



PBO : ABSTRACT CLASS DAN INTERFACE

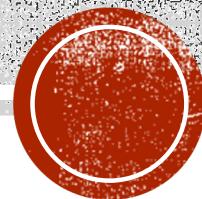
Materi 6

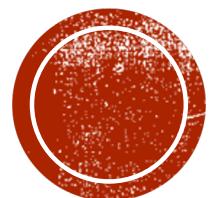
Royana Afwani

Teknik
Informatika FT
Unram

INHERITANCE (LANJUTAN)

Abstract Class
Interface
Inner Class





KELAS ABSTRAK

Abstract method, Abstract Class

ABSTRACT METHOD

- Method yang tidak memiliki implementasi (body)
- Untuk membuat abstract method, hanya tuliskan deklarasi method tanpa bodi, dan gunakan **abstract** keyword
 - tidak ada { }
- Sebagai contoh:
 - `//ingat tidak ada { }`
 - `public abstract void someMethod();`



ABSTRACT CLASS

- Abstract class adalah class yang memiliki **minimal satu abstract method**
- Digunakan apabila kelas induk yang menjadi sumber pewarisan tidak ingin diolah.
- Abstract class tidak bisa dibuat instance-nya (tidak bisa dibuat objectnya)
 - //kode di bawah ini menyebabkan compilation error

```
MyAbstractClass a1 = new MyAbstractClass();
```

- Untuk menggunakan abstract class, digunakan class lain (**concrete class**)
 - concrete class harus mengimplementasi semua abstract method
 - concrete class menggunakan **extends keyword**



CONTOH ABSTRACT CLASS

```
public abstract class MakhlukHidup {  
    public void bernafas(){  
        System.out.println("Makhluk hidup bernafas...");  
    }  
    public void makan(){  
        System.out.println("Makhluk hidup makan...");  
    }  
    /**  
     * Abstract method berjalan()  
     * Kita ingin mengimplementasi method ini melalui Concrete Class  
     */  
    public abstract void berjalan();  
}
```



EXTENDING ABSTRACT CLASS

- Ketika concrete class mengimplementasi abstract class **MakhlukHidup**, maka concrete class tersebut harus mengimplementasi (mengoverride) method **berjalan()**. Jika tidak, maka class tersebut akan jadi abstract class juga, sehingga tidak dapat diinstansiasi / dibuat objectnya.
- Contohnya

```
public class Manusia extends MakhlukHidup {  
    public void berjalan(){  
        System.out.println("Manusia berjalan...");  
    }  
}
```



KAPAN MENGGUNAKAN ABSTRACT METHOD DAN ABSTRACT CLASS?

- Abstract method digunakan ketika dua atau lebih subclass dirancang untuk memenuhi aturan yang sama dengan implementasi yang berbeda
 - Subclass-subclass ini mengextend abstract class yang sama namun dengan implementasi yang berbeda pada abstract methodnya.
- Menggunakan abstract class sebagai konsep umum pada top level class hierarki, dan menggunakan banyak subclass untuk membuat detail implementasi abstract class



CONTOH IMPLEMENTASI

```
public abstract class Printer {  
    public abstract void printing();  
    public void scanner(){  
    }  
}
```

```
public class Canon extends Printer{  
    public void printing(){  
        System.out.println("nge-print pakai tinta");  
    }  
}
```

```
public class Samsung extends Printer {  
    public void fotocopy(){  
        System.out.println("Bisa buat fotocopy");  
    }  
    public void printing(){  
        System.out.println("nge-print pakai carbon");  
    }  
}
```



CONTOH LAIN: ABSTRACT CLASS

Hewan.java

```
abstract class Hewan {  
    protected String nama;  
    protected int jumKaki;  
    protected boolean bisaTerbang = false;  
    public Hewan(String nama, int kaki, boolean terbang) {  
        this.nama = nama;  
        jumKaki = kaki;  
        bisaTerbang = terbang;  
    }  
    public abstract void bersuara();  
    public static void makan() {  
        System.out.println("nyam, nyam, nyam");  
    }  
    public void isHewan() {  
        System.out.println("nama : "+nama);  
        System.out.println("jumlah kaki : "+jumKaki);  
        System.out.println("bisa terbang : "+bisaTerbang);  
    }  
}
```



Perkutut.java

```
class Perkutut extends Hewan {  
    public Perkutut() {  
        super("perkutut", 2, true);  
    }  
    public void bersuara() {  
        System.out.println("\ncuit, cuit, cuit");  
    }  
    public static void main(String[] args) {  
        Perkutut p = new Perkutut();  
        p.isHewan();  
        p.bersuara();  
    }  
}
```

Output :

```
nama : perkutut  
jumlah kaki : 2  
bisa terbang : true  
cuit, cuit, cuit
```



Sapi.java

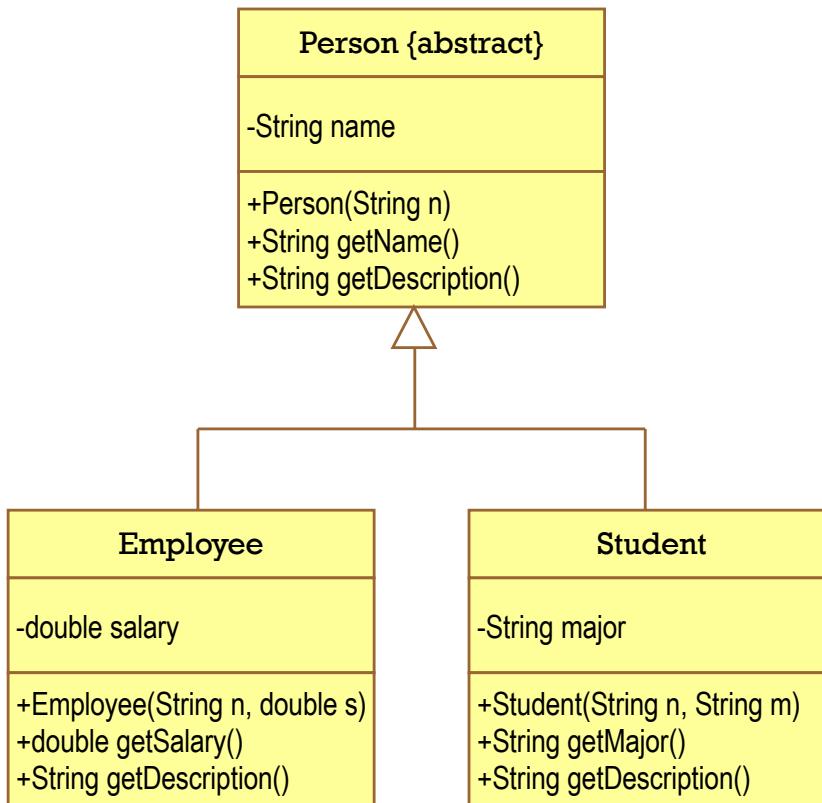
```
class Sapi extends Hewan {  
    public Sapi() {  
        super("sapi", 4, false);  
    }  
    public void bersuara() {  
        System.out.println("\nemoh..., emoh..");  
    }  
    public static void main(String[] args) {  
        Sapi s = new Sapi();  
        s.isHewan();  
        s.bersuara();  
    }  
}
```

Output :

```
jumlah kaki : 4  
bisa terbang : false  
emohà,emoh..
```



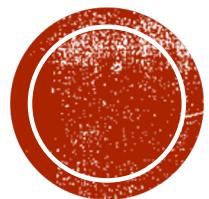
CONTOH PENGGUNAAN KELAS ABSTRAK



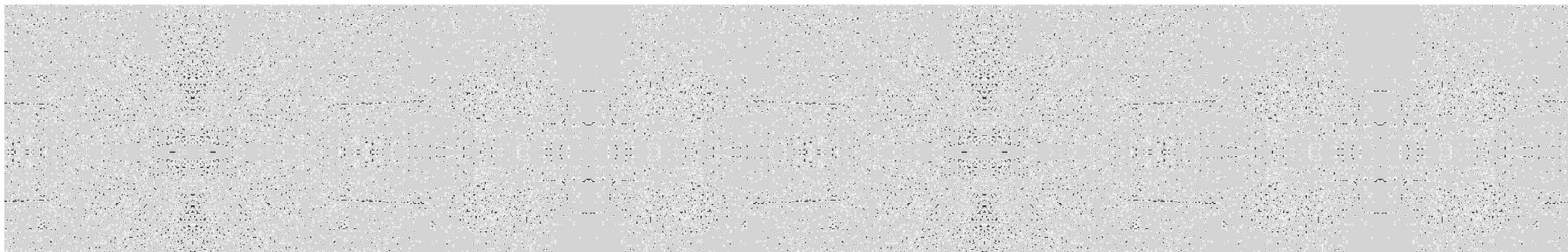
Kode Program

- [Person.java](#)
- [Employee.java](#)
- [Student.java](#)
- [CobaKelasAbstrak.java](#)





PEWARISAN JAMAK DAN INTERFACE



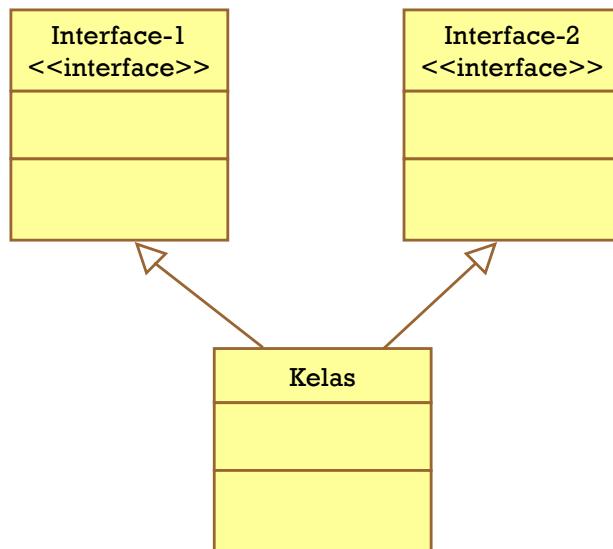
PEWARISAN JAMAK DALAM JAVA

- Java **TIDAK** mendukung pewarisan Jamak
- Java **HANYA** memiliki *pewarisan tunggal*
 - Suatu kelas turunan hanya memiliki **satu kelas induk**
- Lalu Bagaimana perwujudannya dalam Java ?
- Java mewujudkannya dengan menggunakan *Interface*



PEWARISAN JAMAK DALAM JAVA : INTERFACE

- Sebuah kelas dapat menerima pewarisan dari beberapa interface (multiple inheritance).



```
class Kelas implements  
Interface1, Interface2 {  
  
    // Atribut  
  
    // Metdod  
}
```



INTERFACE

- **Interface** digunakan apabila kita ingin menentukan apa yang harus dilakukan oleh suatu class tapi tidak menentukan bagaimana cara untuk melakukannya
- *Interface* kumpulan deklarasi fungsi (tanpa implementasi). Interface juga dapat mendeklarasikan konstanta
- **Interface** sebenarnya sama dengan class, tapi hanya memiliki deklarasi method tanpa implementasi



DEKLARASI INTERFACE

Bentuk umum deklarasi:

```
[modifier] interface NamaInterface {  
    // deklarasi konstanta  
    // deklarasi method  
}
```

Catatan : modifier static tidak boleh digunakan dalam interface



IMPLEMENTASI INTERFACE

Bentuk umum implementasi :

```
[modifier] class NamaKelas implements  
NamaInterface {  
    // penggunaan konstanta  
    // implementasi method  
}
```



INTERFACE

- Prototype kelas yang berisi **definisi konstanta** dan **deklarasi method** (hanya nama method tanpa definisi kode programnya).

```
interface Perusahaan {  
    // Konstanta  
    String KOTA = "Bandung";  
  
    // Deklarasi method  
    void setNama(String n);  
    void setAlamat(String a);  
    String getNama();  
    String getAlamat();  
}
```



INTERFACE (LANJUTAN)

- Dalam sebuah interface:
 - Semua konstanta (atribut) adalah **public**, **static** dan **final**.
 - Semua method adalah **abstract** dan **public**.
 - Tidak boleh ada deklarasi konstruktor.
- Interface digunakan untuk menyatakan **spesifikasi fungsional** beberapa kelas secara umum.



INTERFACE (LANJUTAN)

- Definisi method dilakukan di kelas yang menjadi implementasi dari interface.

```
class Telkom implements Perusahaan {  
    // Deklarasi atribut  
    private String nama, alamat;  
  
    // Definisi method  
    public void setNama(String n) {  
        nama = n;  
    }  
    ...  
}
```



CONTOH INTERFACE

- Interface : StartStop.java

```
interface StartStop {  
    void start();  
    void stop();  
}
```



CONTOH INTERFACE

- Kelas implementasi : UseStartStop.java

```
class Car implements StartStop {  
    private String name;  
  
    Car (String name) {  
        this.name = name;  
    }  
  
    public void start() {  
        System.out.println ("Insert key into ignition and turn.");  
    }  
  
    public void stop() {  
        System.out.println ("Turn key and remove from ignition.");  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```



CONTOH INTERFACE

- Kelas implementasi : UseStartStop.java

```
class WashingMachine implements StartStop {  
    private String name;  
  
    WashingMachine (String name) {  
        this.name = name;  
    }  
  
    public void start() {  
        System.out.println ("Push button and turn right.");  
    }  
  
    public void stop() {  
        System.out.println ("Turn left and push button.");  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```



CONTOH INTERFACE

- Kelas implementasi : UseStartStop.java

```
class UseStartStop {  
    public static void main (String [] args) {  
        Car c = new Car("Impala");  
        System.out.println(c.getName());  
        System.out.print("Start: ");  
        c.start();  
        System.out.print("Stop : ");  
        c.stop();  
  
        WashingMachine w = new WashingMachine("Tzu Zian");  
        System.out.println("\n"+w.getName());  
        System.out.print("Start: ");  
        w.start();  
        System.out.print("Stop : ");  
        w.stop();  
    }  
}
```



INTERFACE VS ABSTRACT CLASS

- Perbedaan *interface* dan *abstract class* cukup terlihat dari pemakaianya. *interface* itu diimplementasikan dan *abstract class* itu diturunkan (diwariskan).

	<i>Variables</i>	<i>Constructors</i>	<i>Methods</i>
Abstract Class	Bebas, tidak ada batasan, no restriction	Konstruktor dapat dipanggil melalui subclass melalui rantai konstruktor. Tidak dapat dibuat objeknya.	Bebas, tidak ada batasan, no restriction Minimal ada satu method abstract
Interfaces	Semua variable harus dideklarasi public static final	Tidak ada konstruktor. Tidak dapat dibuat objeknya.	Semua fungsi harus dideklarasi public abstract



INTERFACE VS ABSTRACT CLASS (LANJUTAN)

- Java hanya mengijinkan *single inheritance* untuk class, tetapi dapat *multiple* untuk *interfaces*.

```
public class NewClass extends BaseClass implements Interface1, ..., InterfacesN {  
    // ....  
}
```

- Sebuah interfaces dapat diturunkan dari kelas interfaces lainnya dengan menggunakan keyword **extends**.

```
public interface NewClass extends Interface1, ..., InterfacesN {  
    // konstanta dan abstract method  
}
```



INHERITANCE PADA INTERFACE

- Interface dapat memiliki hubungan inheritance antar mereka sendiri. Menggunakan keyword “**extends**”.
- Contoh :

interface PersonInterface merupakan superinterface dari student interface.

```
public interface PersonInterface {  
    . . .  
}  
  
public interface StudentInterface extends PersonInterface  
{  
    . . .  
}
```



IMPLEMENTS SUBINTERFACE

- Sebuah class yang mengimplementasikan subinterface wajib mendeklarasi ulang seluruh method yang ada pada subinterface tersebut dan juga method yang ada pada superinterface-nya.
- Contoh :

```
Class Mahasiswa implements StudentInterface {  
    . . .  
}
```

- Karena **StudentInterface** merupakan subclass dari PersonInterface maka class Mahasiswa harus mendeklarasi ulang semua method abstract yang dimiliki oleh kedua interface tersebut.



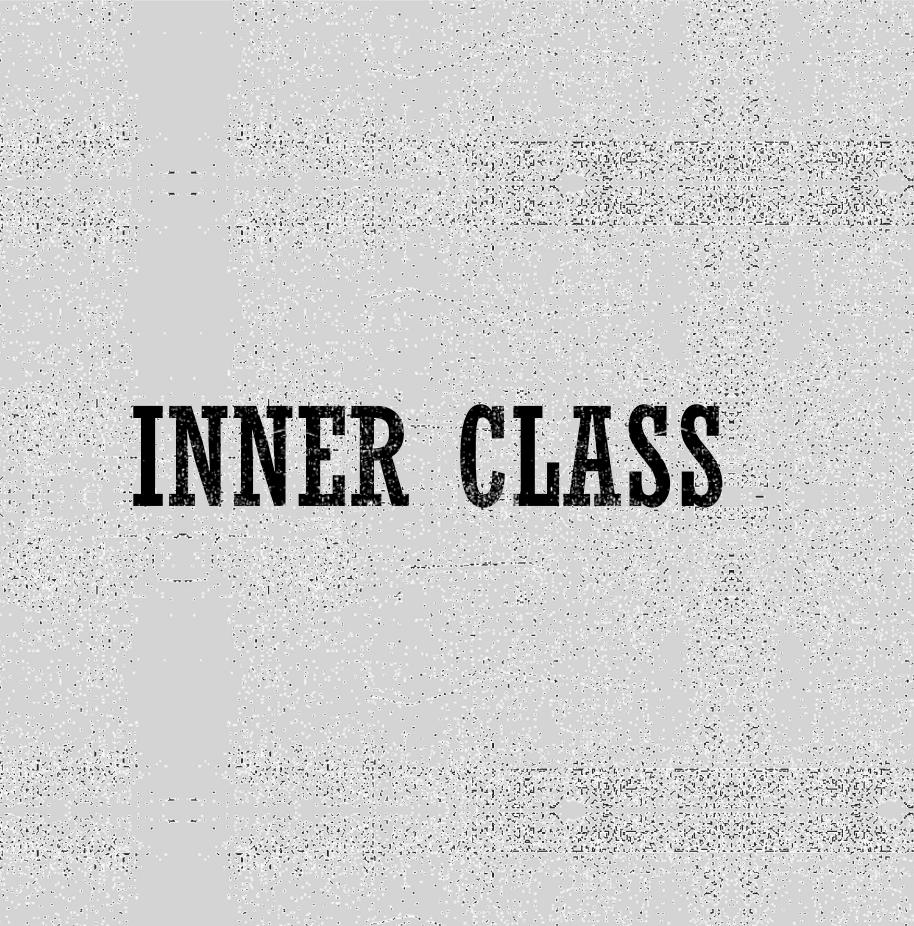
CONTOH MULTIPLE-IMPLEMENT

```
public interface PersonInterface {  
    public void setNama(String nama);  
}
```

```
public interface StudentInterface extends  
PersonInterface{  
    public void setSekolah(String sekolah);  
}
```

```
public abstract class Mahasiswa implements StudentInterface  
{  
  
    public abstract void isiBiodata(String nama, int nilai);  
  
    public void setSekolah(String skul){  
  
    }  
    public void setNama (String nam){  
  
    }  
}
```

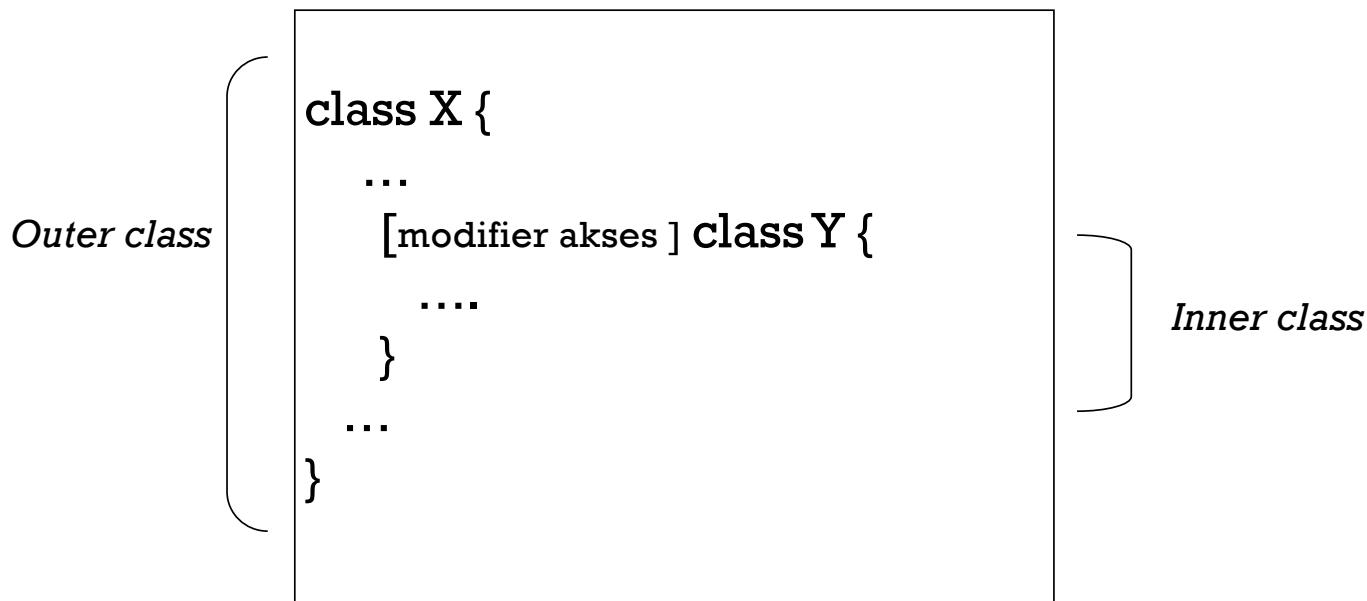




INNER CLASS

INNER KELAS

- Class yang dideklarasikan di dalam class



- Menghasilkan dua file *.class, yaitu
Outer.class dan *Outer\$Inner.class*



X.class
X\$Y.class



INNER KELAS

- Inner class merupakan bagian dari kelas, maka **diperlakukan seperti member class lainnya**
- Inner dapat **mengakses semua member** dari outer class, begitu juga sebaliknya
- Inner class digunakan selayaknya class normal, tapi inner class ini dapat digunakan **di luar outer classnya** tergantung modifier aksesnya.



CONTOH INNER KELAS

```
class Buku {  
    private int nomor;  
    private String judul;  
    class Bab {  
        private int noBab;  
        public String getBab() {  
            return "nomor buku: "+nomor+", judul buku: "+judul+", nomor Bab: "+noBab;  
        }  
    }  
    private Bab bab = new Bab();  
    public void setBab(int nomor, String judul, int noBab) {  
        this.nomor = nomor;  
        this.judul = judul;  
        bab.noBab = noBab;  
    }  
    public void cetak(){  
        System.out.println(bab.getBab());  
    }  
}
```



CONTOH INNER KELAS

```
class DemoBuku {  
    public static void main(String[] args) {  
        Buku buku = new Buku();  
  
        buku.setBab(10,"PBO 1",6)  
        buku.cetak();  
    }  
}
```



TERIMA KASIH

