Staff();
staff1[1].setNama("Anugrah");
staff1[1].setNip("0005");
staff1[1].setGolongan("2");
staff1[1].setLembur(10);
staff1[1].setGajiLembur(55000);
man[0].setStaff(staff1);

staff2[0]=new Staff();
staff2[0].setNama("Hendra");
staff2[0].setNip("0004");
staff2[0].setGolongan("3");
staff2[0].setLembur(15);
staff2[0].setGajiLembur(5500);
```

```
staff2[1]=new Staff();
staff2[1].setNama("Arie");
staff2[1].setNip("0006");
staff2[1].setGolongan("4");
staff2[1].setLembur(5);
staff2[1].setGajiLembur(100000);

staff2[2]=new Staff();
staff2[2].setNama("Mentari");
staff2[2].setNip("0007");
staff2[2].setGolongan("3");
staff2[2].setLembur(6);
staff2[2].setGajiLembur(20000);
man[1].setStaff(staff2);

//cetak informasi dari manager + staffnya
man[0].lihatInfo();
man[1].lihatInfo();
}
```

## 4. Latihan

```
public class PerkalianKu {

 void perkalian(int a, int b){

  System.out.println(a * b);

 }

 void perkalian(int a, int b, int c){

  System.out.println(a * b * c);

 }

 public static void main(String args []){

  PerkalianKu objek = new PerkalianKu();

  objek.perkalian(25, 43);
  objek.perkalian(34, 23, 56);
 }
}
```

4.1 Dari source coding diatas terletak dimanakah overloading?

**Pata method perkalian**

4.2 Jika terdapat overloading ada berapa jumlah parameter yang berbeda

**Ada 2 parameter yang berbeda**

```
public class PerkalianKu {

 void perkalian(int a, int b){

  System.out.println(a * b);

 }

 void perkalian(double a, double b){

  System.out.println(a * b);

 }

 public static void main(String args []){

  PerkalianKu objek = new PerkalianKu();

  objek.perkalian(25, 43);
  objek.perkalian(34.56, 23.7);
 }
}
```

4.3 Dari source coding diatas terletak dimanakah overloading?

**Terletak pada method perkalian**

4.4 Jika terdapat overloading ada berapa tipe parameter yang berbeda?

**Ada 2 parameter yang berbeda**

```
  class Ikan{
   public void swim(){
      System.out.println("Ikan bisa berenang");
   }
}
class Piranha extends Ikan{
   public void swim(){
      System.out.println("Piranha bisa makan daging");
   }
}
public class Fish {
    public static void main(String[] args) {
        Ikan a = new Ikan();
       Ikan b = new Piranha();
       a.swim();
       b.swim();
    }
   }
```

4.5 Dari source coding diatas terletak dimanakah overriding?

**Terdapat pada method swim**

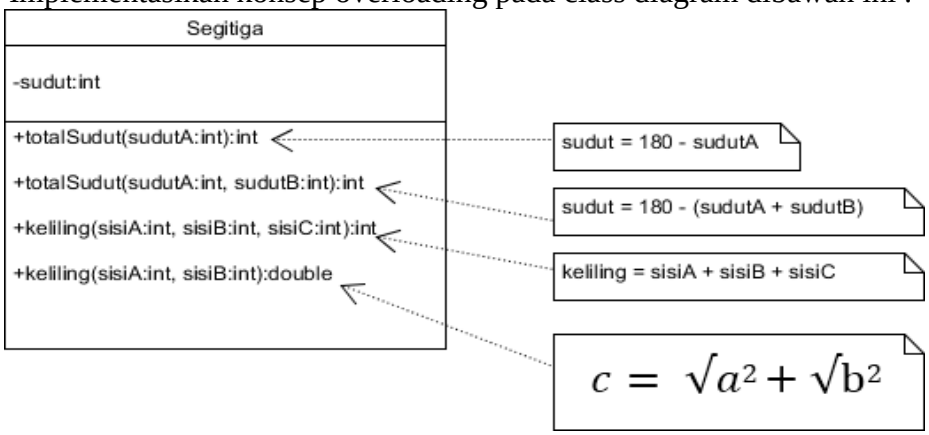
4.6 Jabarkanlah apabila sourcoding diatas jika terdapat overriding?

**Overriding terjadi ketika metode swim() di kelas Piranha menimpa metode yang sama di kelas Ikan, sehingga Piranha memiliki perilaku berbeda. Ketika swim() dipanggil dari objek Piranha, hasilnya adalah "Piranha bisa makan daging", sedangkan dari objek Ikan, hasilnya adalah "Ikan bisa berenang".**

## 5. Tugas

### 5.1 Overloading

Implementasikan konsep overloading pada class diagram dibawah ini :




**Code:**

**Segitiga:**

```
public class Segitiga {
    private int sudut;
    public int totalSudut(int sudutA) {
        return 180 - sudutA;
    }
    public int totalSudut(int sudutA, int sudutB) {
        return 180 - (sudutA + sudutB);
    }
    public int keliling(int sisiA, int sisiB, int sisiC) {
        return sisiA + sisiB + sisiC;
    }
    public double keliling(int sisiA, int sisiB) {
        double sisiC = Math.sqrt((sisiA * sisiA) + (sisiB * sisiB));
        return sisiA + sisiB + sisiC;
    }
    public double hitungLuas(int alas, int tinggi) {
        return 0.5 * alas * tinggi;
    }
    public double hitungLuas(int sisiA, int sisiB, int sisiC) {
        double s = (sisiA + sisiB + sisiC) / 2.0;
        return Math.sqrt(s * (s - sisiA) * (s - sisiB) * (s - sisiC));
    }
}
```

**Main:**

```
public class Main {
    public static void main(String[] args) {
        Segitiga segitiga = new Segitiga();
        System.out.println("Sudut yang belum diketahui dari sudut 60: " +
        segitiga.totalSudut(60));
        System.out.println("Sudut yang belum diketahui dari sudut 60 dan 60: " + segitiga.totalSudut(60, 60));
        System.out.println("\nKeliling segitiga dengan sisi 3, 4, 5: " +
        segitiga.keliling(3, 4, 5));
        System.out.println("Keliling segitiga siku-siku dengan sisi 3, 4: " +
        segitiga.keliling(3, 4));
        System.out.println("\nLuas segitiga dengan alas 3 dan tinggi 4: " +
        segitiga.hitungLuas(3, 4));
        System.out.println("Luas segitiga dengan sisi 3, 4, 5: " +
        segitiga.hitungLuas(3, 4, 5));
    }
}
```

**Hasil:**

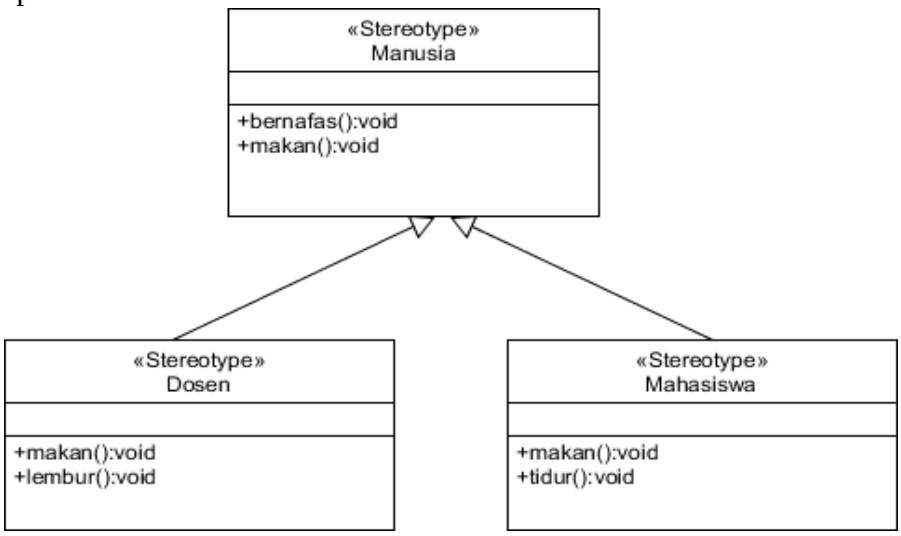
```
Sudut yang belum diketahui dari sudut 60: 120
Sudut yang belum diketahui dari sudut 60 dan 60: 60

Keliling segitiga dengan sisi 3, 4, 5: 12
Keliling segitiga siku-siku dengan sisi 3, 4: 12.0

Luas segitiga dengan alas 3 dan tinggi 4: 6.0
Luas segitiga dengan sisi 3, 4, 5: 6.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 5.2 Overriding

Implementasikan class diagram dibawah ini dengan menggunakan teknik dynamic method dispatch :



**Manusia:**

```
public class Manusia {
    public void bernafas() {
        System.out.println("Manusia bernafas...");
    }
    public void makan() {
        System.out.println("Manusia makan...");
    }
}
```

**Dosen:**

```
public class Dosen  extends Manusia {
    @Override
    public void makan() {
        System.out.println("Dosen makan dengan tertib...");
    }
    public void lembur() {
        System.out.println("Dosen sedang lembur...");
    }
}
```

**Mahasiswa:**

```
public class Mahasiswa  extends Manusia {
    @Override
    public void makan() {
        System.out.println("Mahasiswa makan di kantin...");
    }
    public void tidur() {
        System.out.println("Mahasiswa tidur di kelas...");
    }
}
```

**MainManusia:**

```java
public class MainManusia {
    public static void main(String[] args) {
    Manusia manusia = new Manusia();
    Manusia dosen = new Dosen();
    Manusia mahasiswa = new Mahasiswa();

    System.out.println("=== Program Dynamic Method Dispatch ===");
    System.out.println("\nPerilaku objek Manusia:");
    manusia.bernafas();
    manusia.makan();
    System.out.println("\nPerilaku objek Dosen:");
    dosen.bernafas();
    dosen.makan();

    System.out.println("\nPerilaku objek Mahasiswa:");
    mahasiswa.bernafas();
    mahasiswa.makan();

    System.out.println("\nMemanggil method khusus child class:");
    ((Dosen)dosen).lembur();
    ((Mahasiswa)mahasiswa).tidur();
    }
}
```

**Hasil:**

```
=== Program Dynamic Method Dispatch ===

Perilaku objek Manusia:
Manusia bernafas...
Manusia makan...

Perilaku objek Dosen:
Manusia bernafas...
Dosen makan dengan tertib...

Perilaku objek Mahasiswa:
Manusia bernafas...
Mahasiswa makan di kantin...

Memanggil method khusus child class:
Dosen sedang lembur...
Mahasiswa tidur di kelas...
BUILD SUCCESSFUL (total time: 0 seconds)
```