



# Pemrograman Berorientasi Objek

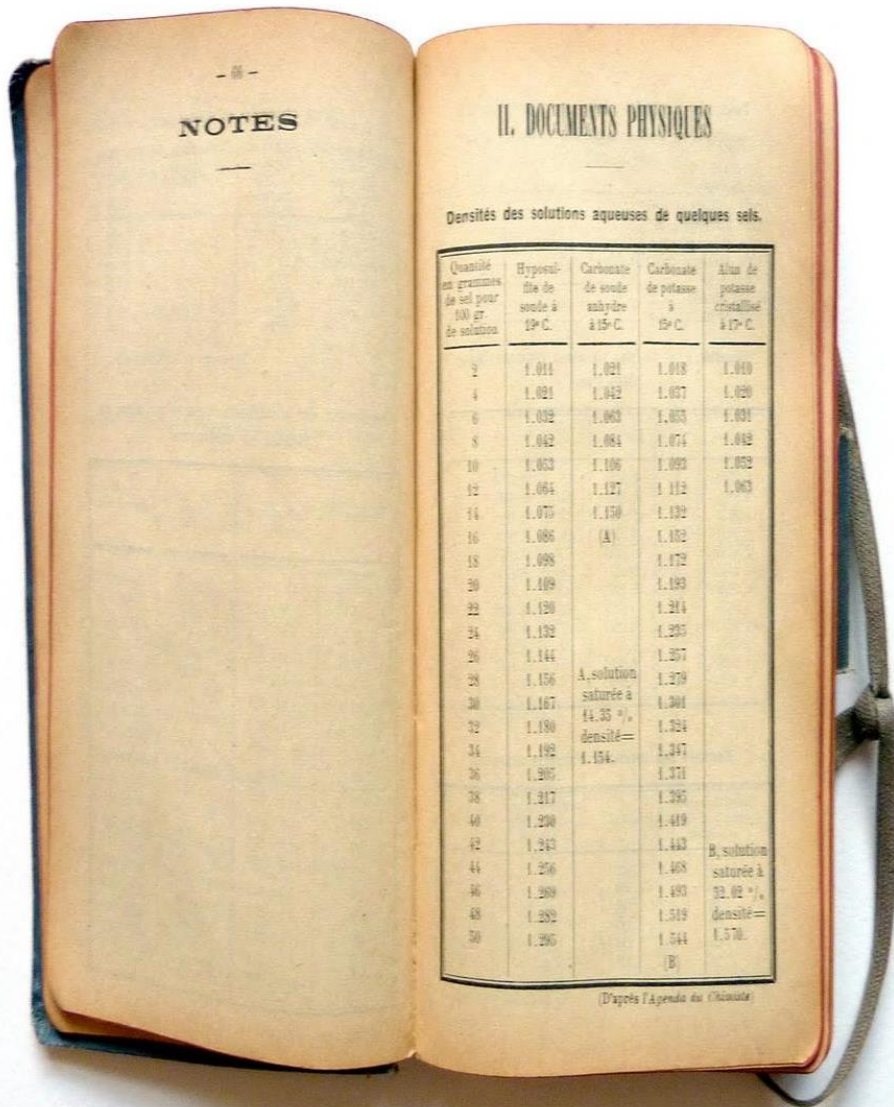


**Pertemuan 9 & 10**

***Abstract Class dan Interface***

Pemateri : Chrismikha Hardyanto S.Kom., M.Kom.

# KONTEN PERKULIAHAN



- Konsep Dasar Abstract Class
- Membuat Abstract Class pada JAVA
- Abstract Method dan implementasinya
- Konsep Dasar Interface
- Membuat Interface pada JAVA

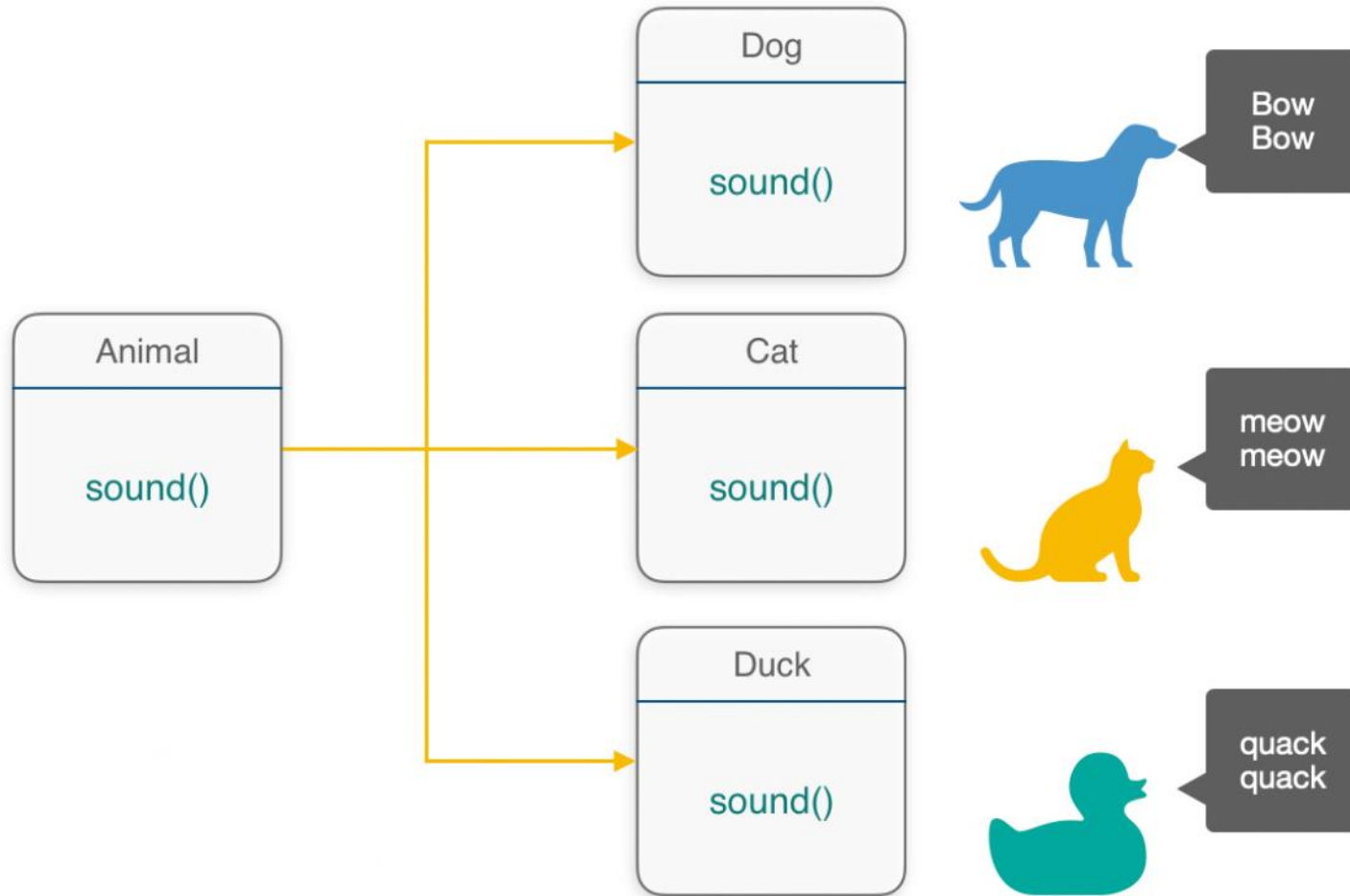
Didalam PBO, Ada konsep yang sangat membantu dalam merancang **behavior** suatu kelas didalam program yaitu : **Abstract Class & Interface**

# Latar Belakang

- ❑ Memungkinkan untuk menerapkan konsep **Abstraksi** pada pembuatan class didalam Program
- ❑ konsep **Abstraksi** merupakan prinsip dasar PBO yang menekankan pada bagaimana **menyembunyikan kompleksitas (implementasi kode)** dari suatu class dan hanya menampilkan **informasi yang relevan saja** dari kelas tersebut (terhadap kebutuhan program).
- ❑ Keuntungannya Kode didalam suatu Class dapat menjadi lebih **sederhana** Dimana Detail kode lainnya akan **diimplementasikan pada class - class yang terpisah (ingat kembali Konsep Inheritance & Polymorphm)**
- ❑ Masalahnya akan ada class yang nantinya **TIDAK LAYAK** untuk diakses isinya (dengan dibuat objeknya)



# Ilustrasi Konsep Abstraksi



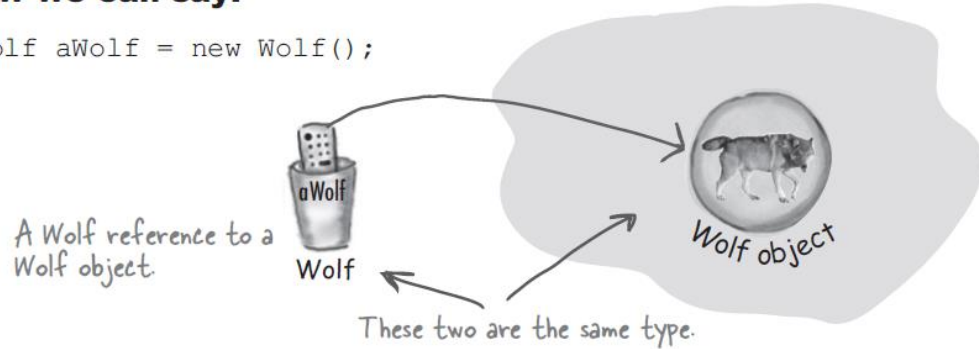
- ❑ Sebelumnya kita sudah tahu bahwa semua hewan (animal) dapat **bersuara**
- ❑ Bentuk / Bagaimana suaranya bergantung dengan **jenis dari hewan** yang bersuara
- ❑ Lalu bagaimana suara dari **Class Animal** ?? Apakah Ada yg tahu  
Kita tidak akan tahu suara dari animal **selama object animal yg dimaksud masih berupa sesuatu yg abstrak**



# Ilustrasi Konsep Abstraksi

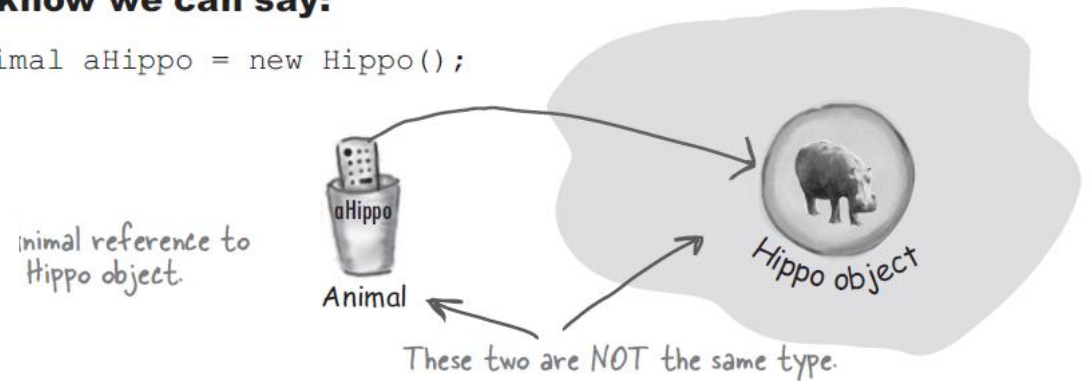
**We know we can say:**

```
Wolf aWolf = new Wolf();
```



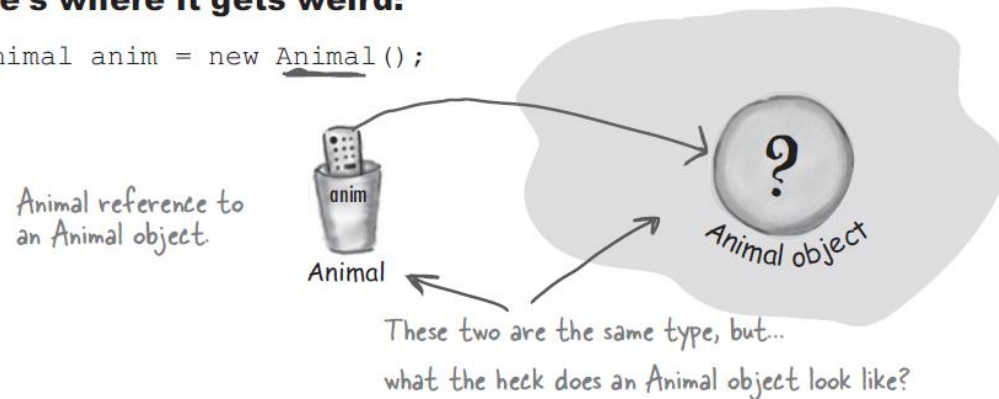
**And we know we can say:**

```
Animal aHippo = new Hippo();
```



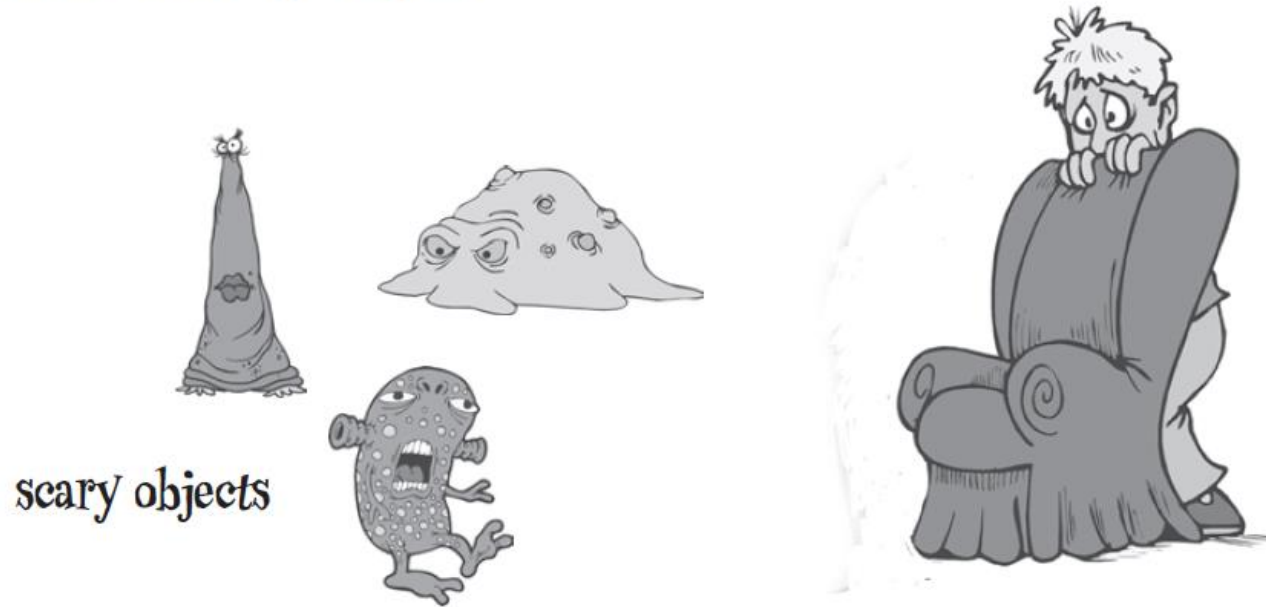
**But here's where it gets weird:**

```
Animal anim = new Animal();
```



# Ilustrasi Konsep Abstraksi

What does a new **Animal()** object *look* like?

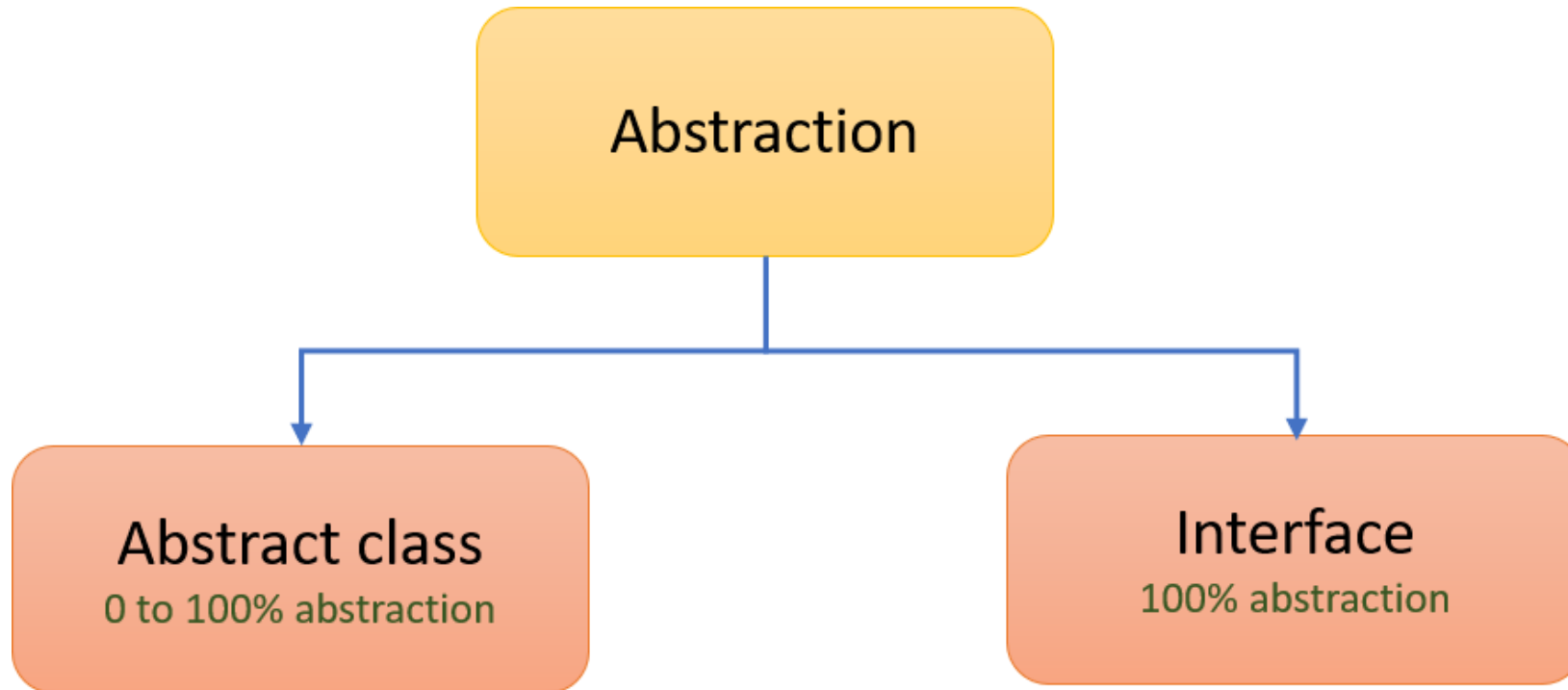


Some classes just should not be instantiated!

- ❑ Ada kalanya kita membuat class dimana implementasinya didalam program **masih berupa sesuatu yang abstract** dan baru diketahui aksi yg dijalankan ketika **diimplementasikan di CLASS lainnya**
- ❑ Maka kelas tersebut **TIDAK LAYAK DIBUATKAN OBJECT**

# Bagaimana Solusinya ?

- ❑ Untuk menangani masalah tersebut , didalam pemrograman object kita dapat menerapkan yang namanya **class abstract** & **interface**



- ❑ **KEUTUNGANNYA APA ?** Memungkinkan kita membuat Class yang berfungsi sebagai **KONTRAK** atau **TEMPLATE** untuk perilaku Class – Class lainnya\*



# Bagian 1:

## Abstract Class

# Konsep *Abstract Class*

- ❑ **Abstract class** merupakan Class yang tidak bisa dibuat **objek-nya** secara langsung (**instansiasi**) didalam program
- ❑ Isi class abstract sama saja dengan class pada umumnya. Bedanya kita bisa membuat **abstract method**
- ❑ Untuk bisa menggunakan **isi** dari abstract class, maka class tersebut **harus diturunkan (extends)** ke class lain yang **bukan abstract**
- ❑ Dengan demikian abstract class bisa kita gunakan sebagai **kontrak/template** untuk kelas anaknya

{OOP}

# Membuat Abstract Class Pada JAVA

Untuk membuat suatu class menjadi abstract, kita hanya perlu menambahkan kata kunci `<abstract>` pada pendefinisian class tersebut. Berikut adalah bentuk umum penulisannya :

```
<Access modifier> <abstract> class <nama Kelas> {
```

```
//Blok Program Dari Class Abstract
```

```
}
```

## ❑ Contoh :

```
public abstract class BangunDatar {
```

```
    //Block Kode dari Class
```

```
}
```

# Contoh Implementasi Abstract Class

Berikut adalah langkah – langkah membuat abstract class sederhana. Mari kita praktekan :

- ❑ **Langkah 1** : Buatlah sebuah class dengan nama BangunDatar, deklarasikan sebagai abstract

```
1  package tutorial1;  
2  
3  public abstract class BangunDatar {  
4  
5      //Block Kode dari Class  
6  
7  }  
8
```

# Contoh Implementasi Abstract Class

- ❑ **Langkah 2 :** Isilah Class BangunDatar dengan atribut dan method berikut

```
3  public abstract class BangunDatar {
4
5      private String namaBentuk;
6
7      public String getNamaBentuk() {
8          return namaBentuk;
9      }
10
11     public void setNamaBentuk(String namaBentuk) {
12         this.namaBentuk = namaBentuk;
13     }
14
15     public double hitungKeliling() {
16         return 0;
17     }
18
19     public double hitungLuas() {
20         return 0;
21     }
22 }
```

# Contoh Implementasi Abstract Class

- ❑ **Langkah 3** : Mari kita coba membuat object dari class BangunDatar. **Apa yang terjadi ??**

```
1  package tutorial1;  
2  
3  public class Main {  
4      BangunDatar is abstract; cannot be instantiated  
5      ----  
6      (Alt-Enter shows hints) (String[] args) {  
7          dari Class BangunDatar  
8          BangunDatar bentuk1 = new BangunDatar(); //TERNYATA PERINTAH INI ERROR  
9      }  
10  
11 }
```

**\*Disini dapat kita lihat bahwa konsep yang mengatakan jika class abstract itu tidak bisa digunakan untuk membuat/instansiasi object terbukti benar**

**SOLUSINYA ?? Kita turunkan class abstract ke kelas lainnya**



# Contoh Implementasi Abstract Class

- ❑ **Langkah 4** : Buatlah class baru dengan nama **PersegiPanjang** sebagai **turunan** dari Class BangunDatar.

```
3  public class PersegiPanjang extends BangunDatar {  
4      private int panjang;  
5      private int lebar;  
6  
7      public PersegiPanjang() {  
8          super.setNamaBentuk("Persegi Panjang");  
9          System.out.println("Nama Objek Adalah "+super.getNamaBentuk());  
10     }  
11  
12     public int getPanjang() { ...3 lines }  
13  
14  
15     public void setPanjang(int panjang) { ...3 lines }  
16  
17  
18  
19     public int getLebar() { ...3 lines }  
20  
21  
22  
23     public void setLebar(int lebar) { ...3 lines }  
24  
25  
26  
27  
28  
29 }  
30
```

# Contoh Implementasi Abstract Class

- ❑ **Langkah 5** : buatlah object dari Class **PersegiPanjang** nya di kelas Tester/Main

```
1  package tutorial1;  
2  
3  public class Main {  
4  
5      public static void main(String[] args) {  
6          //Instansiasi Object dari Class BangunDatar  
7          PersegiPanjang bentuk1 = new PersegiPanjang();  
8  
9      }  
10  
11 }
```

Output - Tutorial1 (run) ×

```
run:  
Nama Objek Adalah Persegi Panjang  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Abstract Method

- ❑ Saat kita membuat abstract class, Kita bisa membuat sebuah **method khusus** didalam kelas tersebut yang disebut sebagai ***abstract method***
- ❑ **Abstrak method** adalah method yang tidak memiliki **block/badan method** (kosong). Hanya berupa **nama method & parameternya saja**.
- ❑ abstract method wajib **dioverride** -> dituliskan kembali **di kelas turunannya**.
- ❑ abstract method tidak boleh memiliki access modifier ***private***

{OOP}

# Abstract Method

## Abstract methods

Besides classes, you can mark *methods* abstract, too. An abstract class means the class must be *extended*; an abstract method means the method must be *overridden*. You might decide that some (or all) behaviors in an abstract class don't make any sense unless they're implemented by a more specific subclass. In other words, you can't think of any generic method implementation that could possibly be useful for subclasses. What would a generic *eat()* method look like?

### An abstract method has no body!

Because you've already decided there isn't any code that would make sense in the abstract method, you won't put in a method body. So no curly braces—just end the declaration with a semicolon.

```
public abstract void eat();
```

No method body!  
End it with a semicolon.

**If you declare an abstract *method*, you MUST mark the *class* abstract as well. You can't have an abstract method in a non-abstract class.**

If you put even a single abstract method in a class, you have to make the class abstract. But you *can* mix both abstract and non-abstract methods in the abstract class.



# {OOP}

# Membuat Abstract Method

- ❑ Contoh membuat method abstract pada JAVA :

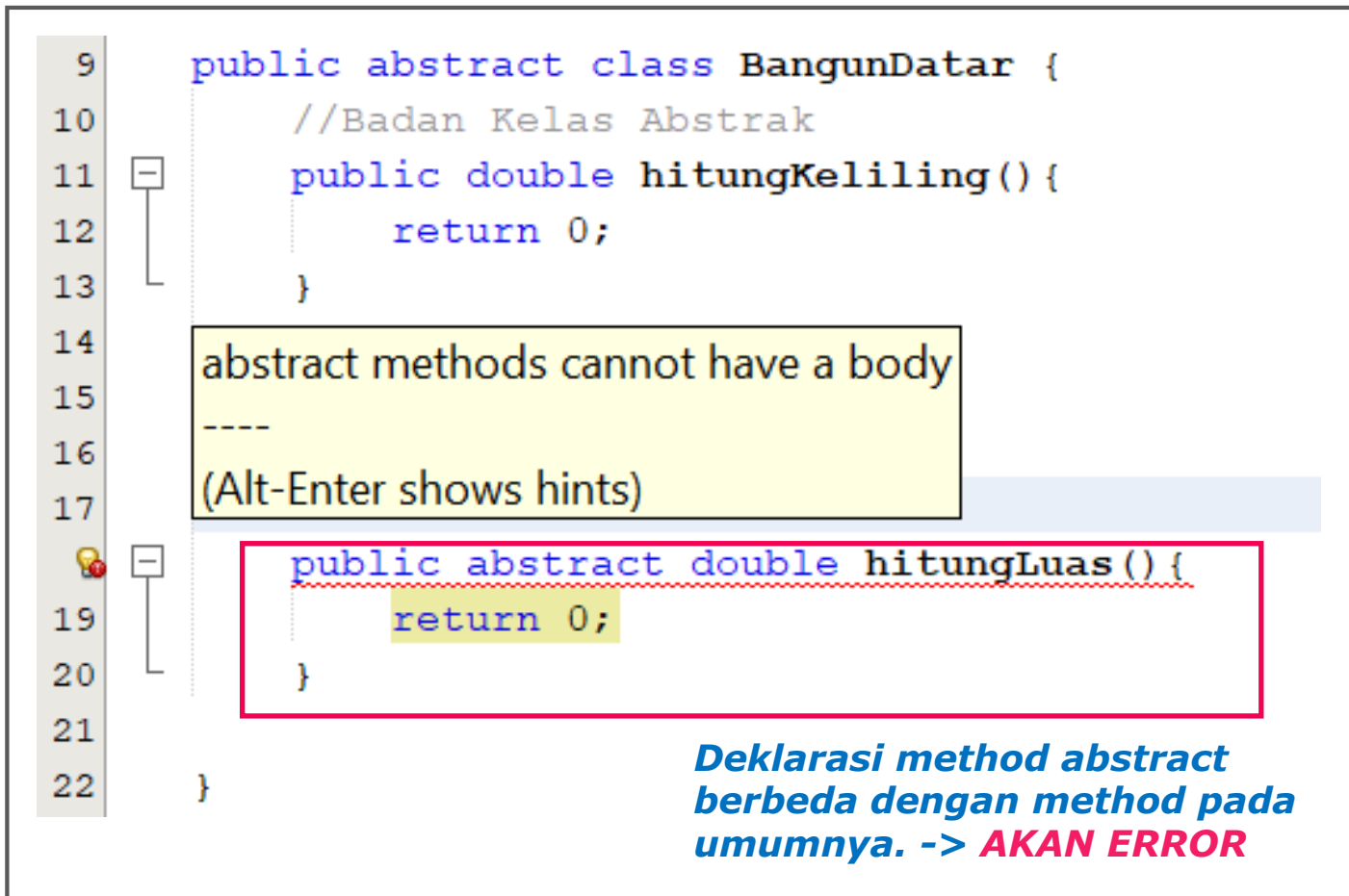
```
1 public abstract class BangunDatar {  
4  
5     private String namaBentuk;  
6  
7     public String getNamaBentuk() { ...3 lines }  
10  
11     public void setNamaBentuk(String namaBentuk)  
14  
15     public abstract double hitungKeliling();  
16  
17     public double hitungLuas() {  
18         return 0;  
19     }  
20 }
```

Untuk membuat **method** menjadi abstract, tambahkan **kata kunci abstract**

{OOP}

# Membuat Abstract Method

- ❑ Jika kita mendeklarasikan method sebagai abstract namun method tersebut mempunyai block method, **maka program akan error**



```
9 public abstract class BangunDatar {
10     //Badan Kelas Abstrak
11     public double hitungKeliling() {
12         return 0;
13     }
14
15     public abstract double hitungLuas() {
16         return 0;
17     }
18 }
19
20
21
22
```

abstract methods cannot have a body  
----  
(Alt-Enter shows hints)

**Deklarasi method abstract berbeda dengan method pada umumnya. -> **AKAN ERROR****

{OOP}



# Membuat Abstract Method

- ❑ Jika terdapat **abstract method** didalam **abstract class**, maka ia WAJIB **dideklarasikan kembali (override)** pada seluruh kelas turunannya (child class).

```
28      @Override
30      public double hitungKeliling() {
31          int keliling;
32          keliling = (2*this.panjang) + (2* this.lebar);
33          return keliling;
34      }
35
36      @Override
37      public double hitungLuas() {
38          int luas;
39          luas = this.panjang * this.lebar;
40          return luas;
41      }
```

*Karena kita sudah tau bahwa object bangun datar berbentuk persegi panjang maka kita bisa tuliskan rumus luas & keliling didalam method*

{OOP}

- ❑ Lengkapi kode pada class PersegiPanjang seperti contoh berikut, kemudian panggil objectnya **di kelas Main**

```
3 public class PersegiPanjang extends BangunDatar {
4     private int panjang;
5     private int lebar;
6
7     //Konstruktor dari Class Persegi Panjang
8     public PersegiPanjang(int panjang, int lebar){
9         this.panjang = panjang;
10        this.lebar = lebar;
11        super.setNamaBentuk("Persegi Panjang");
12        System.out.println("Nama Objek Adalah "+super.getNamaBentuk());
13    }
14
15    public int getPanjang() {...3 lines }
18    public void setPanjang(int panjang) {...3 lines }
21    public int getLebar() {...3 lines }
24    public void setLebar(int lebar) {...3 lines }
27
28    @Override
29    public double hitungKeliling() {
30        int keliling;
31        keliling = (2*this.panjang) + (2* this.lebar);
32        return keliling;
33    }
34
35    @Override
36    public double hitungLuas() {
37        int luas;
38        luas = this.panjang * this.lebar;
39        return luas;
40    }
41 }
```

{OOP}

- ❑ Lengkapi kode pada class PersegiPanjang seperti contoh berikut, kemudian panggil objectnya **di kelas Main**

```
1 package tutorial1;  
2  
3 public class Main {  
4  
5     public static void main(String[] args) {  
6         //Instansiasi Object dari Class BangunDatar  
7         BangunDatar bentuk1 = new PersegiPanjang(10,5);  
8         System.out.println("Keliling Bangun Datar    = "+bentuk1.hitungKeliling()+" cm" );  
9         System.out.println("Luas Bangun Datar       = "+bentuk1.hitungLuas()+" cm");  
10    }  
11 }  
12 }
```

Output - Tutorial1 (run) x

```
run:  
Nama Objek Adalah Persegi Panjang  
Keliling Bangun Datar    = 30.0 cm  
Luas Bangun Datar       = 50.0 cm  
BUILD SUCCESSFUL (total time: 0 seconds)
```

{OOP}

**Lanjutkan program tersebut agar dapat menampilkan hasil perhitungan luas untuk objek *Lingkaran* dan *Segitiga***

# Bagian 2:

## *Interface*

# Definisi Interface (PBO)

- ❑ Sebelumnya kita sudah tahu bahwa abstract class bisa kita gunakan sebagai **kontrak untuk class turunannya**. Namun masih ada cara lain yang dapat kita gunakan yaitu dengan **interface**
- ❑ Interface disini bukanlah membuat **User Interface (jangan salah sangka)**.
- ❑ **Interface** merupakan sebuah **block program** yang hanya berisi **sekumpulan method abstract** untuk diimplementasikan dikelas yang lain

{OOP}

# Aturan Pada Interface

- ❑ Semua method yang ditulis didalam interface harus merupakan **method abstract (tidak memiliki block)**.
- ❑ **Semua method** yang ada interface **wajib diimplementasikan** didalam kelas yang menggunakan interface tersebut **(Override)**.
- ❑ Di Interface kita **tidak boleh** memiliki **field atribut**, namun boleh memiliki **field constant** (atribut yang tidak bisa diubah/Konstanta)
- ❑ Untuk mewariskan **interface**, tidak menggunakan kata kunci **extends**, melainkan **implements**

{OOP}



# Membuat Interface Pada JAVA

Membuat interface, caranya sedikit berbeda dengan ketika membuat class karena disini kita deklarasikan dengan kata kunci interface . Berikut adalah bentuk umum pembuatan interface

```
<Access modifier> <interface> <Nama Interface> {  
  
    //Block Kode Interface  
  
}
```

## ❑ Contoh :

```
3 //Contoh membuat Interface dengan nama Button  
4 public interface Button {  
5  
6     //Block Kode Interface  
7  
8 }
```

# Contoh Implementasi Interface

Berikut adalah langkah – langkah sederhana menggunakan interface. Mari kita praktekan :

- ❑ **Langkah 1** : Buatlah sebuah interface dengan nama **Button**, lalu isilah **2 buah method abstract** seperti contoh berikut :

```
1  package tutorial2;
2
3  //Contoh membuat Interface dengan nama Button
4  public interface Button {
5
6      public void createButton();
7      public void viewButton();
8
9  }
10
```

# Contoh Implementasi Interface

- ❑ **Langkah 2 :** Buatlah class baru dengan nama **buttonSimpan** , selanjutnya implementasikan interface button pada Class tersebut

```
1 package tutorial2;  
2  
3  
4 public class buttonSimpan implements Button {  
5     buttonSimpan is not abstract and does not override abstract method isClicked() in Button  
6     Missing javadoc.  
7  
8     Create Test Class  
9  
10    Implement unimplemented abstract methods of tutorial2.Button  
11  
12    Create Subclass  
13    ----  
14    (Alt-Enter shows hints)
```

Disini dapat dilihat bahwa ketika kita **mengimplementasikan interface** pada suatu class. Maka class tersebut akan **dipaksa untuk mengoveride** seluruh isi didalam interface. Jika tidak **PROGRAM ERROR**

# Contoh Implementasi Interface

- ❑ **Langkah 3** : lakukan **Overriding** pada seluruh method abstract didalam interface. Lalu tulislah kode didalam method tersebut sesuai kebutuhan dari kelas buttonSimpan.

```
3 public class ButtonSimpan implements Button {
4
5     @Override
6     public void createButton() {
7
8     }
9
10    @Override
11    public void viewButton() {
12
13    }
14
15 }
```

Seluruh method pada Button , harus ada pada class buttonSimpan. Inilah mengapa interface dikatakan sebagai **kontrak suatu class**

# Contoh Implementasi Interface

- ❑ **Langkah 4 :** Lengkapilah isi dari class buttonSimpan sesuai kebutuhan objectnya

```
3  public class ButtonSimpan implements Button, Event {
4
5      private int sumbuX;
6      private int sumbuY;
7      private String namaButton;
8      private String warna;
9
10     public ButtonSimpan() { ...3 lines }
13
14     public int getSumbuX() { ...3 lines }
17     public void setSumbuX(int sumbuX) { ...3 lines }
20     public int getSumbuY() { ...3 lines }
23     public void setSumbuY(int sumbuY) { ...3 lines }
26     public String getNamaButton() { ...3 lines }
29     public void setNamaButton(String namaButton) { ...3 lines }
32     public String getWarna() { ...3 lines }
35     public void setWarna(String warna) { ...3 lines }
38
```

# Contoh Implementasi Interface

- ❑ **Langkah 4 :** Lengkapilah isi dari class buttonSimpan sesuai kebutuhan objectnya

```
39
40     @Override
41     public void viewButton(){
42         System.out.println("Tampil Info " +this.namaButton);
43         System.out.println("Sumbu X = " +this.getSumbuX()+" pixel");
44         System.out.println("Sumbu y = " +this.getSumbuY()+" pixel");
45         System.out.println("Warna    = " +this.warna);
46     }
47
48     //Method override dari Interface Button
49     @Override
50     public void createButton() {
51         this.setSumbuX(50);
52         this.setSumbuY(20);
53         this.setWarna("Biru");
54         this.setNamaButton("Button Simpan");
55     }
56
```



# Contoh Implementasi Interface

- ❑ **Langkah 5 :** Buatlah object pada class Main untuk mengimplementasikan class ButtonSimpan

```
1  package tutorial2;
2
3  public class Main {
4
5      public static void main(String[] args) {
6
7          Button btnSimpan = new ButtonSimpan();
8
9          btnSimpan.viewButton();
10     }
11 }
12
```

Output - Tutorial2 (run) x

```
run:
Tampil Info Button Simpan
Sumbu X = 50 pixel
Sumbu y = 20 pixel
Warna    = Biru
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Multiple Interface

- ❑ DI JAVA, sebuah kelas hanya boleh menjadi turunan dari **1 parent/Superclass**. Jadi Tidak ada yang namanya konsep **multiple-Inheritance** di JAVA.
- ❑ Tapi JAVA **memperbolehkan** sebuah kelas untuk dapat mengimplementasikan **lebih dari satu interface**. (**Pseudo Multiple Inheritance**)
- ❑ Tujuan dari multiple interface adalah memudahkan kita dalam merancang class apabila ada **kesamaan perilaku** dari **2 atau lebih interface**

{OOP}

# Membuat Multiple Interface pada JAVA

Membuat implementasi lebih dari 1 interface caranya sama dengan sebelumnya . Hanya saja kita gunakan symbol , (**Koma**) sebagai pemisah antar interface yang diimplementasikan

```
<Access modifier> <interface> Interface ke-1, Interface ke-2, ke-n {
```

```
    //Block Kode Interface
```

```
}
```

[dipisahkan dengan  
tanda koma (,)]

❏ Contoh :

```
public class ButtonSimpan implements Button, Event {
```

```
}
```

# Contoh Implementasi Multiple Interface

- ❑ **Langkah 1** : Masih pada project yang sama dengan Sebelumnya, silakan tambah sebuah interface dengan nama Event dan sebuah **method abstract** seperti contoh berikut

```
1 package tutorial2;
2
3 public interface Event {
4
5     public void isClicked();
6
7 }
8
```

# Contoh Implementasi Multiple Interface

- ❑ **Langkah 2 :** Implementasikan interface pada langkah ke-1 di class ButtonSimpan. Selanjutnya lakukan overriding pada method isClicked()

```
public class ButtonSimpan implements Button, Event {  
  
    private int sumbuX;  
    private int sumbuY;  
    private String namaButton;  
    private String warna;  
  
    public ButtonSimpan() {  
        this.createButton();  
    }  
  
    ...  
  
    @Override  
    public void isClicked() {  
        System.out.println("=====");  
        System.out.println("Button Ini akan Menyimpan Data didalam FORM ketika Diclick User");  
        System.out.println("Method ini akan berisi Kode untuk proses menyimpan Data");  
    }  
}
```

# Contoh Implementasi Multiple Interface

- ❑ **Langkah 3 :** Impementasikan kode tersebut didalam kelas Main (gunakan kode dari contoh sebelumnya)

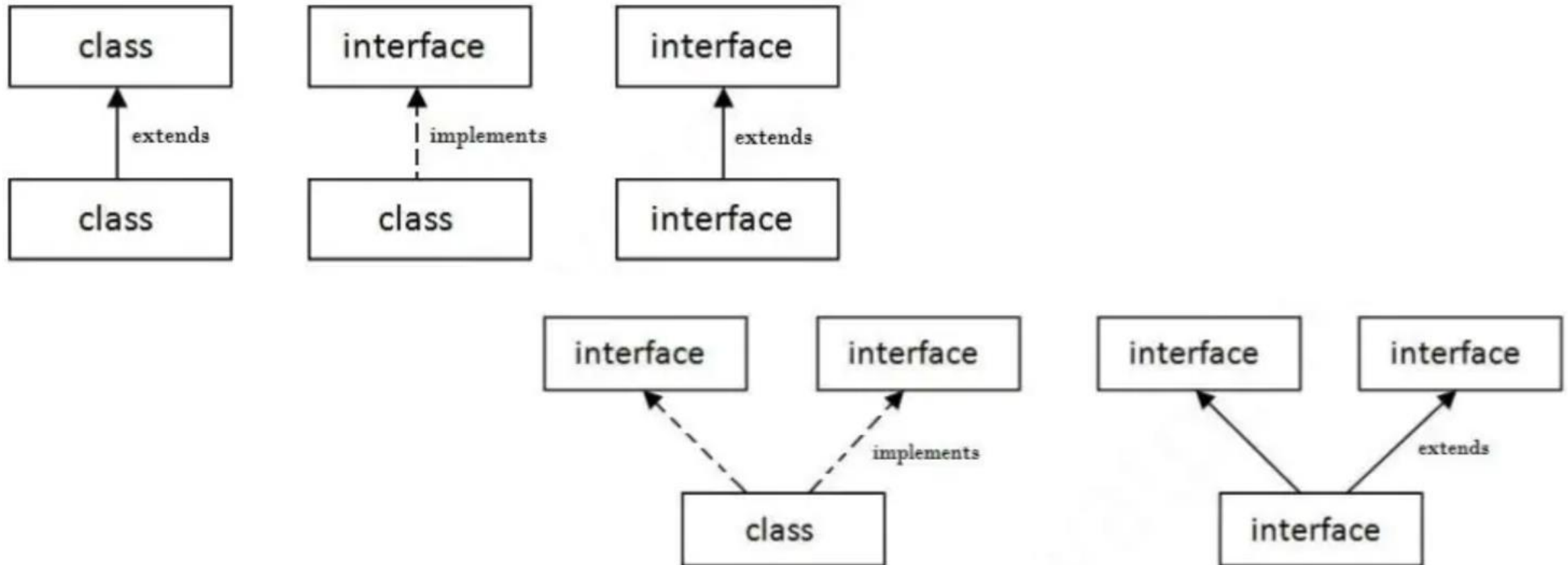
```
3 public class Main {
4
5     public static void main(String[] args) {
6
7         Button btnSimpan = new ButtonSimpan();
8         Event event = new ButtonSimpan();
9
10        btnSimpan.viewButton();
11        event.isClicked();
12    }
13 }
14
15
```

Output - Tutorial2 (run) x

```
run:
Tampil Info Button Simpan
Sumbu X = 50 pixel
Sumbu y = 20 pixel
Warna    = Biru
=====
Button Ini akan Menyimpan Data didalam FORM ketika Diclick User
Method ini akan berisi Kode untuk proses menyimpan Data
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Pewarisan Pada Interface

- ❑ Beberapa bentuk hubungan (**extends & implements**) yang diijinkan pada JAVA



# Kelas Abstract VS Interface

- ❑ Berikut adalah Tabel Perbandingan antara Kelas Abstrak dan Interface :

Kategori	Kelas Abstrak	Interface
Kata Kunci	extends	implements
Member/Anggota	Atribut, Method (Concrete), & Min 1 Abstract Method	Abstract Method
Bentuk pewarisan	Hanya mendukung Single Inheritance (extends)	Mendukung Multiple Interface (implement)

- ❑ Untuk Kapan **pemakaian Kelas Abstracts & Interface didalam program** bergantung dengan kebutuhan masalah didalam pemrograman yang kita selesaikan (Dinamis)



Setelah memahami **Materi Hari ini** pada JAVA,  
mari kita berlatih

**\*Silakan buka modul praktikum pertemuan 9&10**

# Terima Kasih