



Inheritance



Topik

- Pengertian inheritance
- Deklarasi inheritance
- Single inheritance
- Multilevel inheritance
- Access Control
- Konstruktor tidak diwariskan
- super keyword



Pengertian dasar inheritance

- Inheritance (Pewarisan) merupakan salah satu dari tiga konsep dasar OOP.
- Konsep inheritance ini mengadopsi dunia riil dimana suatu entitas/obyek dapat mempunyai entitas/obyek turunan.
- Dengan konsep inheritance, sebuah class dapat mempunyai class turunan.



Pengertian dasar inheritance

- Suatu class yang mempunyai class turunan dinamakan **parent class** atau **base class**.
- Sedangkan class turunan itu sendiri seringkali disebut **subclass** atau **child class**.
- Suatu subclass dapat mewarisi apa-apa yang dipunyai oleh parent class.



Pengertian dasar inheritance

- Karena suatu subclass dapat mewarisi apa-apa yang dipunyai oleh parent class-nya, maka member dari suatu subclass adalah terdiri dari apa-apa yang ia punyai dan juga apa-apa yang ia warisi dari class parent-nya.
- Kesimpulannya, boleh dikatakan bahwa suatu subclass adalah tidak lain hanya memperluas (extend) parent class-nya.



Deklarasi inheritance

- Dengan menambahkan kata kunci **extends** setelah deklarasi nama class, kemudian diikuti dengan nama parent class-nya.
- Kata kunci extends tersebut memberitahu kompiler Java bahwa kita ingin melakukan perluasan class.



Deklarasi inheritance

```
public class B extends A {  
    ...  
}
```



- Semua class di dalam Java adalah merupakan subclass dari class super induk yang bernama **Object**.
- Misalnya saja terdapat sebuah class sederhana :

```
public class A {  
    ...  
}
```


- Pada saat dikompilasi, Kompiler Java akan membacanya sebagai subclass dari class Object.

```
public class A extends Object {  
    ...  
}
```



Kapan kita menerapkan inheritance?

- Kita baru perlu menerapkan inheritance pada saat kita jumpai ada suatu class yang dapat diperluas dari class lain.



Keuntungan Inheritance

- Keuntungan dari Inheritance adalah Reusability
 - Sekali perilaku(method) didefinisikan pada super class, maka perilaku tersebut secara otomatis diwariskan ke subclass. Sehingga hanya perlu menulis method sekali dan bisa digunakan untuk semua subclass.
 - Sekali properti/field di definisikan di superclass, maka semua properti di wariskan ke semua subclass. Superclass dan subclass berbagi properti
 - Subclass hanya perlu mengimplementasikan jika ada perbedaan dengan parentnya.

Misal terdapat class Pegawai

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



Misal terdapat class Manager

```
public class Manajer {  
    public String nama;  
    public double gaji;  
    public String departemen;  
}
```



- Dari 2 buah class diatas, kita lihat class Manajer mempunyai data member yang identik sama dengan class Pegawai, hanya saja ada tambahan data member departemen.
- Sebenarnya yang terjadi disana adalah class Manajer merupakan perluasan dari class Pegawai dengan tambahan data member departemen.
- Disini perlu memakai konsep inheritance, sehingga class Manajer dapat kita tuliskan seperti berikut



```
public class Manajer extends Pegawai {  
    public String departemen;  
}
```



Single Inheritance

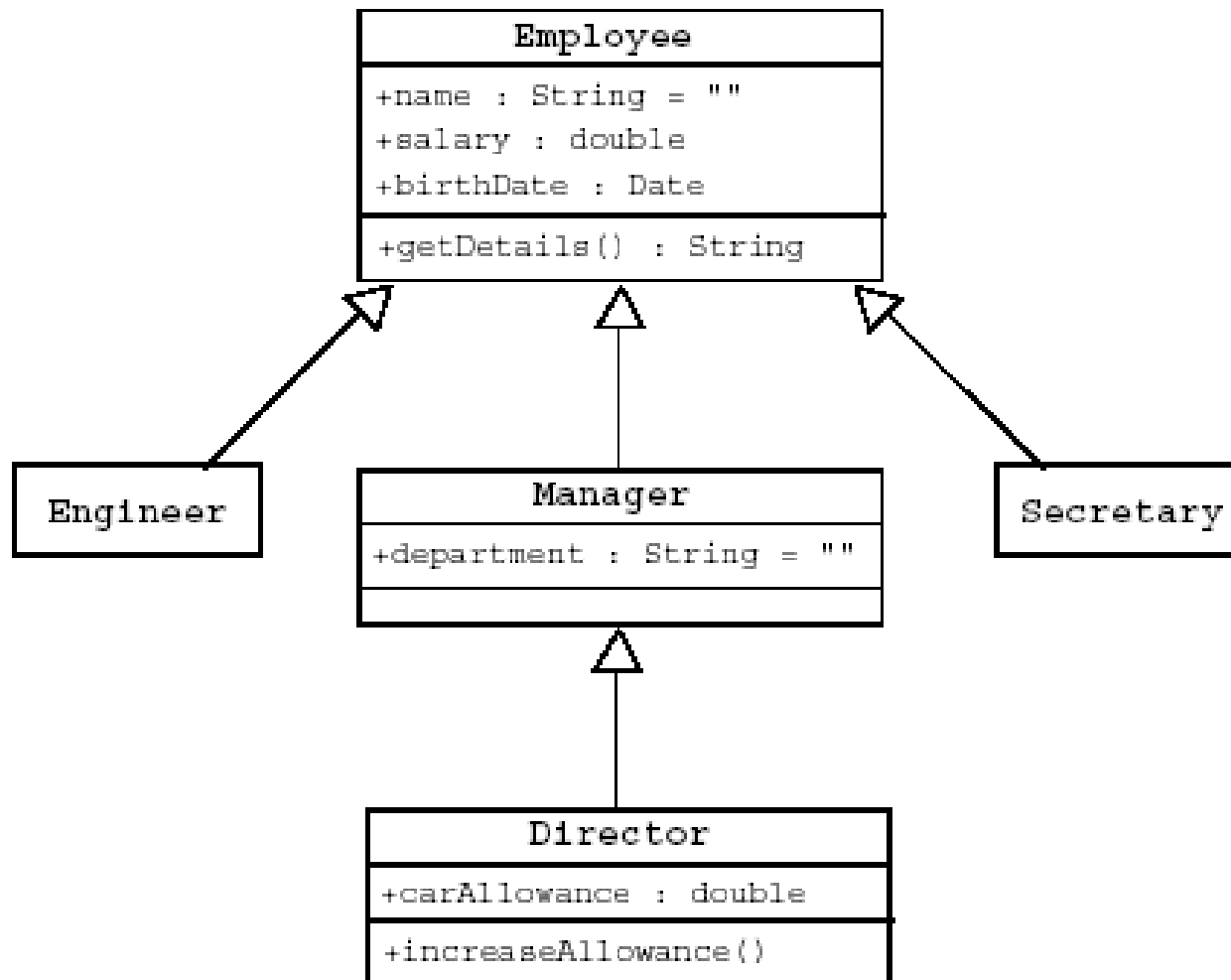
- Konsep inheritance yang ada di Java adalah Java hanya memperkenankan adanya **single inheritance**.
- Konsep single inheritance hanya memperbolehkan suatu subclass mempunyai satu parent class.



Multilevel Inheritance

- Konsep inheritance yang ada di Java memperkenalkan adanya **multilevel inheritance**.
- Konsep multilevel inheritance memperbolehkan suatu subclass mempunyai subclass lagi.

Single dan Multilevel Inheritance





Pengaksesan member yang dideklarasikan di parent class dari subclass

- Pengaksesan member yang ada di parent class dari subclass-nya tidak berbeda dengan pengaksesan member subclass itu sendiri.
- Misalnya di class Manajer kita ingin mengakses data member nama melalui sebuah function member IsiData(), sekaligus kita juga ingin mengakses data member departemen di class Manajer.



```
public class Manajer extends Pegawai {  
    public String departemen;  
  
    public void IsiData(String n, String d) {  
        nama=n;  
        departemen=d;  
    }  
}
```



KONTROL PENGAKSESAN



Kontrol pengaksesan

- Dalam dunia riil, suatu entitas induk bisa saja tidak mewariskan sebagian dari apa-apa yang ia punyai kepada entitas turunan karena sesuatu hal.
- Demikian juga dengan konsep inheritance dalam OOP.
- Suatu parent class dapat tidak mewariskan sebagian member-nya kepada subclass-nya.
- Sebagai contoh, kita coba untuk memodifikasi class Pegawai.



```
public class Pegawai {  
    private String nama;  
    public double gaji;  
}
```



- Coba untuk mengkompilasi class Manajer pada contoh sebelumnya.
- Apa yang terjadi?
- Pesan kesalahan akan muncul seperti ini :

```
Manajer.java:5: nama has private access in Pegawai
        nama=n;
```

- Ini membuktikan bahwa class Manajer tidak mewarisi data member nama dari parent class-nya (Pegawai).



Kontrol pengaksesan

Modifier	class yang sama	package yang sama	subclass package lain	class manapun
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√



private

- Variabel dan method yang dideklarasikan *private* hanya bisa diakses oleh class yg mendeklarasikan variabel dan method tersebut.



Example1:

Mengakses private variabel dari class lain

```
1. class Complex {
2.     private double real, imaginary;
3.
4.     public Complex(double r, double i) {
5.         real = r; imaginary = i;
6.     }
7.     public Complex add(Complex c) {
8.         return new Complex(real + c.real,
9.             imaginary + c.imaginary);
10.    }
11. }
12.
13. class Client {
14.     void useThem() {
15.         Complex c1 = new Complex(1, 2);
16.         Complex c2 = new Complex(3, 4);
17.         Complex c3 = c1.add(c2);
18.         double d = c3.real; // Illegal!
19.     }
20. }
```



Example2:

Mengakses private variabel dari subclass.

```
1. class Complex {
2.     private double real, imaginary;
3. }
4.
5.
6. class SubComplex extends Complex {
7.     SubComplex(double r, double i) {
8.         real = r; // Trouble!
9.     }
10. }
```



Default

- Bukan merupakan Java keyword.
- Merupakan jenis akses kontrol jika kita tidak menuliskan akses kontrol secara eksplisit.
- Semua feature class-class yang ada dalam satu package bisa diakses oleh semua yang ada dalam package tersebut.
- Class diluar package boleh melakukan subclass, tetapi subclass tersebut tidak bisa mengakses feature superclass.



Example1: default

```
1. package sportinggoods;  
2. class Ski {  
3.     void applyWax() { . . . } → default access  
4. }
```

```
1. package sportinggoods;  
2. class DownhillSki extends Ski {  
3.     void tuneup() {  
4.         applyWax(); → OK  
5.         // other tuneup functionality here  
6.     }  
7. }
```



Example1: default

```
1. package sportinggoods;  
2. class Ski {  
3.     void applyWax() { . . . } → default access  
4. }
```

```
1. package differentPackage;  
2. class DownhillSki extends Ski {  
3.     void tuneup() {  
4.         applyWax(); → error  
5.         // other tuneup functionality here  
6.     }  
7. }
```



protected

- Protected mempunyai kemampuan akses yang lebih besar daripada private dan default.
- Protected feature dari suatu class bisa diakses oleh semua class dalam satu package.
- Class diluar package boleh melakukan melakukan subclass, dan subclass tersebut bisa mengakses feature superclass.



Example: protected

```
1. package adifferentpackage; // Class Ski now in
// a different package
2. class Ski {
3.     protected void applyWax() { . . . }
4. }
```

```
1. package sportinggoods;
2. class DownhillSki extends Ski {
3.     void tuneup() {
4.         applyWax(); → OK
5.         // other tuneup functionality here
6.     }
7. }
```



Summary of Access Modes to Class Members

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From any class outside the package	Yes	No	No	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes	No	No



Kata kunci super

- Kata kunci super dipakai untuk merujuk pada member dari parent class.
- Sebagaimana kata kunci this yang dipakai untuk merujuk pada member dari class itu sendiri.
- Format penulisannya adalah sebagai berikut :
 - `super.data_member`
→ merujuk pada data member pada parent class
 - `super.function_member()`
→ merujuk pada function member pada parent class
 - `super()`
→ merujuk pada konstruktor pada parent class



Contoh

```
class Parent {
    public int x = 5;
}

class Child extends Parent {
    public int x = 10;

    public void Info(int x) {
        System.out.println("Nilai x sebagai parameter = " + x);
        System.out.println("Data member x di class Child = " + this.x);
        System.out.println("Data member x di class Parent = " + super.x);
    }
}

public class NilaiX {
    public static void main(String args[]) {
        Child tes = new Child();
        tes.Info(20);
    }
}
```



Hasil

- Nilai x sebagai parameter = 20
- Data member x di class Child = 10
- Data member x di class Parent = 5



Kesimpulan

- `x`
→ merujuk pada `x` terdekat, yaitu parameter `Info()`
- `this.x`
→ merujuk pada data member dari class-nya sendiri, yaitu data member pada class `Child`
- `super.x`
→ merujuk pada data member dari parent class-nya, yaitu data member pada class `Parent`



Konstruktor tidak diwariskan

- Konstruktor dari parent class tidak dapat diwariskan ke subclass-nya.
- Konsekuensinya, setiap kali kita membuat suatu subclass, maka kita harus memanggil konstruktor parent class di konstruktor subclass.
- Pemanggilan konstruktor parent harus dilakukan pada baris pertama dari konstruktor subclass.



Konstruktor tidak diwariskan

- Jika kita tidak mendeklarasikannya secara eksplisit, maka kompiler Java akan menambahkan deklarasi pemanggilan konstruktor parent class di konstruktor subclass.



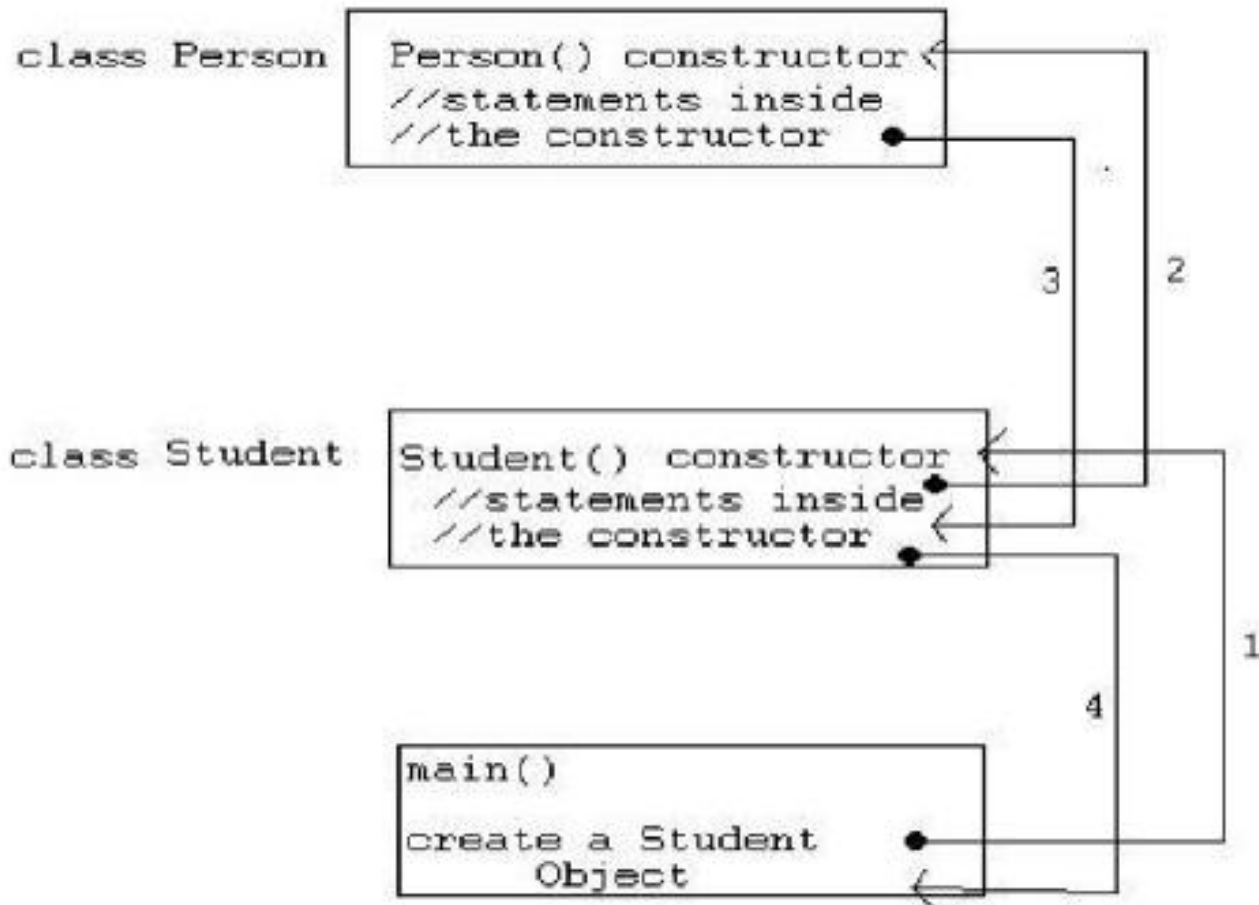
Konstruktur tidak diwariskan

- Sebelum subclass menjalankan konstruktornya sendiri, subclass akan menjalankan constructor superclass terlebih dahulu.
- Hal ini terjadi karena secara implisit pada constructor subclass ditambahkan pemanggilan `super()` yang bertujuan memanggil constructor superclass oleh kompiler.



- Pada saat program tersebut dikompilasi, maka kompiler Java akan menambahkan :
 - konstruktor class Parent
 - konstruktor class Child
 - pemanggilan konstruktor class Parent di konstruktor class Child

Pemanggilan konstruktor





Pemanggilan konstruktor

```
public class Parent  
{  
  
}
```

```
public class Child extends Parent {  
  
}
```

Pemanggilan konstruktor

- Sehingga program tersebut sama saja dengan yang berikut ini :

```
public class Parent {  
    public Parent() {  
  
    }  
}
```

```
public class Child extends Parent {  
    public Child() {  
        super();  
    }  
}
```

pemanggilan kostruktor class Parent

```
public class Child extends Parent {  
    int x;  
    public Child() {  
        x = 5;  
        super();  
    }  
}
```

X

```
public class Child extends Parent {  
    int x;  
    public Child() {  
        super();  
        x = 5;  
    }  
}
```

√



Contoh

```
public class Parent {
    String parentName;

    public Parent(String parentName) {
        this.parentName= parentName;
    }
}

class Baby extends Parent {
    public void Cry() {
        System.out.println("Owek owek");
    }
}
```

Selanjutnya bila kita membuat : `Baby bayi = new Baby()` → error!!