

OVERLOADING & OVERRIDING



POLYMORPHISM

- Polymorphism: poly (many), morph (shape)
- The concept of polymorphism in OOP allows an action to be implemented differently

POLYMORPHISM

1. Overloading

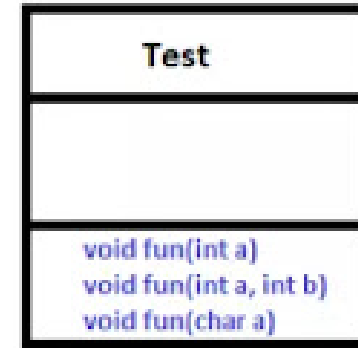
The JVM specifies which method to call on compile-time polymorphism

Also called static binding or early binding

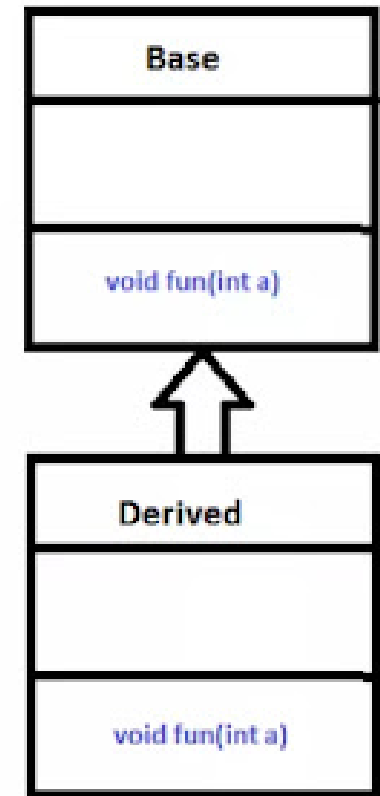
2. Overriding

The JVM specifies which method to call at run-time polymorphism

Also called dynamic binding or late binding



Overloading



Overriding

OVERLOADING

- Method overloading means a condition where there are methods with the same name, but have different method signatures.
- **Method signature**: number, data type and arrangement of parameters
- Method overloading can occur in the same class or other related classes in the inheritance hierarchy.
- Overloading characteristics:
 - The name of the method is the same.
 - The signature method is different.
 - The return type can be the same or different.

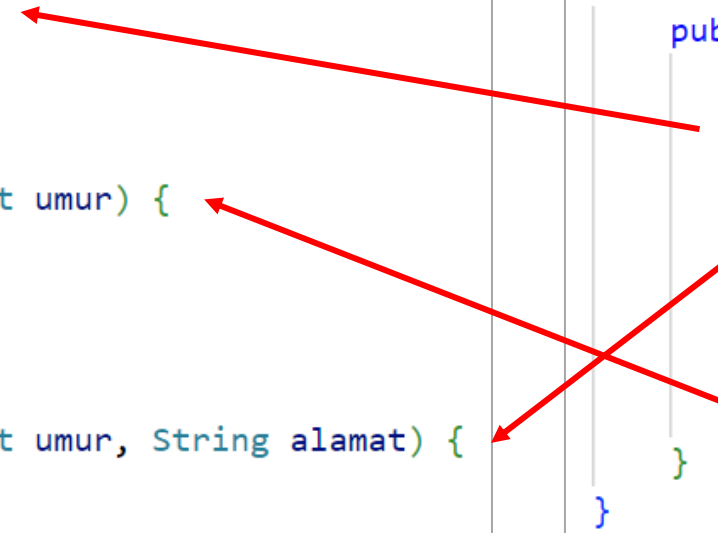
OVERLOADING EXAMPLE

```
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;  
    }  
}
```

In the Student class, there are 3 methods with the name `setBiodata()` but with different signatures.

```
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

- One of the characteristics of overloading is the different signature method
- **Method signature:** Quantity, data type and array of parameters
- Method signatures are called differently if:
 - ❑ The amount is different, or
 - ❑ The number is the same but the data type is different, or
 - ❑ The number and type of data are the same but the arrangement is different

Note that parameter names are not taken into account.

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;  
    }  
}
```

Amount, data type,
arrangement are
the same

Compile error:

Duplicate method setBiodata(String, int)

METHOD SIGNATURE (3)

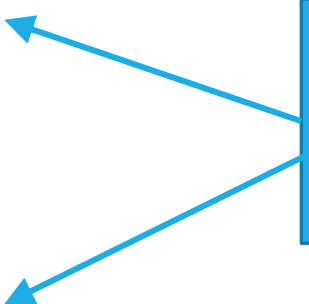
```
public class Mahasiswa {  
    public String nama;  
    public int umur;  
    public String alamat;  
  
    public void setBiodata(String nama, int umur) {  
        this.nama = nama;  
        this.umur = umur;  
    }  
  
    public void setBiodata(String alamat, int umur) {  
        this.alamat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

The number and type of data are the same, but the arrangement is different



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?

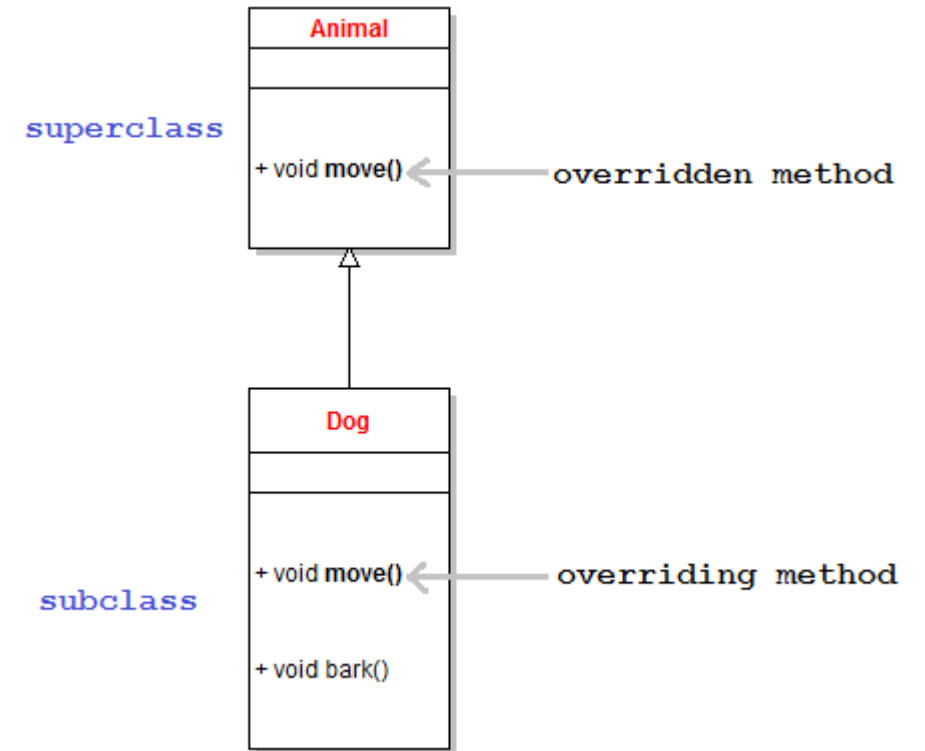
- Sometimes we need to create multiple methods that have similar functions/purposes, but with different parameters or different implementations
- Simplifying program code and bringing about consistency in method naming

OVERRIDING

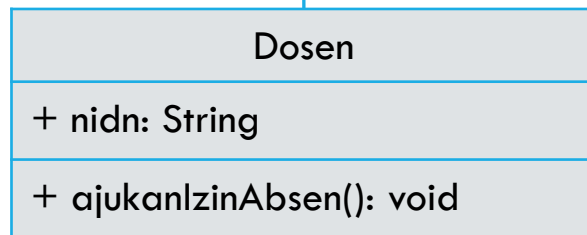
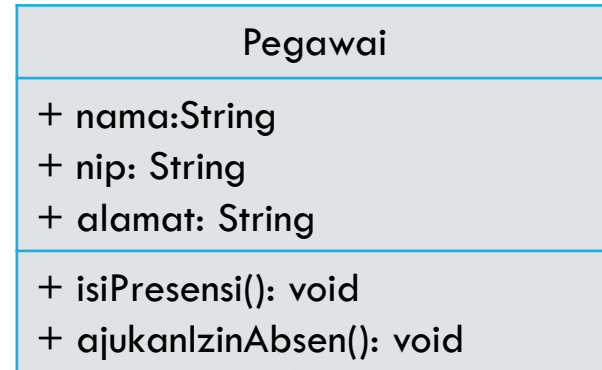
Overriding occurs when a subclass has a method with same name and signature as the superclass.

Characteristic:

1. Performed by methods in a subclass against method in a superclass
2. Same method name
3. Same method signature



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

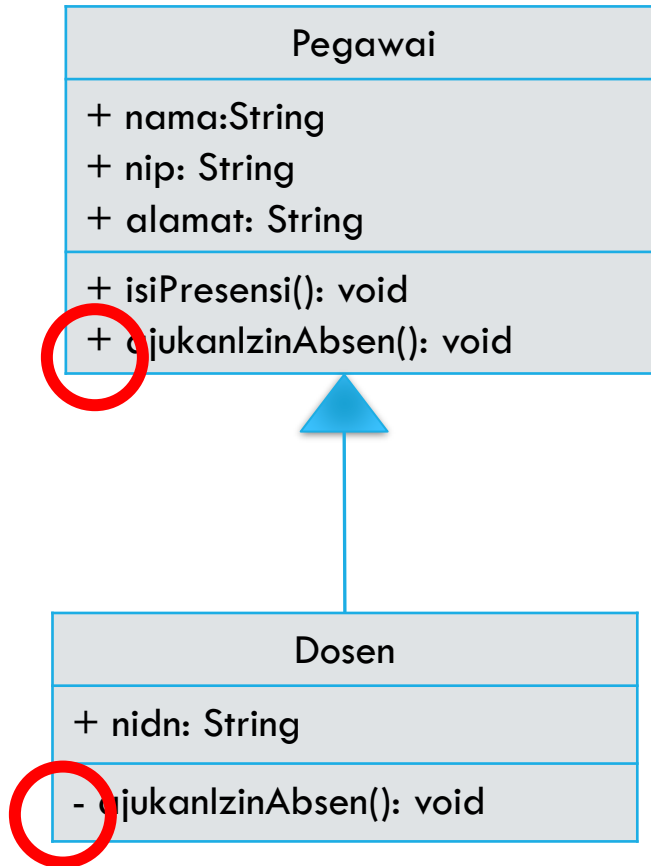


```
public void ajukanIzinAbsen() {  
    System.out.println("Menghubungi Biro Kepegawaian");  
}
```

```
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			



Overridden methods (methods in superclass) should not have a broader access level modifier than methods that override (methods in subclass).

If the method in the superclass uses public, then the method in the subclass should not use private because this will narrow the accessibility. Reducing the level of access from public to private will cause compilation errors because the overriding rule requires that the method that overrides has the same or broader level of access.

Superclass uses protected access level, subclass can be protected or public(broader)

If the rule is violated, a compile error will appear:
Cannot reduce the visibility of the inherited method from Pegawai

WHEN TO USE OVERRIDING

Overriding is used when there is a class that is a derivative of another class, but the implementation of a method is different

FINAL KEYWORD

The final keyword is a modifier used to make a class, attribute, or method immutable

- ✓ Final class → cannot be downgraded/extended
- ✓ Final attribute → The value cannot be modified
- ✓ Final method → cannot be overridden

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 → cannot be downgraded/extended

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 = "jsdfhdsd";  
        this.nama = nama;  
        this.gaji = gaji;  
    }  
  
    public void setNIP(String nip) {  
        this.nip = nip;  
    }  
}
```

Final attribute → The value cannot be modified

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 attribute → The value cannot be modified

FINAL METHOD

Final method → cannot be overridden

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

Cannot override the final method from Pegawai

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