

OVERLOADING & OVERRIDING



POLYMORPHISM

- Polymorphism: poly (banyak), morph (bentuk)
- Konsep polimorfisme pada OOP membolehkan sebuah aksi diimplementasikan secara berbeda

POLYMORPHISM

1. Overloading

- JVM menentukan method mana yang akan dipanggil pada **compile-time** → compile-time polymorphism
- Disebut juga static binding atau early binding

2. Overriding

- JVM menentukan method mana yang akan dipanggil pada saat **run-time** → run-time polymorphism
- Disebut juga dynamic binding atau late binding

OVERLOADING

- Method overloading berarti kondisi dimana ada method dengan nama yang sama, tetapi memiliki method signature yang berbeda.
- **Method signature**: jumlah, tipe data dan susunan parameter
- Method overloading dapat terjadi pada kelas yang sama atau kelas lain yang terkait dalam hierarki pewarisan.
- Karakteristik overloading:
 1. Nama method sama.
 2. Method signature berbeda.
 3. Return type boleh sama atau berbeda.

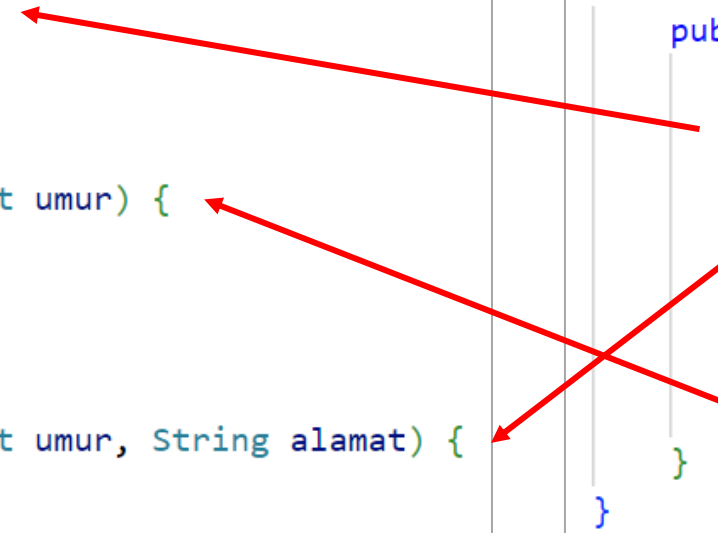
OVERLOADING EXAMPLE

```
public class Mahasiswa {  
    public String nama;  
    public int umur;  
    public String alamat;  
  
    public void setBiodata(String nama) {  
        this.nama = nama;  
    }  
  
    public void setBiodata(String nama, int umur) {  
        this.nama = nama;  
        this.umur = umur;  
    }  
  
    public void setBiodata(String nama, int umur, String alamat) {  
        this.nama = nama;  
        this.umur = umur;  
        this.alamat = alamat;  
    }  
}
```

Di kelas Mahasiswa, terdapat 3 method dengan nama setBiodata() tetapi dengan signature yang berbeda.

```
public class Mahasiswa {  
    public String nama;  
    public int umur;  
    public String alamat;  
  
    public void setBiodata(String nama) {  
        this.nama = nama;  
    }  
  
    public void setBiodata(String nama, int umur) {  
        this.nama = nama;  
        this.umur = umur;  
    }  
  
    public void setBiodata(String nama, int umur, String alamat) {  
        this.nama = nama;  
        this.umur = umur;  
        this.alamat = alamat;  
    }  
}
```

```
public class Test {  
    Run | Debug  
    public static void main(String[] args) {  
        Mahasiswa mahasiswa1 = new Mahasiswa();  
        mahasiswa1.setBiodata("Ani");  
  
        Mahasiswa mahasiswa2 = new Mahasiswa();  
        mahasiswa2.setBiodata("Budi", 18, "Batu");  
  
        Mahasiswa mahasiswa3 = new Mahasiswa();  
        mahasiswa3.setBiodata("Cica", 19);  
    }  
}
```



METHOD SIGNATURE

- Salah satu karakteristik overloading adalah method signature yang berbeda
- **Method signature**: jumlah, tipe data dan susunan parameter
- Method signature disebut berbeda jika:
 - ☐ Jumlahnya berbeda, *atau*
 - ☐ Jumlahnya sama tapi tipe data berbeda , *atau*
 - ☐ Jumlah dan tipe data sama tapi susunannya berbeda
- Perhatikan bahwa **nama parameter tidak diperhitungkan**

METHOD SIGNATURE (2)

```
public class Mahasiswa {  
    public String nama;  
    public int umur;  
    public String alamat;  
  
    public void setBiodata(String nama, int umur) {  
        this.nama = nama;  
    }  
  
    public void setBiodata(String alamat, int umur) {  
        this.alamat = alamat;  
    }  
}
```

Jumlah, tipe data,
susunan sama

Compile error:

Duplicate method setBiodata(String, int)

METHOD SIGNATURE (3)

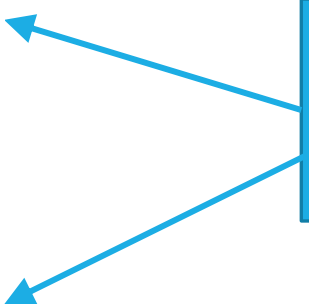
```
public class Mahasiswa {  
    public String nama;  
    public int umur;  
    public String alamat;  
  
    public void setBiodata(String nama, int umur) {  
        this.nama = nama;  
        this.umur = umur;  
    }  
  
    public void setBiodata(String alamat, int umur) {  
        this.alamat = alamat;  
        this.umur = umur;  
    }  
}
```

```
public class Test {  
    Run | Debug  
    public static void main(String[] args) {  
        Mahasiswa mahasiswa1 = new Mahasiswa();  
        mahasiswa1.setBiodata("abcdefg", 19);  
    }  
}
```

METHOD SIGNATURE (4)

```
public class Mahasiswa {  
    public String nama;  
    public int umur;  
    public String alamat;  
  
    public void setBiodata(String nama, int umur) {  
        this.nama = nama;  
        this.umur = umur;  
    }  
  
    public void setBiodata(int umur, String nama) {  
        this.nama = nama;  
        this.umur = umur;  
    }  
}
```

Jumlah dan tipe data sama, tetapi susunan berbeda



OVERLOADING OF A CONSTRUCTOR

```
public class Donat {  
    public String topping;  
  
    public Donat() {  
  
    }  
  
    public Donat(String topping) {  
        this.topping = topping;  
    }  
}
```

OVERLOADING OF A CONSTRUCTOR

```
public class Donat {  
    public String topping;  
  
    public Donat() {  
  
    }  
  
    public Donat(String topping) {  
        this.topping = topping;  
    }  
}
```

```
public class DemoDonat {  
    Run | Debug  
    public static void main(String[] args) {  
        Donat donat1 = new Donat("Choco");  
  
        Donat donat2 = new Donat();  
        donat2.topping = "Strawberry";  
    }  
}
```

WHEN DO WE USE OVERLOADING?

- Terkadang kita perlu membuat beberapa method yang memiliki fungsi/tujuan yang mirip, tetapi dengan parameter yang berbeda atau implementasi yang berbeda
- Menyederhanakan kode program dan memunculkan konsistensi dalam penamaan method

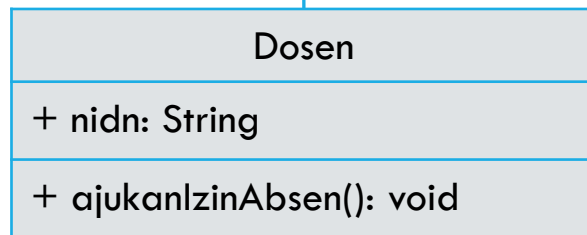
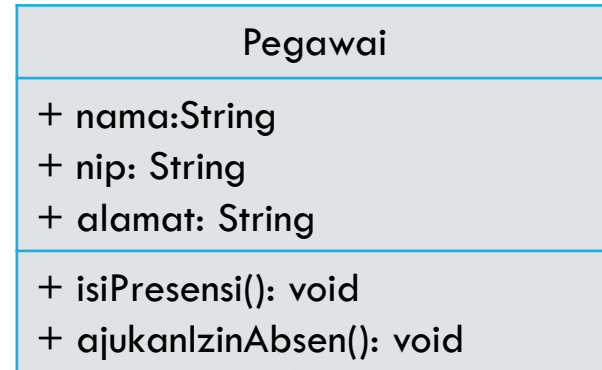
OVERRIDING

Overriding terjadi ketika subclass memiliki method dengan nama dan signature yang sama dengan superclass nya.

Karakteristik:

1. Dilakukan oleh method pada subclass terhadap method pada superclass
2. Nama method sama
3. Method signature sama

Pada class Dosen terdapat method `ajukanIzinAbsen()` dengan signature yang sama dengan class Pegawai → *Overriding*

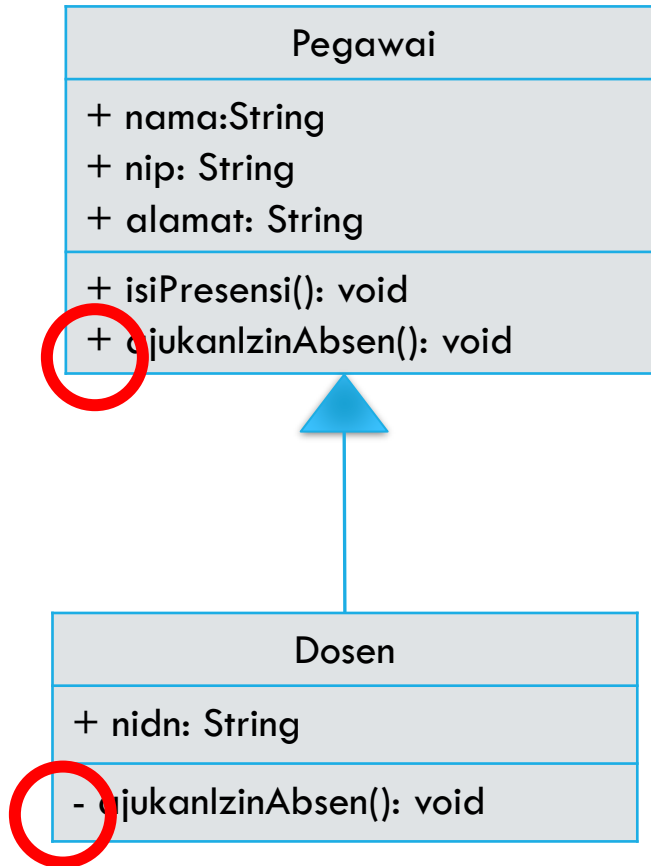


```
public void ajukanIzinAbsen() {  
    System.out.println("Menghubungi Biro Kepegawaian");  
}
```

```
public void ajukanIzinAbsen() {  
    System.out.println("Menghubungi Biro Kepegawaian");  
    System.out.println("Menginfokan kepada ketua kelas");  
}
```

ACCESS LEVEL MODIFIER

Modifier	Class	Package	Subclass	Outside Package
public	v	v	v	v
protected	v	v	v	
no modifier	v	v		
private	v			



Method yang di-override (method di superclass) tidak boleh memiliki access level modifier yang lebih luas daripada method yang meng-override (method di subclass).

Jika aturan tersebut dilanggar, akan muncul compile error: *Cannot reduce the visibility of the inherited method from Pegawai*

WHEN TO USE OVERRIDING

Overriding digunakan jika terdapat suatu kelas yang merupakan turunan dari kelas lain, tetapi implementasi suatu method berbeda

FINAL KEYWORD

Kata kunci **final** merupakan modifier yang digunakan untuk membuat sebuah class, atribut, atau method tidak dapat diubah

- ✓ Final class → tidak dapat diturunkan/di-extend
- ✓ Final atribut → nilainya tidak dapat dimodifikasi
- ✓ Final method → tidak dapat di-override

FINAL CLASS

```
public final class Pegawai {  
    public String nip;  
    public String nama;  
    public double gaji;  
}
```

```
public class Dosen extends Pegawai {  
    public String nidn;  
}
```

Final class → tidak dapat diturunkan/di-extend

The type Dosen cannot subclass the final
class Pegawai

FINAL ATRIBUT

```
public class Pegawai {  
    public final String nip;  
    public String nama;  
    public double gaji;  
  
    public Pegawai(String nip, String nama, double gaji) {  
        this.nip = "jsdfhsd";  
        this.nama = nama;  
        this.gaji = gaji;  
    }  
  
    public void setNIP(String nip) {  
        this.nip = nip;  
    }  
}
```

Final atribut → nilainya tidak dapat dimodifikasi

The final field
Pegawai.nip cannot be
assigned

FINAL ATRIBUT (2)

```
public class Lingkaran {  
    public final double phi = 3.14;  
    public double jariJari;  
  
    public Lingkaran(double jariJari) {  
        this.jariJari = jariJari;  
    }  
}
```

Final atribut → nilainya tidak dapat dimodifikasi

FINAL METHOD

Final method → tidak dapat di-override

Cannot override the final method from Pegawai

```
public class Pegawai {  
    public String nip;  
    public String nama;  
    public double gaji;  
  
    public final void ajukanIzin() {  
        System.out.println("Izin ke biro kepegawaian");  
    }  
}
```

```
public class Dosen extends Pegawai {  
    public String nidn;  
  
    public void ajukanIzin() {  
        System.out.println("Izin ke biro kepegawaian");  
        System.out.println("Menginfokan ketua kelas");  
    }  
}
```

TUGAS KELOMPOK (PROGRESS 1)

1. Diskusikan secara kelompok PBL tentang class diagram (class, object, attribute, method) yang akan digunakan pada sistem Anda (perhatikan jika ada overloading atau overriding)
2. Analisis singkat gambaran project Anda.