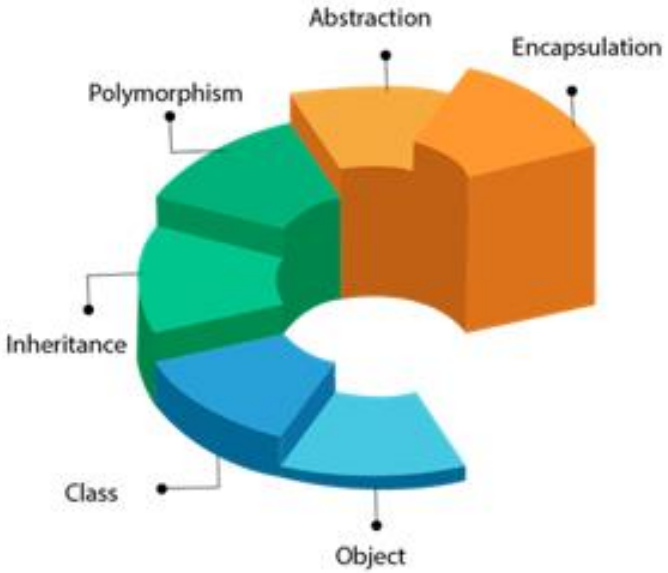


Pemrograman Berorientasi Object

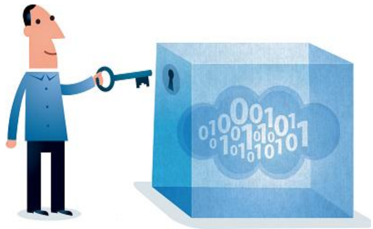
Inheritance

Saiful Nur Budiman S.Kom., M.Kom
Teknik Informatika – UNISBA 2024-2025

Materi yang Dibahas:

- **Konsep OOP**
- **Inheritance**
- **Single Inheritance**
- **Multiple Inheritance**
  - **Penggunaan kata kunci implements**

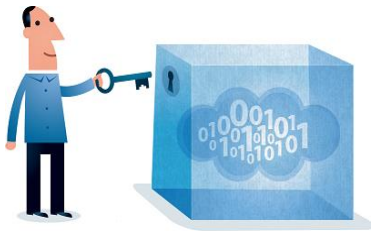

Java
JavaScript
Python
C++
Haskell
PHP
Pascal
Ruby
Objective-C
Perl
C
ABAP
Visual Basic
Transact-SQL
Clarion
Logo
Prolog
ColdFusion
Forth
PL/SQL
Ada
Lua
Yacc
TCL
Delphi
MATLAB
ActionScript
Erlang
C#
D
F#
ML
Scala
OpenEdge ABL
FoxPro
PostScript
Assembly
Scratch
Scheme
COBOL
Dart
Max/MSP
SAS
Haskell
Tcl
Groovy
Lisp
Mathematica
Swift
Fortran
Go
R
Q
PL/I



Konsep OOP

Konsep OOP

- Ada empat element utama penyusun OOP:
  1. **Inheritance**, pewarisan
  2. **Polymorphism**, banyak bentuk beda fungsi
  3. **Abstraction**, melihat object dalam bentuk yang sederhana. **Tidak bisa di instance secara langsung**, melainkan di **extends**
  4. **Encapsulation**, suatu cara untuk meyembunyikan implementasi detil dari class untuk mencegah akses yang ilegal. Ada dua kegunaanya yaitu information hiding dan modularitas



Inheritance

Inheritance (Pewarisan)

- Konsepnya **menciptakan class baru dengan mewarisi karakteristik class yang telah ada/ dibuat**
- Dengan konsep ini **memungkinkan class baru mewarisi fungsionalitas class yang sudah ada**.
- Untuk menciptakan class baru, hanya perlu **men-spesifikasi-kan cara class baru itu berbeda** dari class yang sudah ada

Inheritance (Pewarisan)

- Class yang sudah ada disebut dengan class induk (**Super Class**)
- Class yang mewarisi class induk disebut dengan class turunan (**Sub Class**)
- Java memungkinkan **pewarisan tunggal (single inheritance)** pada class dan bisa **pewarisan majemuk (multiple inheritance)** dengan menerapkan **interface**.

Keyword yang digunakan: **extends**, **super**, **override** dan **implements**

Class Diagram untuk Inheritance (Pewarisan)

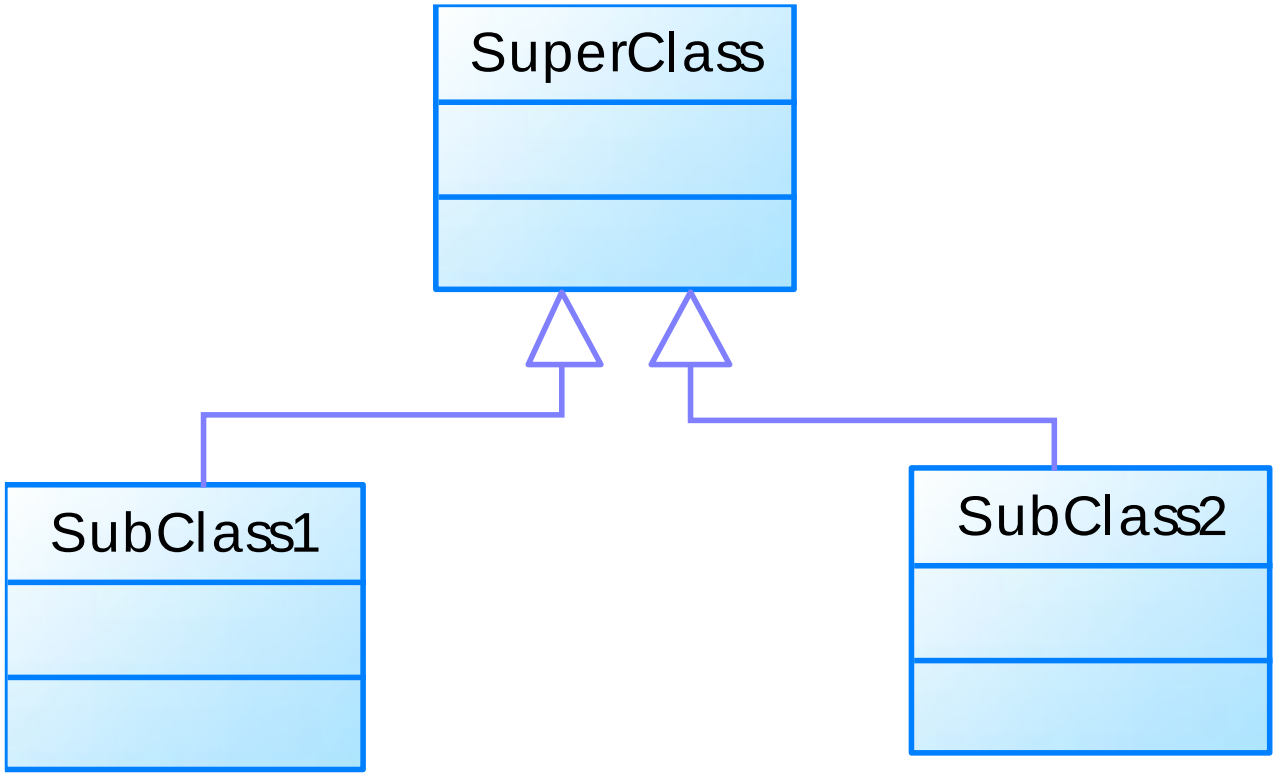
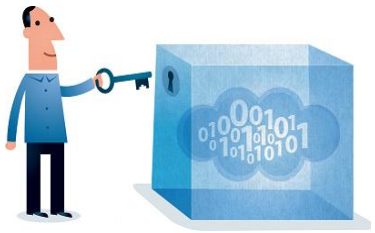


Diagram Inheritance

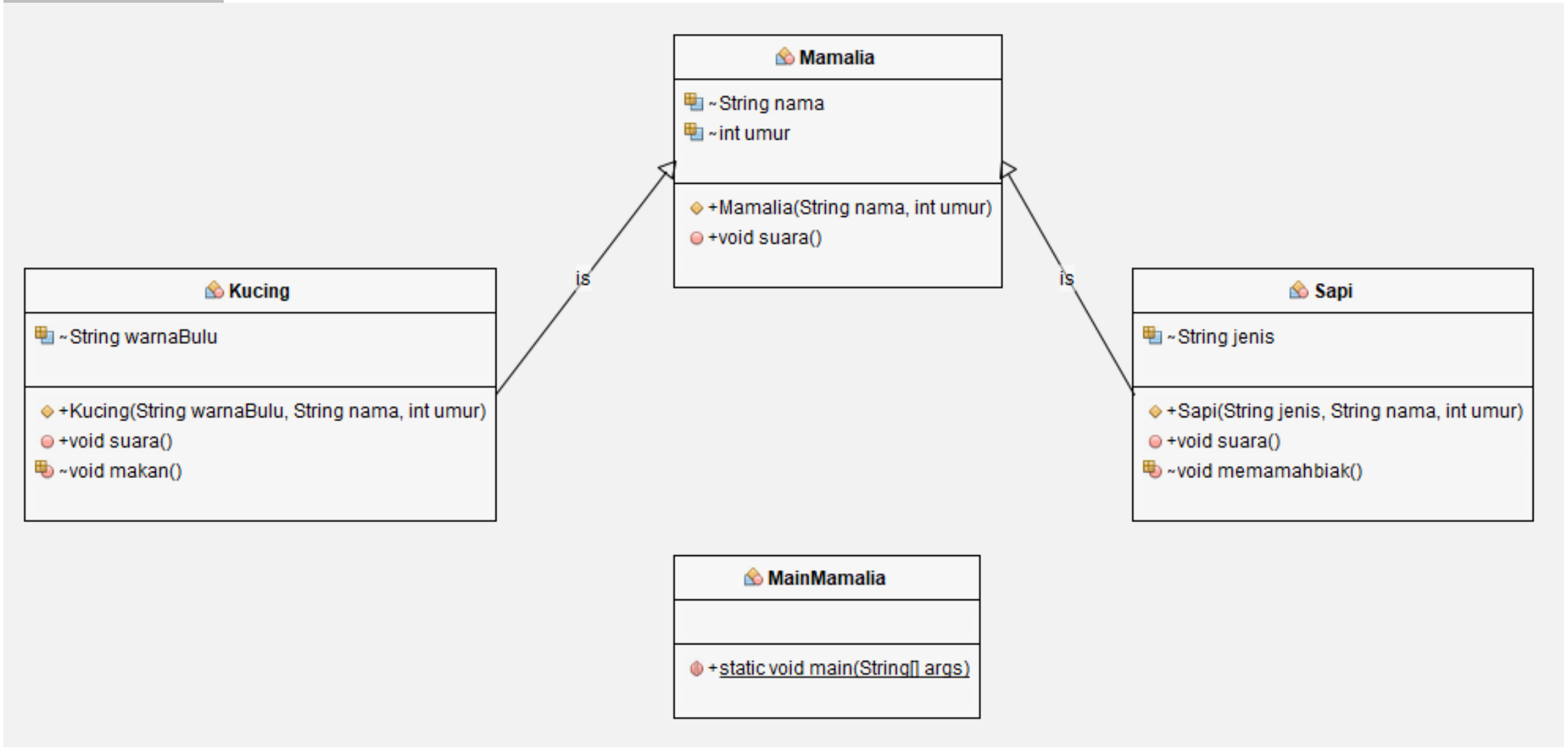


Single Inheritance

Single Inheritance

- Konsep dasarnya adalah **Generalization** (generalisasi) hubungan antara **Sub Class dengan Super Class**.
- Misalkan:
  - **Mamalia** adalah **generalization dari sapi dan kucing**
  - **Mamalia memiliki nama, umur dan suara**
  - **Sapi bersuara mooo dan memamahbiak**
  - **Kucing bersuara meong dan makan daging**

Class Diagram



```java
public class Mamalia {
    String nama;
    int umur;

    public Mamalia(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
    }

    public void suara(){
        System.out.println("suara");
    }
}
```

SingleInheritance
- Kucing.java
- MainMamalia.java
- Mamalia.java
- Sapi.java

Concrete Method

```java
public class Kucing extends Mamalia {

    String warnaBulu;

    public Kucing(String warnaBulu, String nama, int umur) {
        super(nama, umur);
        this.warnaBulu = warnaBulu;
    }

    @Override
    public void suara() {
        System.out.println("Suaranya meong");
    }

    void makan() {
        System.out.println("Kucing makan daging");
    }

}
```

```java
public class Sapi extends Mamalia {

    String jenis;

    public Sapi(String jenis, String nama, int umur) {
        super(nama, umur);
        this.jenis = jenis;
    }

    @Override
    public void suara() {
        System.out.println("Suaranya moooo");
    }

    void memamahbiak() {
        System.out.println("Sapi makan rumput");
    }

}
```

```
11  public class MainMamalia {
12
13      public static void main(String[] args) {
14          Kucing kucing = new Kucing("Oren", "Momoi", 2);
15          Sapi sapi = new Sapi("Brahma", "Milo", 5);
16
17          kucing.makan();
18          kucing.suara();
19
20          sapi.memamahbiak();
21          sapi.suara();
22      }
23  }
```

Kenapa object kucing dan sapi bisa memanggil function suara() dan hasilnya berbeda-beda?
Padahal diturunkan dari class Mamalia dengan **function suara()** outputnya "suara"

Output - Pewarisan (run)

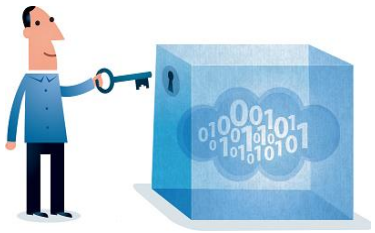
```
run:
Kucing makan daging
Suaranya meong
Sapi makan rumput
Suaranya moooo
```

```
11  public class MainMamalia {
12
13      public static void main(String[] args) {
14          Kucing kucing = new Kucing("Oren", "Momoi", 2);
15          Sapi sapi = new Sapi("Brahma", "Milo", 5);
16
17          kucing.makan();
18          kucing.suara();
19          System.out.println(kucing.nama);
20
21          sapi.memamahbiak();
22          sapi.suara();
23      }
24  }
```

Output - Pewarisan (run)

```
run:
Kucing makan daging
Suaranya meong
Momoi
Sapi makan rumput
Suaranya moooo
```

Tambahkan coding berikut? Bagaimana hasilnya dan kenapa **object kucing** bisa memanggil variable **nama**, padahal variable **nama** tidak dimiliki langsung oleh **class Kucing ?**



Multiple Inheritance

Multiple Inheritance

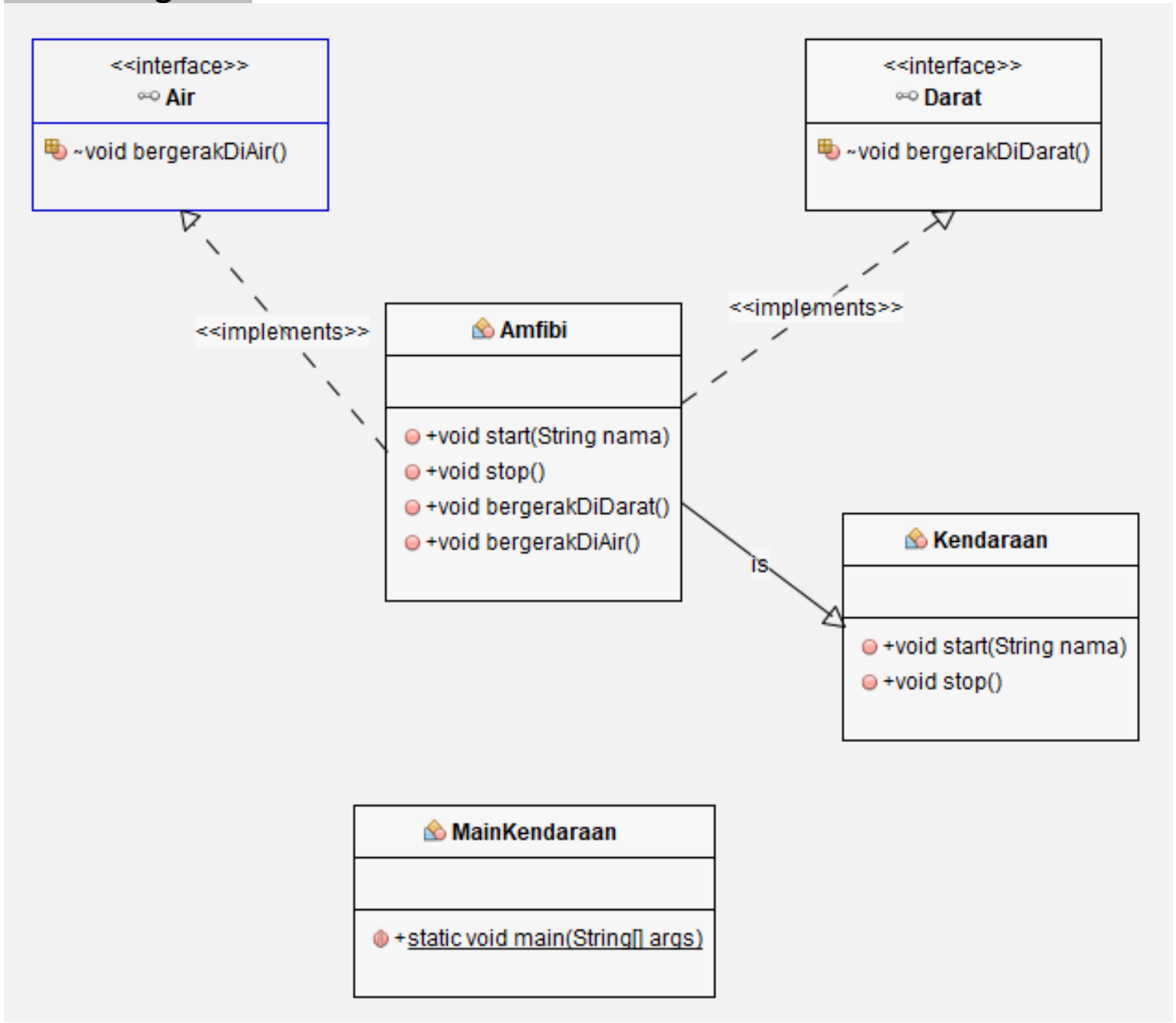
- Jika pada **Single Inheritance, setiap sub class mewarisi atribut ataupun method dari super class**.
- Sedangkan pada Multiple Inheritance, **memungkinkan suatu class mewarisi atribute maupun method lebih dari satu super class**.
- Prinsip multiple inheritance di java menggunakan **interface.**

Interface: kumpulan method yang hanya memuat deklarasi dan struktur method tanpa detail implementasinya

Komponen Penyusun Interface

Komponen Penyusun	Interface
Tipe data / variabel	**Hanya boleh berupa konstanta**
Method	**Hanya berupa *signature*, *method* yang terdapat dalam *interface* adalah *method* abstrak**
Sintaks *method*	**Tidak perlu membubuhkan *modifier* abstract pada semua *method* di dalam kelas**

Class Diagram



Studi Kasus:

- Kendaraan memiliki method **start()**, **stop()**
- Darat memiliki method **bergerakDiDarat()**
- Laut memiliki method **bergerakDiLaut()**
- **Amfibi** dapat bergerak di darat dan laut, sehingga harus mengimplementasikan kedua interface tersebut

```java
public interface Darat {
    void bergerakDiDarat();
}
```

```java
public interface Air {
    void bergerakDiAir();
}
```

Abstract Method

```java
public class Kendaraan {

    public void start(String nama){
        System.out.println("nyala");
    }

    public void stop(){
        System.out.println("mati");
    }
}
```

```
11  public class Amfibi extends Kendaraan implements Darat, Air{
12
13      @Override
14      public void start(String nama) {
15          System.out.println(nama +" dinyalakan");
16      }
17
18      @Override
19      public void stop() {
20          System.out.println("Kendaraan dimatikan");
21      }
22
23
24      @Override
25      public void bergerakDiDarat() {
26          System.out.println("Kendaraan bergerak di jalan");
27      }
28
29      @Override
30      public void bergerakDiAir() {
31          System.out.println("Kendaraan bergerak di air");
32      }
33
34  }
```

```
11  public class MainKendaraan {
12
13      public static void main(String[] args) {
14          Amfibi amfibi = new Amfibi();
15          amfibi.start("Panser APC Anoa 6x6 Amphibious");
16          amfibi.bergerakDiDarat();
17          amfibi.bergerakDiAir();
18          amfibi.stop();
19      }
20  }
```

Output

Debugger Console × | Pewarisan (run) ×

```
run:
Panser APC Anoa 6x6 Amphibious dinyalakan
Kendaraan bergerak di jalan
Kendaraan bergerak di air
Kendaraan dimatikan
```

Pertanyaanya….

- Apakah wajib mengimplemtasikan **semua method** yang dimiliki oleh interface pada subclass?
- Apakah wajib mengimplementasikan **semua method yang dimiliki oleh class Induk** ketika melakukan inheritance?

- **Ya wajib mengimplementasikan semua function yang dimiliki oleh interface ketika diimplemtasikan**
- Tidak, untuk **abstract class dan class biasa**, tidak wajib mengimplementasikan method. Bisa satu ataupun tidak sama sekali.

Kenapa Harus Menggunakan Inheritance?

- Jika kita ingin membuat sebuah class baru dan ternyata sudah ada class lain yang mengandung **sebagian** kode yang kita butuhkan.
- Maka class baru dibuat dengan menurunkan dari class lama dan **tinggal menambahkan kode yang belum ada**
- Penggunaan inheritance **sering digunakan saat menggunakan library**

Studi Kasus

- Apa yang terjadi jika akses modifier **public** diganti private/protected/default pada method start() di class Kendaraan, Jelaskan!

```
public class Kendaraan {

    private void start(String nama){
        System.out.println("nyala");
    }

    public void stop(){
        System.out.println("mati");
    }
}
```

“Don’t be afraid to make a mistake. But make sure you don’t make the same mistake twice”

Jangan takut untuk membuat sebuah kesalahan. Tapi pastikan Anda tidak melakukan kesalahan yang sama dua kali

~ Akio Morita ~

co-founder of Sony