

Abstract class & Interface

Pertemuan 5



Topics

1. **Abstract Class**
2. **Interface**
3. **Multiple Inheritance**



(1) Abstract Class

- Abstract class merupakan suatu class yang tidak dapat diinstansiasikan.
- Umumnya digunakan dalam konsep **inheritance**
- Sebagai contoh :
 - Animal : Cat, Dog, Cow..
 - Object 'animal' itu seperti apa ?
 - Person : Manager, Employee, Executive..
 - 'Person' itu konkritnya bagaimana ?
- Abstract class merupakan suatu solusi agar kelas (superclass) **tidak diinstasiasi**.



Karakteristik abstract class

- Mengumpulkan semua **properties** umum dari kelas turunannya
- Boleh memiliki data member (instance variables)
- Abstract class dapat menyediakan :
 - **Implemented method** : mengimplementasikan method secara konkrit (ada body method-nya)
 - **Abstract method** : method bersifat abstrak (tidak ada body method), dan harus diimplementasikan oleh kelas turunannya.

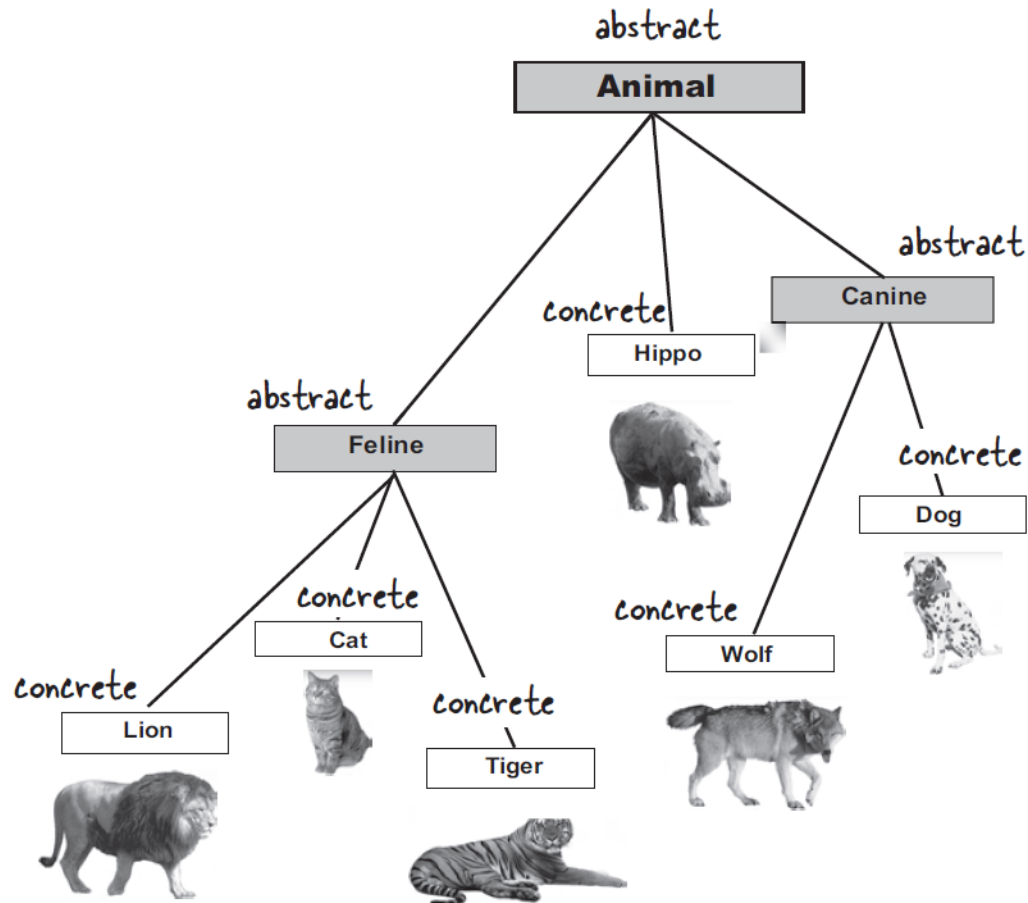


Contd..

- Must be defined by using the **abstract** keyword
- **Cannot** be instantiated into **objects**
- Can have **local** declared **variables**
- Can have **method definitions and implementations**
- Must be subclassed (**extends**) rather than implemented (**implements**)



Contoh kasus :



Contoh abstract class

- `public abstract class Animal {
 abstract int eat();
 abstract void breathe();
}`
- Kelas ini tidak dapat diinstansiasikan
- Semua kelas turunan (subclass) dari Animal harus mengimplementasikan method `eat()` dan `breathe()`



Contoh implementasi abstract class

```
abstract class Shape {  
    protected int x, y;  
    Shape(int _x, int _y) {  
        x = _x;  
        y = _y;  
    }  
}
```

Abstract class: objects
cannot be instantiated

```
Shape s1 = new Circle();  
Shape s = new Shape(10, 10) // compile error
```

```
class Circle extends Shape {  
    private int r;  
    public Circle(int _x, int _y, int _r) {  
        super(_x, _y);  
        r = _r;  
    }  
    ...  
}
```

Concrete class: objects
can be instantiated

Abstract Method

- Terkadang kita ingin mendefinisikan suatu method pada superclass yang perlu diimplementasikan pada kelas turunannya (subclass)
 - Contoh : **Animal.makeASound()** ?
- Solusinya : **abstract method**
- Abstract method hanya berisi deklarasi tanpa body method nya :
 - **abstract void makeASound();**
- Suatu kelas yang memiliki abstract method harus dideklarasikan sebagai abstract class
- Suatu abstract method harus diimplementasikan oleh kelas turunannya (subclass) agar dapat diinstansiasikan



Contd...

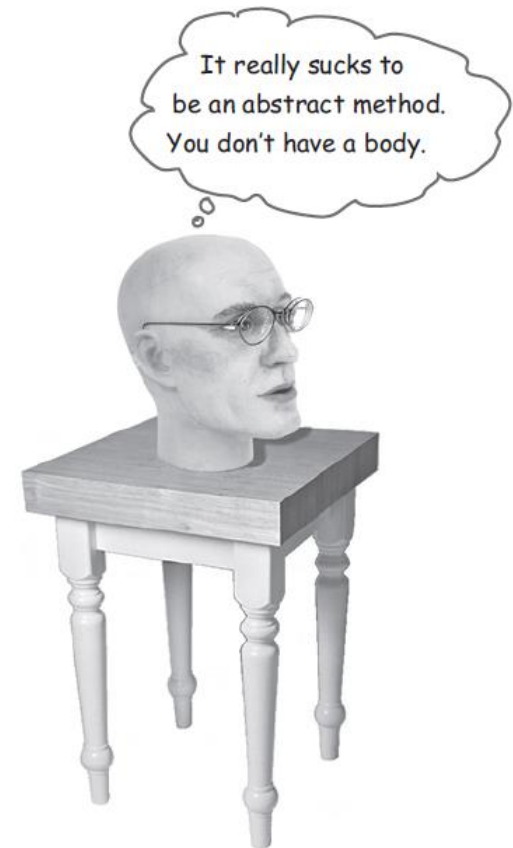

- An abstract class means the class must be **extended**. An abstract method means the method must be **overridden**.
- Abstract method has no body. If you declare an abstract method, you **MUST** mark the **class abstract** as well. You can't have an abstract method in a non-abstract class.
- You **MUST** implement **all abstract method**.



Contd..

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

No method body!
End it with a semicolon.



Contoh implementasi abstract method :

```
abstract class Shape {  
    protected int x, y;  
    Shape(int _x, int _y) {  
        x = _x;  
        y = _y;  
    }  
    abstract public void draw();  
    abstract public void erase();  
    public void moveTo(int _x, int _y) {  
        erase();  
        x = _x;  
        y = _y;  
        draw();  
    }  
}
```

```
class Circle extends Shape {  
    private int r;  
    public Circle(int _x, int _y, int _r) {  
        super(_x, _y);  
        r = _r;  
        draw();  
    }  
    public void draw() {  
        System.out.println("Draw circle at (" + x + ", " + y + ")");  
    }  
    public void erase() {  
        System.out.println("Erase circle at (" + x + ", " + y + ")");  
    }  
}
```

Diskusi

1. Apa saja yang membedakan antara abstract class dengan concrete class ?
2. Jelaskan apa pertimbangan dalam menggunakan abstract method ?
3. Jelaskan mengapa abstract class dapat mendukung konsep inheritance ?



(2) Interfaces

- Java does not support multiple inheritance
 - What if we want an object to be multiple things ?
- Specify the form (specification, behavior) of something (a concept)
 - Not providing an implementation
 - Interface's methods define the contract / protocol (signature) to be respected for this concept
- Based on the “has-a” relationship
- Useful when you want to use code written by others



Contd..

- A special type of class – a “pure” abstract class :
 - No data (only static or final)
 - Defines a set of abstract methods (prototypes)
 - Doesn't provide the implementation for the prototypes, only the definition (signature)
- A class can implement any number of interfaces
 - It will respect a “contract”: implement all its methods
 - NB : implementation \leftrightarrow inheritance
- An Interface does not have a constructor method



Contd..

- It is implemented by a class (using the keyword **implements**) or extended by another interface.
- All methods in a Java interface are **abstract**.



Why Use Interface ?

- When implementing a class from an interface we force it to implement all the abstract methods.
 - The interface **forces separation** of **what a class can do**, to **how it actually does it**.
 - A programmer can change how something is done at any point, without changing the function of the class
 - This facilitates the idea of polymorphism as the method described in the interface will be implemented by all classes that implement the interface.



What an interface can do

- While a class can only inherit from a single superclass

```
public class ClassName extends Superclass {  
    //class implementation  
} //end class ClassName
```

- A class can implement from one interface

```
public class ClassName implements InterfaceName {  
    //class implementation  
} //end class ClassName
```

- A class can implement from more than one interface

```
public class ClassName implements InterfaceName, InterfaceName2 {  
    //class implementation  
} //end class ClassName
```

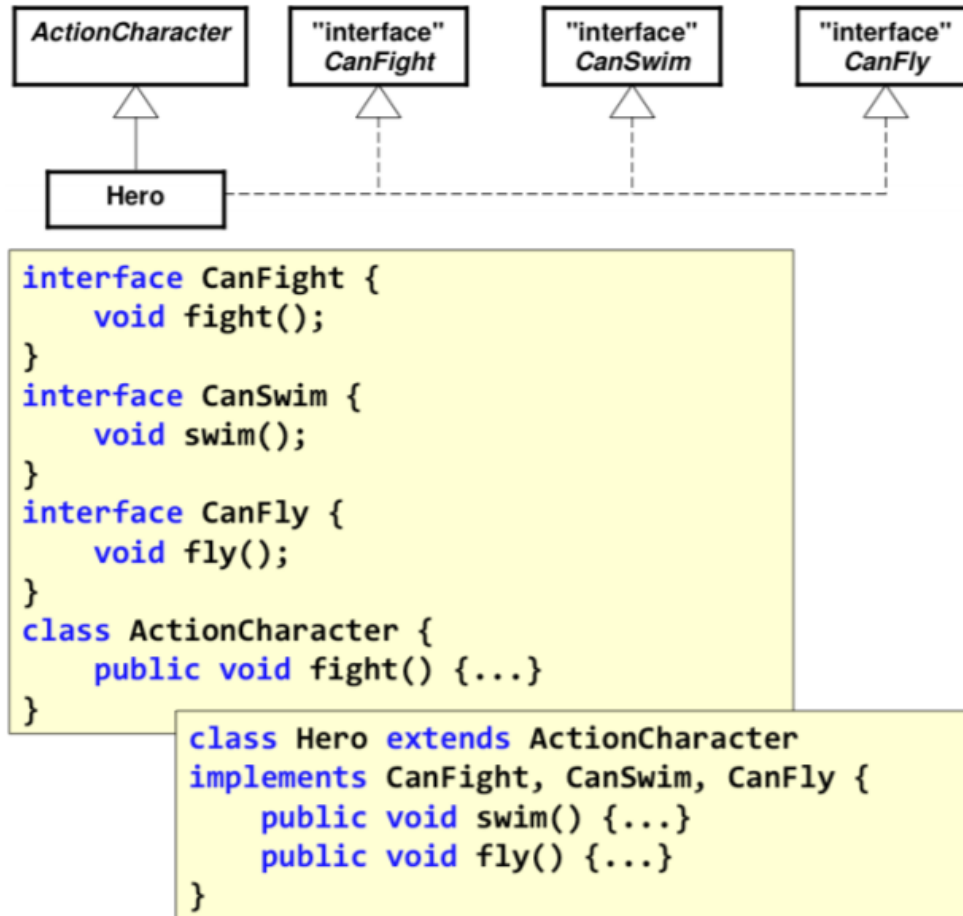


Interface example

```
public interface Comparable<T>{  
    public int compareTo(T o);  
}  
  
public class Name implements Comparable<Name>{  
    ...  
    public int compareTo(Name n){  
        ... //implementation of compareTo  
    }  
}
```



Implementing interface



Interface vs Abstract Class

Interfaces :

- Specify the form of a concept : not implementing it
- Cannot have data members, only constants
- Lightweight to implement
- Multiple implementation
- Based on “has-a” (composition)

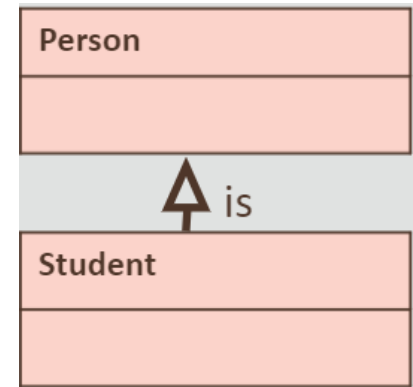
Abstract class :

- Incomplete class (dapat memiliki implementasi secara parsial)
- Dapat memiliki data member
- Superclass : digunakan untuk membentuk hirarki dari kelas
- Single inheritance
- Based on “is-a” (inheritance)



Contd..

- Abstract classes often have an “Is-A” relationship
 - A Student is a Person, so Person may be better as an Abstract Class
- Interfaces often have a “Has-A” relationship
 - A vehicle has an engine, so Engine may be better as an interface



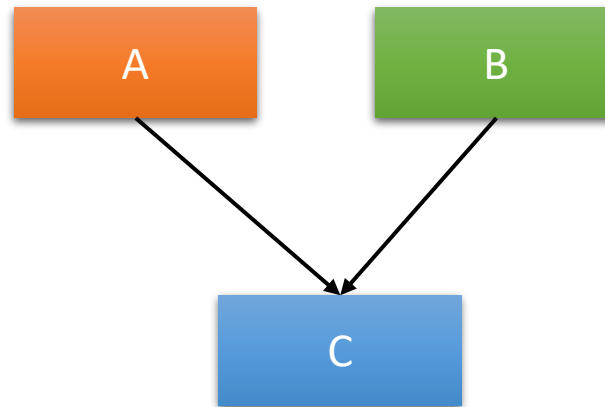
Diskusi

1. Apa pertimbangan dalam memilih abstract class vs interface ?
2. Manakah yang bersifat paling abstract diantara keduanya ?



(3) Multiple Inheritance

- Multiple inheritance merupakan suatu konsep dimana suatu kelas memiliki lebih dari 1 (satu) parent / superclass.

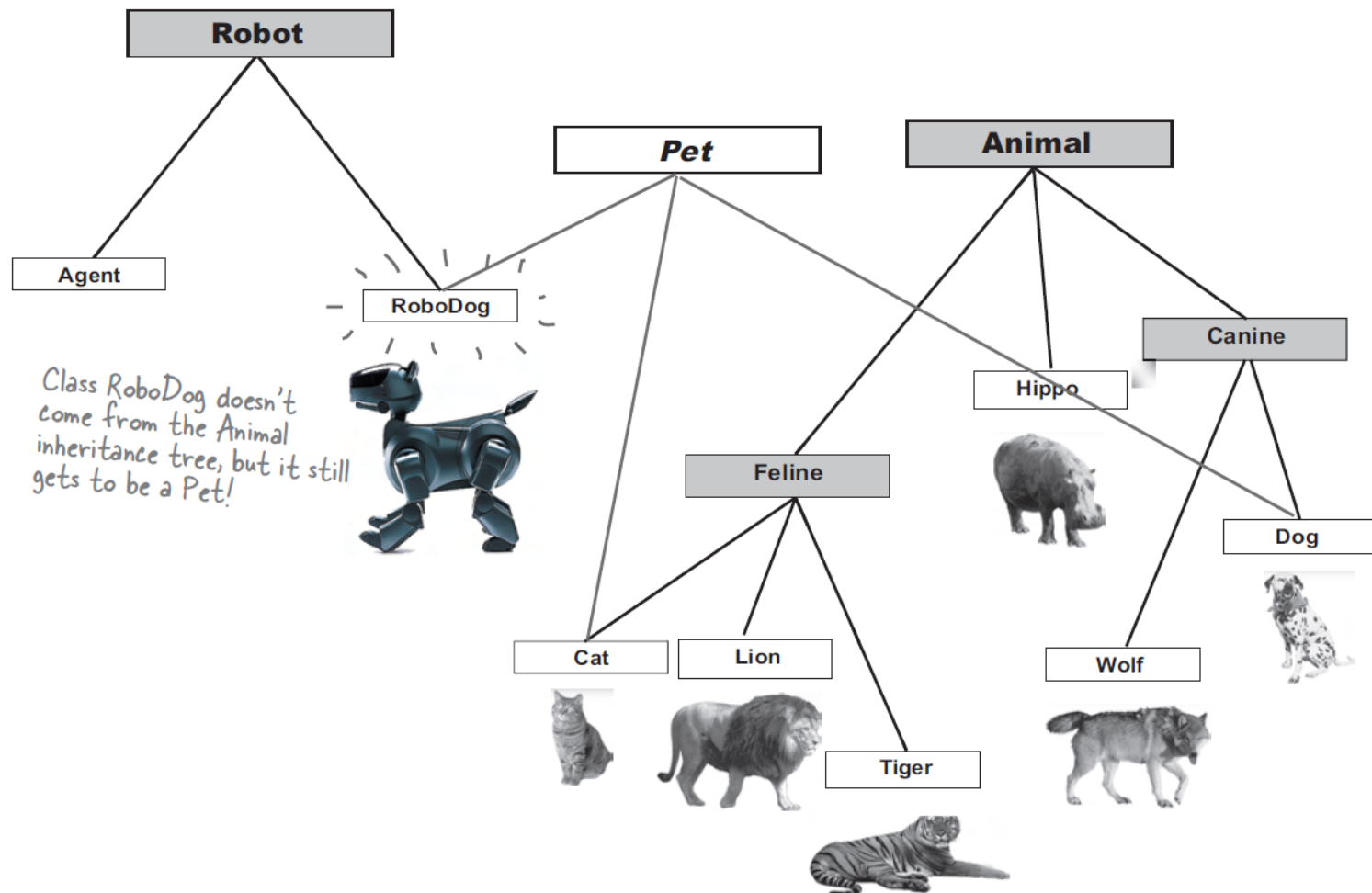


Multiple Inheritance

- Umumnya konsep ini jarang digunakan, karena Bahasa OO tidak mendukung dalam penerapan ini (ex : java)
- Namun untuk mengimplementasikannya, kita dapat menggunakan interface.



Contoh kasus multiple inheritance :



Contd..

- Kita dapat menggunakan interface untuk mengimplementasikan multiple inheritance
- Robodog → mengimplementasikan multiple inheritance
- Robodog → Inherit dari Animal (abstract class) dan Pet (interface)

Contoh deklarasi :

```
public class Robodog extends Animal implements Pet
{
    .....
}
```



Demo

- Demo contoh kasus menggunakan eclipse.



Questions



Conclusion :

- **abstract method**—a method which is declared but not defined (it has no method body)
- **abstract class**—a class which either (1) contains abstract methods, or (2) has been declared **abstract**
- **instantiate**—to create an instance (object) of a class
- **interface**—similar to a class, but contains only abstract methods (and possibly constants)
- **multiple inheritance** — object or class (subclass) can inherit from more than one parent object or class (superclass)



References :

1. https://www.irisa.fr/kerdata/people/Alexandru.Costan/prog2/PROG2_CM2.pdf
2. Object oriented programming and Java 2nd Edition – Chapter 6
3. HeadFirst Java 2nd Edition – Chapter 8
4. Oracle academy – Java Programming.

