Inheritance (Pewarisan)

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. Suatu class yang mempunyai class turunan dinamakan parent class atau base class. Sedangkan class turunan itu sendiri seringkali disebut subclass atau child class.

Dalam dunia riil, suatu entitas turunan dapat mewarisi apa-apa yang dipunyai dari entitas induknya. Misalkan saja antara entitas Bapak dan entitas Anak. Entitas anak dapat mewarisi apa-apa yang dipunyai oleh entitas Bapaknya. Demikian juga dalam konsep inheritance, suatu subclass dapat mewarisi apa-apa yang dipunyai oleh parent class. Inilah yang terpenting dari konsep 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

Di dalam Java untuk mendeklarasikan suatu class sebagai subclass cukup mudah, yaitu 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. Berikut adalah contoh deklarasi inheritance : –> class B adalah subclass dari class A

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

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

```
public class A{

….

}
```

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

```
public class A extends Object{

….

}
```

Single inheritance

Satu hal penting yang harus kita ketahui dari konsep inheritance yang ada di Java adalah Java hanya memperkenankan adanya single inheritance. Konsep single inheritance hanya memperbolehkan suatu sublass mempunyai satu parent class. Dengan konsep single inheritance ini, masalah pewarisan akan dapat diamati dengan mudah. Inilah salah satu keunggulan OOP dalam Java.

Kapan kita menerapkan inheritance?

Pertanyaan diatas seringkali muncul bagi siapa saja yang mempelajari konsep OOP. Pertanyaan tersebut dapat terjawab jika kita kembali pada konsep dasar inheritance. Dalam konsep dasar inheritance dikatakan bahwa suatu subclass adalah tidak lain hanya memperluas (extend) parent class-nya. Nah, jawaban yang tepat untuk menjawab pertanyaan diatas adalah kita baru perlu menerapkan inheritance pada saat kita menjumpai ada suatu class yang perlu memperluas class lain yang sudah ada.
Masih bingung?. Baiklah untuk lebih mudahnya kita ikuti saja ilustrasi berikut. Misalnya kita mempunyai suatu class sederhana bernama Pegawai sebagai berikut :

```
public class Pegawai {

public String nama;

public double gaji;

}
```

Selain class tersebut, kita juga mempunyai sebuah class Manajer yang tampak sebagai berikut :

```
public class Manajer {

public String nama;

public double gaji;

public String departemen;

}
```

Dari 2 buah class di atas, 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. Disinilah kita perlu memakai konsep inheritance, sehingga class Manajer dapat kita tuliskan seperti berikut :

```
public class Manajer extends Pegawai {

public String departemen;

}
```

Pada saat class Manajer menurunkan atau memperluas (extend) class Pegawai, maka ia mewarisi data member yang dipunyai oleh class Pegawai. Dengan demikian, class Manajer mempunyai data member yang diwarisi oleh dari class Pegawai (atribut nama & gaji), ditambah dengan data member yang ia punyai (atribut departemen).

Pengaksesan member dari parent class

Pengaksesan member yang ada di parent class dari subclass-nya tidak jauh berbeda dengan pengaksesan member subclass itu sendiri. Misalnya di class Manajer kita ingin mengakses data member nama di class Pegawai melalui sebuah function member isiData(), sekaligus kita juga ingin mengakses data member departemen di class Manajer. Kita dapat menuliskannya seperti dibawah ini :

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

}
```

Dapat kita lihat bahwa pengaksesan member dari parent class (nama) adalah tidak berbeda dengan pengaksesan member dari subclass-nya sendiri (departemen).

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 seperti dibawah ini :

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

Sekarang kita coba untuk mengkompilasi class Manajer pada contoh sebelumnya. Apa yang terjadi?. Pesan kesalahan akan muncul seperti ini :

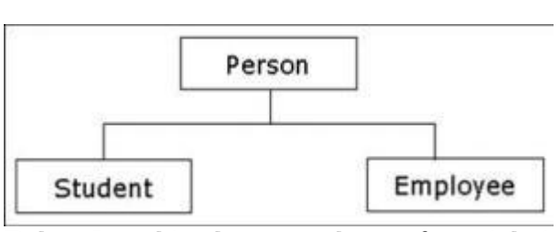
```
Manajer.java:5: nama has private access in Pegawai
nama=n;
```

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

POLYMORPHISM

Polymorphism adalah suatu object dapat memiliki berbagai bentuk, sebagai object dari class sendiri atau object dari superclassnya.

Kemampuan sebuah variabel reference untuk merubah behavior sesuai dengan apa yang dipunyai object.
Polymorphism 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.
Untuk menjelaskan polymorphism, mari kita bahas sebuah contoh.
Berikut adalah hirarki class nya.



Dalam Java, kita dapat membuat referensi dari suatu superclass ke object dari subclassnya. Sebagai contoh,

public static main(String[] args) {
Person ref;
Student studentObject = new Student();
Employee employeeObject = new Employee();
ref = studentObject; //titik referensi Person kepada // sebuah object Student
}
misalnya, kita memiliki sebuah method getName dalam superclass Person. Dan kita meng-override method ini di kedua subclass yaitu Student dan Employee.
public class Student {
public String getName(){
System.out.println("Student Name:" + name);
return name;
}
}
public class Employee {
public String getName(){
System.out.println("Employee Name:" + name);
return name;
}
}
Kembali ke method utama kita, ketika kita mencoba memanggil method getName dari referensi Person ref, method getName dari object Student akan dipanggil.
Sekarang, jika kita memberi ref kepada object Employee, maka method getName juga akan dipanggil
public static main(String[] args) {

Person ref;

Student studentObject = new Student();

Employee employeeObject = new Employee();

ref = studentObject; //titik referensi Person kepada object Student //getName dari class Student dipanggil

String temp=ref.getName();
System.out.println(temp);

ref = employeeObject; //titik referensi Person kepada object Employee

//getName dari class Employee dipanggil
String temp = ref.getName();
System.out.println(temp);
}
Contoh lain yang menggambarkan polymorphism adalah ketika kita mencoba untuk passing reference kepada method. jika kita memiliki sebuah method static *printInformation* yang menerima referensi Person sebagai parameter.
public static printInformation(Person p){
. . . .
}

Sebenarnya kita dapat passing reference dari Employee dan Student kepada method *printInformation* selama kedua class tersebut merupakan subclass dari Person.
public static main(String[] args)
{
Student studentObject = new Student();
Employee employeeObject = new Employee();

printInformation(studentObject);
printInformation(employeeObject);
}

Overloading

Overloading : penggunaan satu nama untuk beberapa method yang berbeda (beda parameter)

LuasBeraksi.java | Luas.java

```
class Luas{
    double panjang, lebar, tinggi;
    //konstruktor pertama
    Luas(double pj){
        this.panjang=pj;
        this.lebar=1;
        this.tinggi=1;
    }
    //konstruktor kedua
    Luas(double pj,double lb){
        this.panjang=pj;
        this.lebar=lb;
        this.tinggi=1;
    }
    //konstruktor ketiga
    Luas(double pj,double lb,double tg){
        this.panjang=pj;
        this.lebar=lb;
        this.tinggi=tg;

    }
}
```

LuasBeraksi.java * | Luas.java

```
public class LuasBeraksi{
    public static void main (String args[]){
    System.out.println("Panggil konstruktor pertama");
        Luas bentuk=new Luas(3.5);
        System.out.println("Panjang = "+bentuk.panjang);
        System.out.println("Lebar = "+bentuk.lebar);
        System.out.println("Tinggi = "+bentuk.tinggi);


    System.out.println("Panggil konstruktor kedua");
        Luas bentuk2=new Luas(7.2,2.3);
        System.out.println("Panjang = "+bentuk2.panjang);
        System.out.println("Lebar = "+bentuk2.lebar);
        System.out.println("Tinggi = "+bentuk2.tinggi);


    System.out.println("Panggil konstruktor ketiga");
        Luas bentuk3=new Luas(5.2,3.5,2.1);
        System.out.println("Panjang = "+bentuk3.panjang);
        System.out.println("Lebar = "+bentuk3.lebar);
        System.out.println("Tinggi = "+bentuk3.tinggi);

    }
}
```

hasil dari script diatas adalah dibawah ini:

Overriding

Overriding : terjadi ketika deklarasi method subclass dengan nama dan parameter yang sama dengan method dari superclassnya.

mobilBergerak.java | kijang.java | LuasBeraksi.java | Luas.java

```
class mobilBergerak{
    private String merek;
    mobilBergerak(String m){
    this.merek=m;
    System.out.println(this.merek+" adalah kendaraan beroda 4");
    }
    //procedure bergerak
    void bergerak(String arah){
    System.out.println("melaju ke arah "+arah);
}
//overload procedure bergerak
void bergerak(String arah, int jauh){
    System.out.println("melaju ke arah "+arah+" sejauh "+jauh+" km");
}
}
```

mobilBergerak.java | kijang.java | LuasBeraksi.java | Luas.java

```
public class kijang{
    public static void main(String args[]){
        mobilBergerak kijang=new mobilBergerak("Kijang");
        kijang.bergerak("maju");
        kijang.bergerak("mundur",3);
    }
}
```

hasil dari script diatas adalah dibawah ini: