Pemrograman Berorientasi Obyek

Abstract & Interface

anton@ukdw.ac.id

Latar Belakang

- Kita sudah mengenal pewarisan, bahwa class anak akan selalu mendapat warisan atribut dan behavior dari class induk
- Class anak dapat mengoverridenya (tapi tidak wajib)
- Kadang kala dibutuhkan class yang sebagian methodnya WAJIB dioverride di class anak!

Abstrak

- Abstrak = tidak memiliki implementasi
- Class abstract berarti class tersebut terdapat method yg tidak memiliki implementasi (kode program), hanya judul method dan parameternya

Abstract class

abstract class <NamaClass> {

    <variabel class>

    <konstruktor>

    <method biasa>

    <method abstract>

}

Contoh method dgn implementasi

```
public void showData(int n) {
    for(int i=0; i<n; i++) {
        System.out.println("Baris : " + i);
    }
}
```

Contoh method abstract

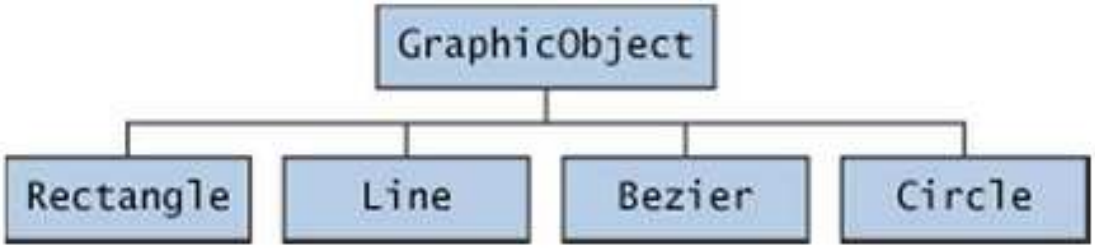
```
public void showData(int n);
```

```
public abstract void showData(int n);
```

Abstract Class

- Terdiri dari **satu atau lebih abstract method**
- Bisa memiliki **static** method
- Bisa memiliki **static** field
- Bisa memiliki **variabel** class biasa
- Abstract method **harus** dibuat **implementasinya** oleh **class turunannya**
- Abstract class **tidak dapat dibuat instancenya**

Contoh abstract class




```
abstract class GraphicObject {
   int x, y;
   ...
   void moveTo(int newX, int newY) {
      ...
   }
   abstract void draw();
   abstract void resize();
}
```

Abstract Class

- Misalkan method **bersuara()** pada class **Binatang**
- Method **bersuara()** seharusnya **dioverride** karena tidak memiliki bentuk di class Binatang (-> tergantung jenis binatangnya)
- Untuk memaksa method **bersuara() harus** / **wajib** dioverride, maka method tersebut perlu dibuat menjadi **abstract**.
- Sedangkan class nya dibuat **abstract class**

Abstract Class

- Penulisan **abstract method** adalah sebagai berikut:

```
public abstract String bersuara();
```

- Untuk selanjutnya, method ini disebut sebagai **abstract method** atau **prototype method**
- Dapat kita lihat bahwa method bersuara:
  - tidak memiliki **'tubuh'/implementasi** method.
  - menggunakan keyword **abstract**.

Abstract Class

- Jika kita **paksa** untuk memiliki kode implementasi program seperti ini:

```
public abstract String bersuara()
{
        System.out.println("weee…");
}
```

- Maka akan terjadi error:

```
abstract methods cannot have a body
```

Abstract Class

- Apabila **abstract** method **bersuara()** tadi kita tambahkan ke **class Binatang** yang sudah kita buat, maka akan muncul **error**:

  `Binatang is not abstract and does not override abstract method bersuara() in binatang`

- Pemecahannya adalah dengan mengubah **class Binatang** menjadi sebuah **abstract class**.

  (Bila ada abstract methods dalam satu class, maka class tersebut harus abstract juga)

Abstract Class

- Abstract class ini **tidak bisa diinstansiasi**

  (karena memiliki method yang abstract)
- **Questions:**
  - Apakah dalam sebuah abstract class boleh tak memiliki abstract methods **sama sekali**?
    - Boleh
  - Bagaimana jika sebuah abstract class memiliki **constructor**? Bagaimana fungsi dan pemakaian constructor tersebut?
    - Diletakkan pada class anak

Constructor of Abstract Class

```
public abstract class TestEngine
{
   private String engineId;
   private String engineName;

   public TestEngine(String engineId , String engineName)
   {
     this.engineId = engineId;
     this.engineName = engineName;
   }
   //public gettors and settors
   public abstract void scheduleTest();
}


public class JavaTestEngine extends TestEngine
{

   private String typeName;

   public JavaTestEngine(String engineId , String engineName , String typeName)
   {
      super(engineId , engineName);
      this.typeName = typeName;
   }

   public void scheduleTest()
   {
     //do Stuff
   }
}
```

Setiap class dalam Java pasti memiliki **konstruktor** secara default!

Contoh yang salah

```
public abstract class KelasAbstrak {
    public KelasAbstrak() {
        tulis("Aku berada di "+this.getClass());
    }
    public void tulis(String teks)
    {
        System.out.println(teks);
    }
}
```

Pada class abstract tidak ada method abstrak!

```
public class ImplementasiAbstrak extends KelasAbstrak {
    public ImplementasiAbstrak()
    {
        super.tulis("Aku ada di kelas ImplementasiAbstrak");
    }
}
```

Contoh yang benar

```
public abstract class KelasAbstrak {
    public KelasAbstrak() {
       tulis("Aku berada di "+this.getClass());
    }
    public void tulis(String teks)
    {
        System.out.println(teks);
    }

    abstract void methodAbstrak();
}
```

Pada class abstract sebaiknya ada method abstrak!

```
public class ImplementasiAbstrak extends KelasAbstrak {
    public ImplementasiAbstrak()
    {
        super.tulis("Aku ada di kelas ImplementasiAbstrak");
    }
    public void methodAbstrak()
    {
        super.tulis("Method abstrak di kelas induk sudah aku definisikan ulang");
    }
}
```

Contoh

```
abstract class AAA
{
   //Fungsi abstrak "callme()" tanpa implementasi
   abstract void callme();

   //Fungsi dengan implementasi tetap dibolehkan
   //dalam kelas abstrak
   public void callmetoo()
   {
      System.out.println("Ini fungsi test");
   }
}

class BBB extends AAA
{
   //Implementasi dari fungsi callme() pada kelas abstract AAA
   void callme()
   {
      System.out.println("Implementasi fungsi callme di kelas BBB");
   }
}

class abstractdemo
{
   public static void main(String args[])
   {
      BBB varBBB = new BBB();
      varBBB.callme();
      varBBB.callmetoo();
   }
}
```

Interface

- Kumpulan **konstanta** dan method yang tidak memiliki implementasi
- Interface seperti kontrak dengan ikatan implements
- Suatu class dapat meng-implementasikan lebih dari satu interface
- Class yang mengimplementasikan suatu interface harus membuat **implementasi** dari method-method yang ada pada interface
- Seluruh method pada interface secara otomatis bersifat **public**, method pada class yang mengimplementasikan interface harus diset ke **public**

Jika class implementasi tidak diset ke public

```
D:\ContohAbstrak.java:30: error: aaa() in X cannot implement aaa() in Coba
    void aaa(){
         ^
  attempting to assign weaker access privileges; was public
```

Contoh interface

```
public interface Shape {

    double volume();

    double area();

}
```

Interface bernama Shape

Method yang tidak berisi Implementasi

Seluruhnya bersifat public abstract
(tidak dapat diset private, protected atau *default*)

Contoh interface

```
public class Persegi implements Shape {
        private int sisi;

        public double volume() {
                return 0;
        }

        public double area() {
                return sisi * sisi;
        }
}
```

Contoh Interface

```
public interface mp3Player {
    public static final int STATUS; //final dan static
    List TRACKLIST;
    void playTrack();
    void stopTrack();
    void volumeUp();
    void volumeDown();
}
```

Interface

- Interface adalah sebuah **blok yang berisi deklarasi metode** untuk diimplementasikan di class lain.
  - *Tidak ada tubuh method*
- Berarti **semua** metode dalam interface adalah **abstract**.
- Tapi pada pendeklarasiannya **tidak perlu** menggunakan keyword **abstract**.

Interface

- Fungsinya adalah membuat suatu class yang **bisa diimplementasikan oleh berbagai class** lain bahkan yang *tidak berelasi sama sekali*.
- Contoh:

```
public interface kamera
{
        public void setPixel(float pixel);
        public void ambilGambar();
}
```

Interface

- Suatu class yang mengimplementasi suatu interface, memiliki “kewajiban” untuk membuat implementasi method-method yang disediakan oleh interface tersebut
- Implements = contract
- Pada interface didefinisikan method **signature** yang harus diikuti pada class yang mengimplementasi interface tersebut

Interface

- Interface digunakan untuk mendukung **multiple inheritance** (satu class memiliki superclass lebih dari satu)
- Selain bisa mendeklarasikan method abstract, di dalam interface juga dapat diberikan attribute **final** (konstanta).
  - Tidak bisa jika berupa variabel class
- Konstanta ini juga **diwariskan** kepada class yang mengimplementasikan interface tersebut.

Mengapa Interface

- Mengapa dibutuhkan interface? Dalam bahasa pemrograman lain seperti C++, dikenal istilah **multiple-inheritance**,
  - artinya sebuah objek bisa diturunkan dari dua atau lebih objek berbeda. Misalnya, objek X memiliki super kelas A dan B.
- Pada Java, hal ini **tidak dimungkinkan**, karena objek hanya bisa extends dari 1 objek saja, dan disinilah **interface** Java menjadi solusinya

Interface

- Interface dapat menerima warisan dari interface lain (bisa lebih dari satu)
- Class bisa mengimplements lebih dari satu interface
- Class yang mengimplementasikan interface harus mengimplementasikan semua method interface
- Jika tidak, maka class tersebut harus dideklarasikan sebagai sebuah **abstract** class

Pewarisan Interface

```
public interface NamaInterface
extends InterfaceA, InterfaceB{
...
}
```

Extends dan implements Interface

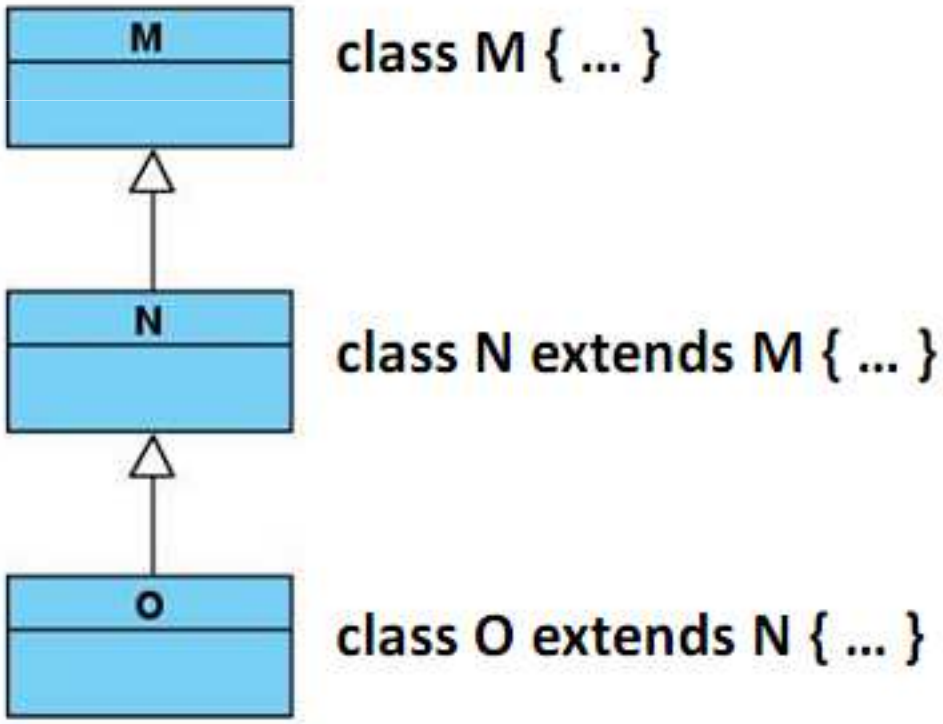
- **class NamaKelas [extends NamaKelasSuper]**
  implements NamaInterface1, NamaInterface2 {
  …
  }

Interface vs abstract

- Penggunaan **keyword extends** dan **implements** adalah salah satu perbedaan dari keduanya.
- **Semua** methods di interface bersifat abstract, sedangkan methods di abstract class **bisa tidak ada**.
- abstract class **bisa** mengimplementasikan interface, tapi interface **tak bisa** mewarisi abstract class.
- abstract dapat memiliki **variabel kelas**, interface hanya dpt berupa **konstanta**

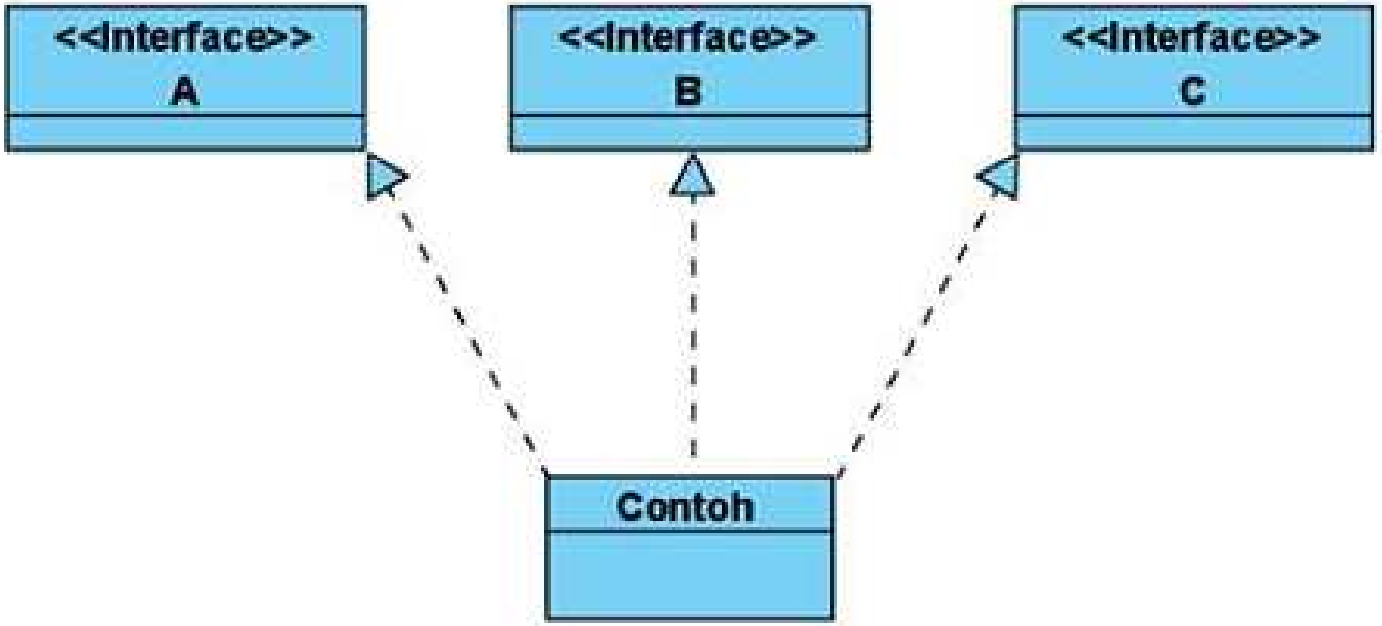
Extends vs Interface

- Suatu class hanya bisa meng-extends satu class lainnya



Extends vs Interface

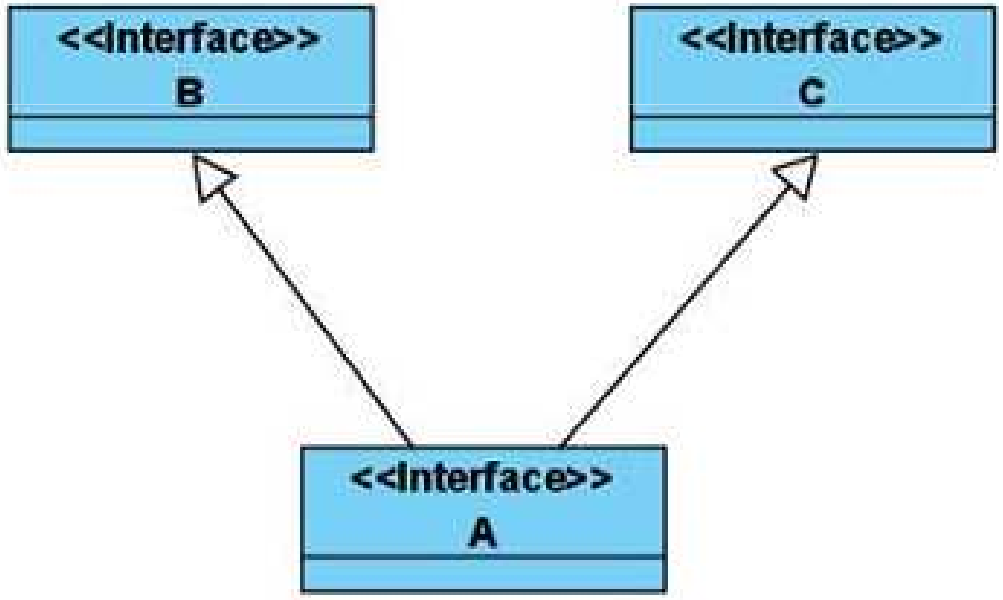
- Satu class bisa meng-implements lebih dari satu interface



class Contoh implements A, B, C { … }

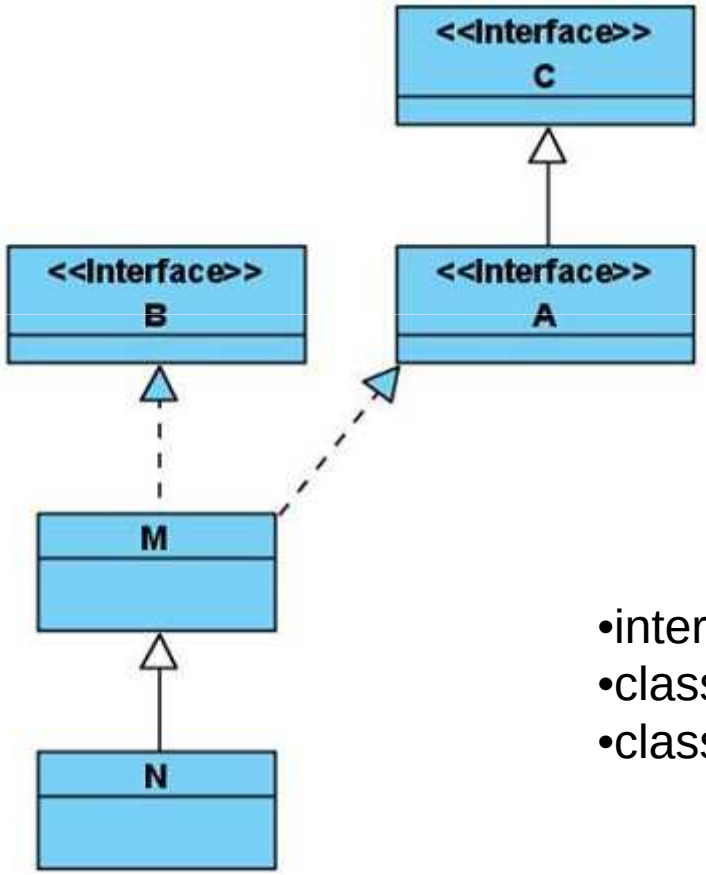
Extends vs Interface

- Suatu interface dapat meng-extends lebih dari satu interface lainnya



interface A extends B, C { ... }

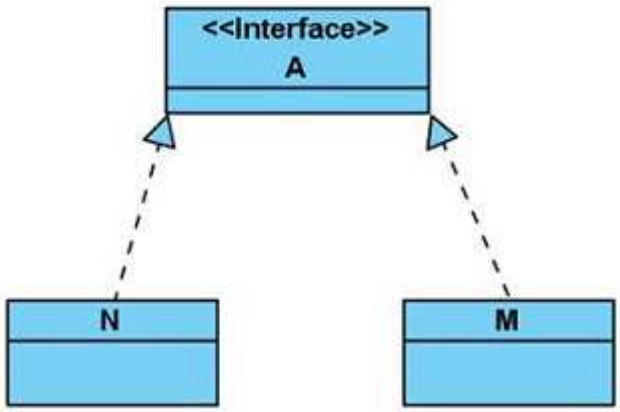
Contoh-contoh



- interface A extends C
- class M implements B, A
- class N extends M

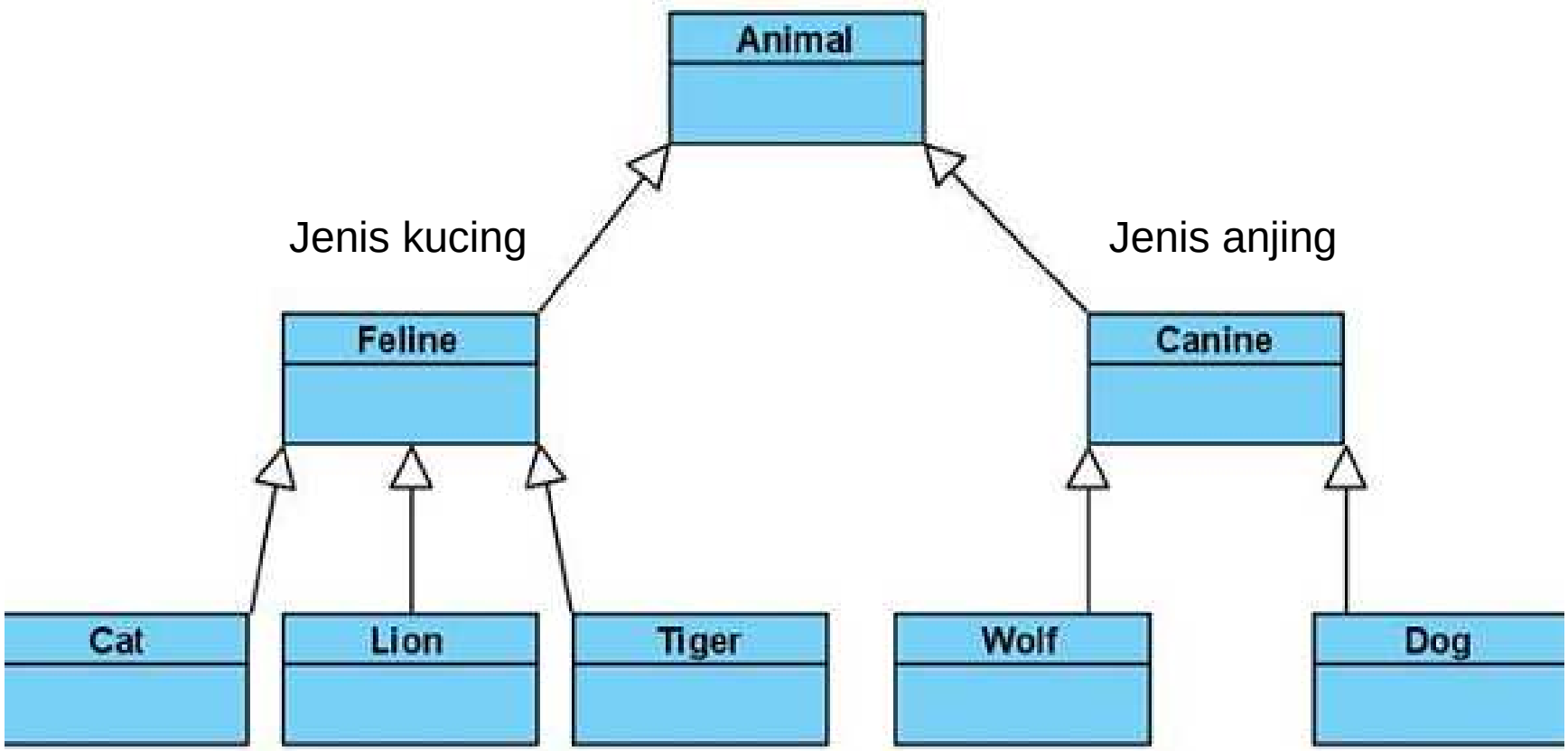
Konversi

- Konversi dapat dilakukan dari type class menjadi type interface dengan syarat class tersebut mengimplementasi interfacenya

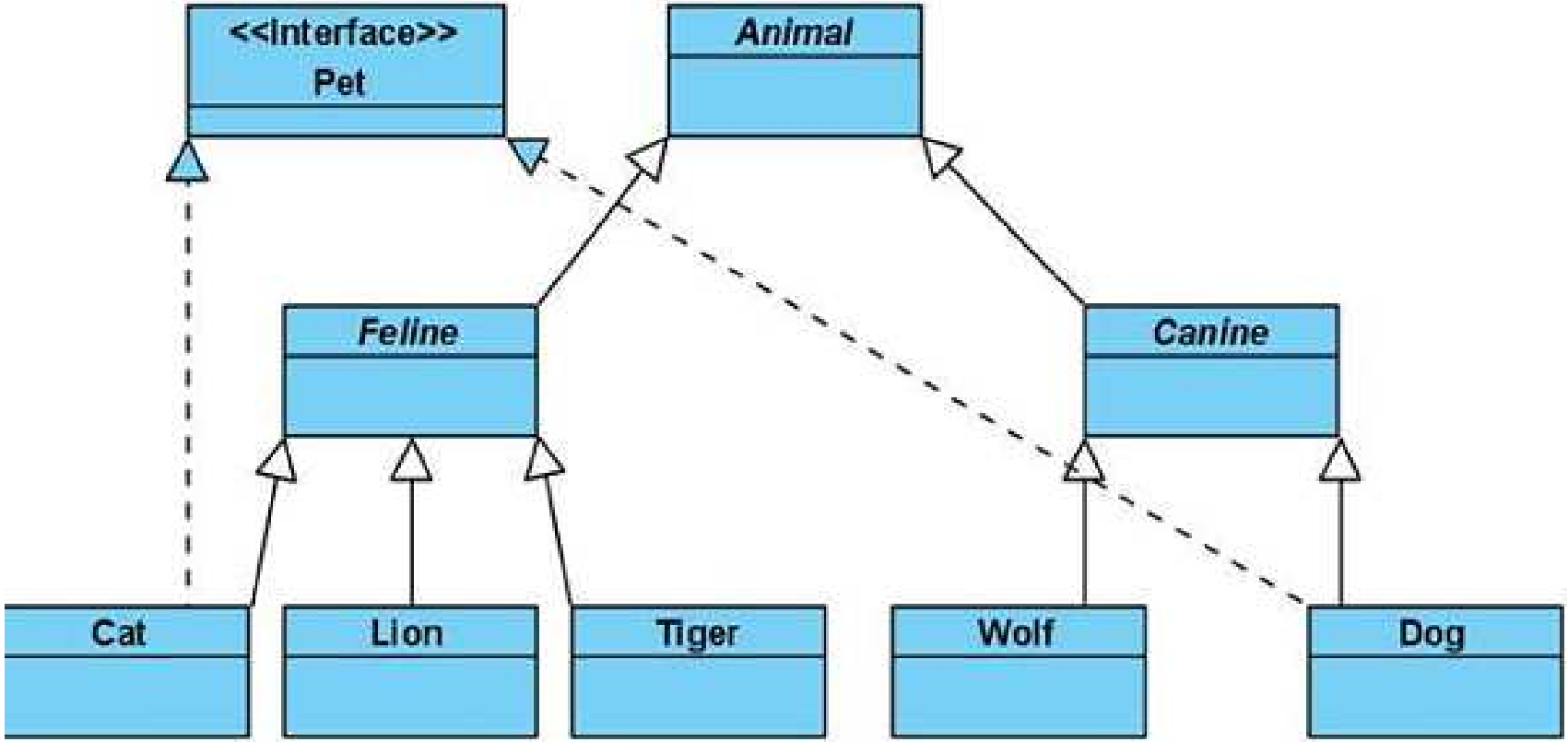



```
N object1 = new N();

A interface1 = object1


M object2 = new M();

A interface2 = object2
```

Contoh


Animal
Jenis kucing
Jenis anjing
Feline
Canine
Cat
Lion
Tiger
Wolf
Dog

Contoh interface



NEXT

- Tes Kecil I
- Polimorfisme