Poliformisme

Brigida Arie Minartiningtyas, M.Kom.

Poliformisme

Kemampuan suatu objek untuk mengungkap banyak hal melalui satu cara yang sama

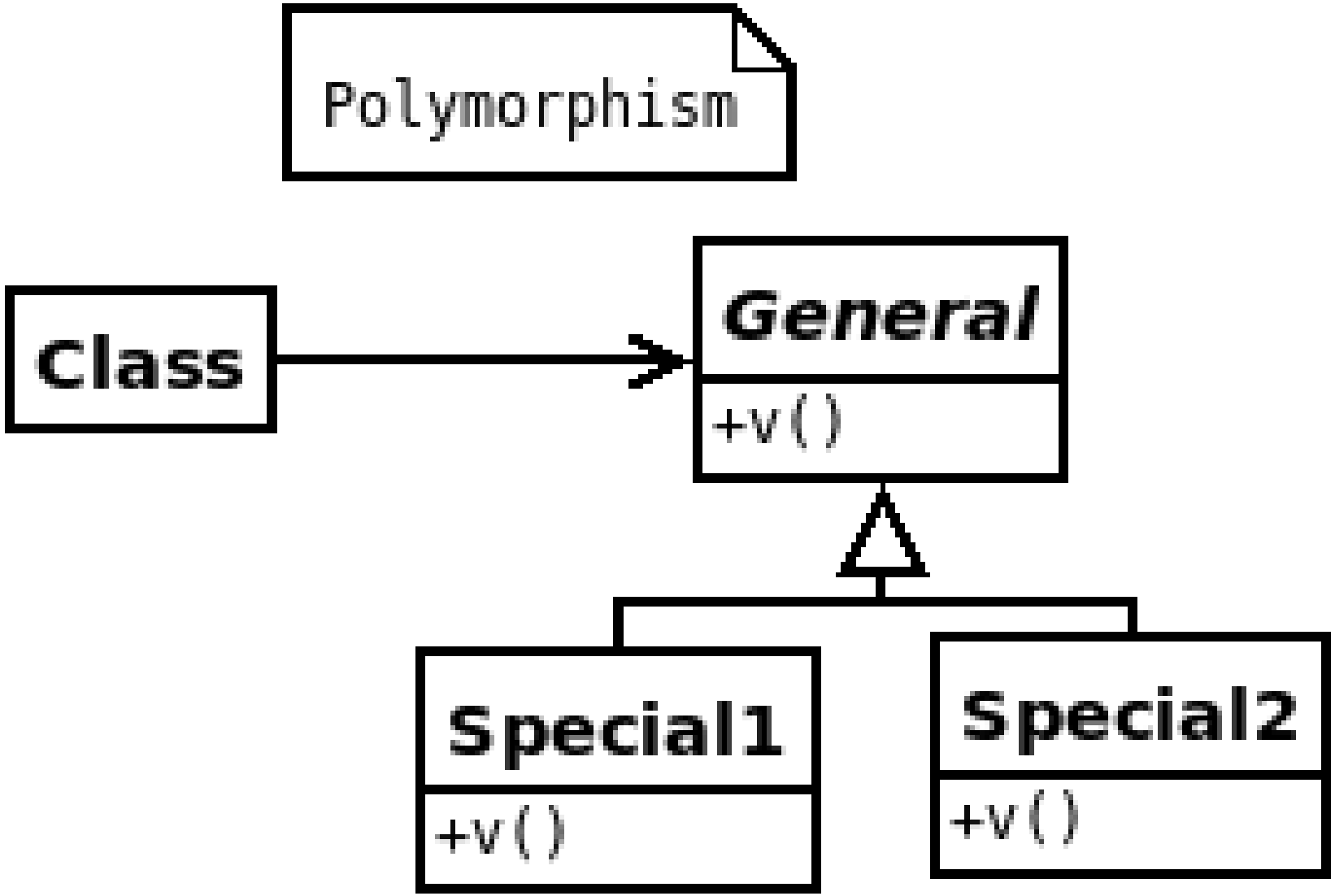
Poly - Morphos

Banyak Bentuk

Mengirimkan pesan yang sama ke berbagai objek yang berbeda

- Tiap objek memberi respons dengan cara yang berbeda:
  - bike15.start();
  - car11.start();

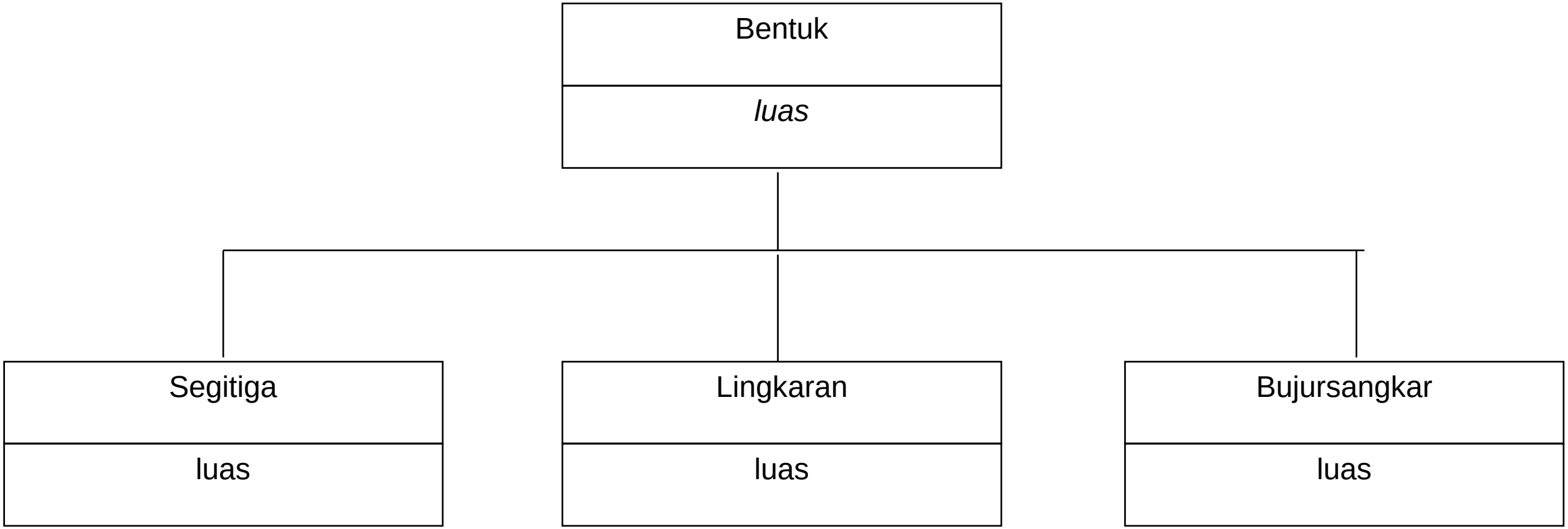
Polimorfisme mengijinkan kelas induk untuk mendefinisikan sebuah method **general** (bersifat umum) untuk semua kelas turunannya, dan selanjutnya kelas-kelas turunan dapat memperbarui implementasi dari method tersebut secara lebih **spesifik** sesuai dengan karakteristiknya masing-masing

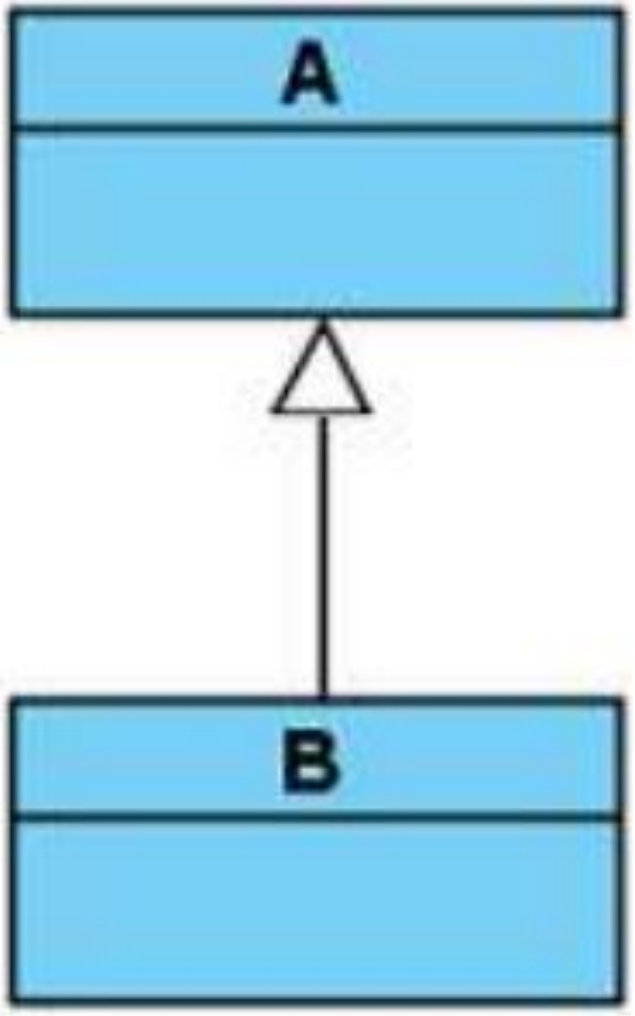


Object dinamis suatu class induk dapat berperilaku seperti class turunan

Ketika object menunjuk class induk, object berperilaku seperti class induk.

Ketika object menunjuk class turunan, object tersebut berperilaku seperti class turunan


Bentuk
luas
Segitiga
luas
Lingkaran
luas
Bujursangkar
luas




```
A  contoh = new A();
```

```
B  contoh = new B();
```

```
A  contoh = new B();
```

- ✓ Obyek contoh sebenarnya bertipe class A namun berperilaku secara class B
- ✓ Atribut dan method pada obyek contoh akan mengikuti class A, bukan class B

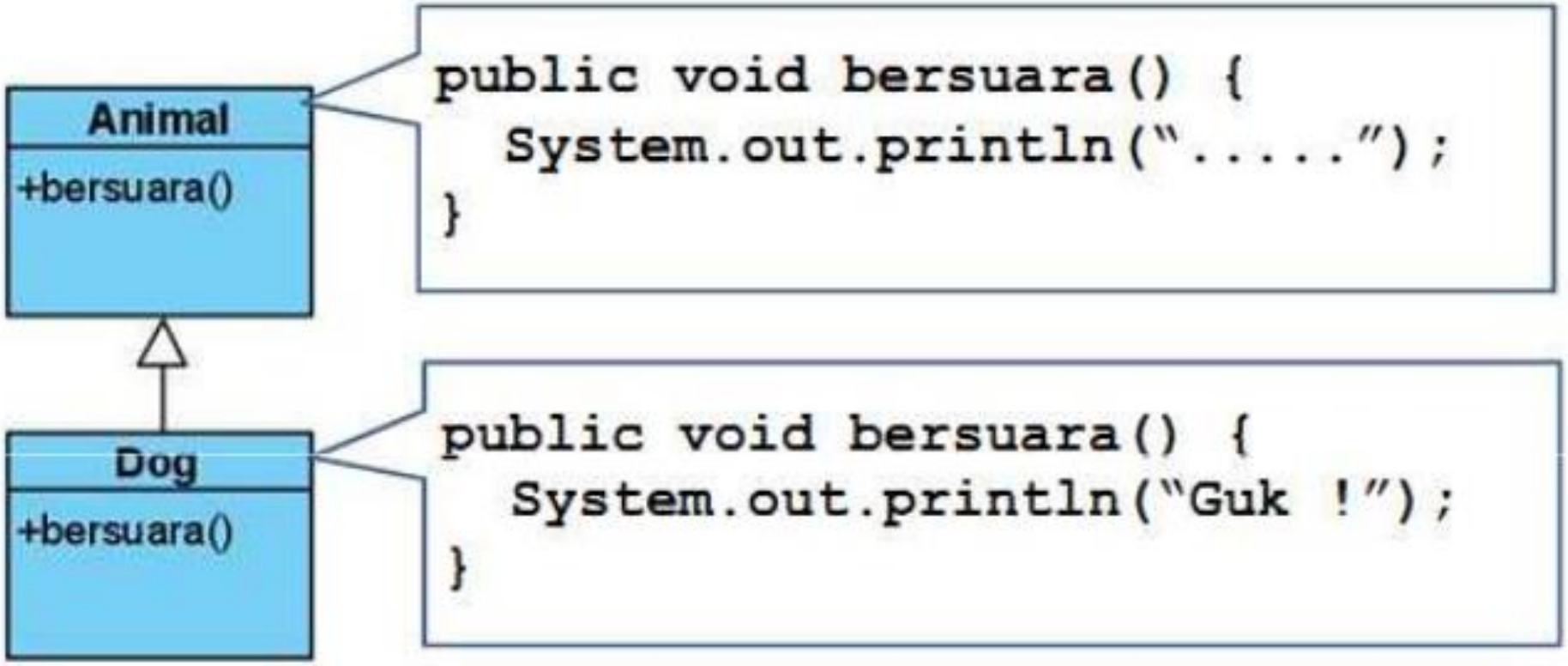
Overriding

Subclass yang berusaha memodifikasi tingkah laku yang diwarisi dari superclass.

Tujuan: subclass memiliki tingkah laku yang lebih spesifik

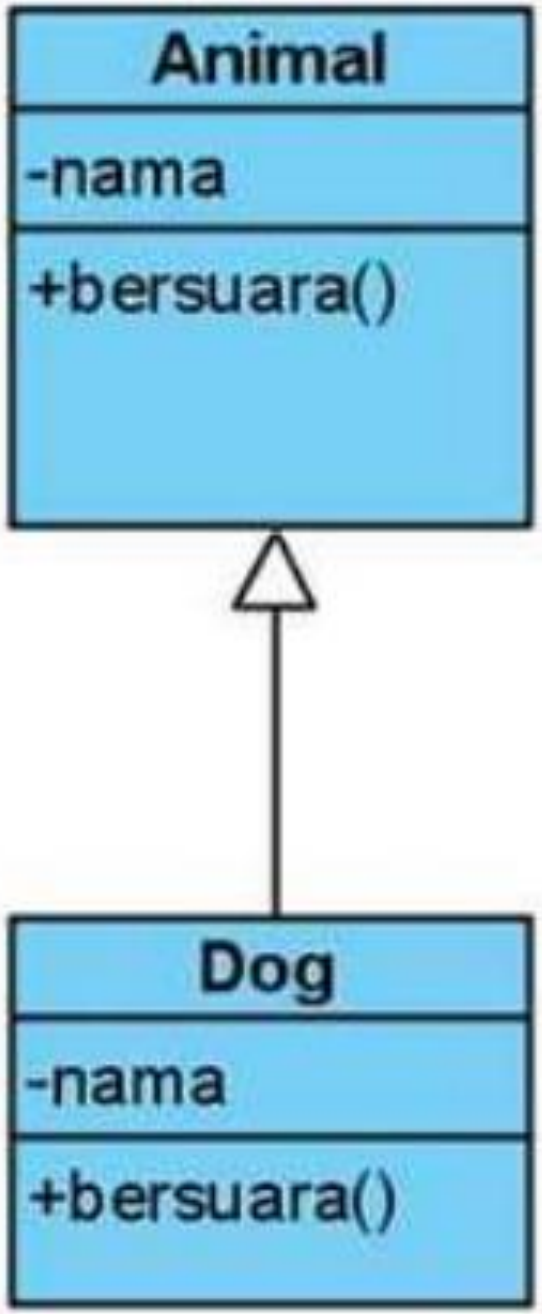
Dilakukan dengan cara mendeklarasikan kembali method milik parent class di subclass.

```
class Animal {
      public String bersuara() {
            return "Suara binatang";
      }
}

class Dog extends Binatang {
      public String bersuara() {
            return "Guk guk";
      }
}
```




```
Animal  contoh = new Animal();
contoh.bersuara();

Dog  contoh = new Dog();
contoh.bersuara();
```

```
Animal  contoh = new Dog();
contoh.bersuara();
```



Field dari class Animal

```
Animal  contoh = new Dog();
contoh.bersuara();
```

Method dari class Dog

Override Terhadap Method

Pada saat mendefinisikan suatu method di dalam kelas turunan yang nama dan daftar parameternya sama persis seperti yang terdapat pada kelas induk, maka dikatakan telah melakukan **override** (mengesampingkan) method yang tersimpan dalam kelas induk.

Apabila melakukan pemanggilan method yang telah di-overrride melalui objek dari kelas turunan, maka yang akan dieksekusi adalah *method yang terdapat pada kelas turunan*, bukan yang terdapat pada kelas induk.

Overloading

Nama method harus sama

Daftar parameter harus berbeda

Return type boleh sama, juga boleh berbeda

```
public void println (char c);
public void println (String s);
public void println (int i);
public void println (String s, double d);
```

```java
class Induk{
  public void metode1(int x){
    System.out.println("Induk1");
  }
  public void metode1(int x,int y){
    System.out.println("Induk1-2");
  }
  public void metode2(String x){
    System.out.println("Induk2");
  }
```

```
D:\>java Anak
ini dari Anak1 1
Induk1-2
ini Anak 3 parameter
Induk1-2
Induk2
Induk2
MetodeAnak
ini dari Anak1 1
Induk1-2
Induk2
```

```java
public class Anak extends Induk{
  public void metode1(int x){
    System.out.println("ini dari Anak1 "+x);
  }
  public void metode1(int x,int y,int z){
    super.metode1(x,y);
    System.out.println("ini Anak 3 parameter");
  }
  public void metodeAnak(){
    super.metode2("anton");
    System.out.println("MetodeAnak");
  }

  public static void main(String args[]){
      Anak a = new Anak();
      a.metode1(1);
      a.metode1(1,2,3);
      a.metode1(1,2);
      a.metode2("aaa");
      a.metodeAnak();

      Induk i = new Anak(); //polimorfisme
      i.metode1(1);
      i.metode1(1,2);
      i.metode2("aaa");
      //i.metodeAnak(); //error!
  }
```

```java
public class A_override {
    private int a;

    public void setA(int nilai){
        a = nilai;
    }

    public int getA(){
        return a;
    }

    public void tampilkanNilai(){
        System.out.println("Nilai a : "+getA());
    }
}
```

```java
public class B_override extends A_override {
    private int b;

    public void setB(int nilai){
        b = nilai;
    }

    public int getB(){
        return b;
    }

    public void tampilkanNilai(){
        System.out.println("Nilai b : "+getB());
    }
}
```

Override method tampilkanNilai()

```java
public class DemoOverride1 {
    public static void main (String [] args){

        B_override obj = new B_override();

        obj.setA(100);
        obj.setB(200);

        obj.tampilkanNilai();
    }
}
```

DemoOverride1 > main >

tput - Pertemuan9 (run)

```
run:
Nilai b : 200
BUILD SUCCESSFUL (total time: 0 seconds)
```

Override VS Overload

Override VS Overload

```java
public class Induk {
    public void test(){
        System.out.println("Method di dalam kelas Induk");
    }
}
```

```java
public class Turunan extends Induk {
    public void test (String s){
        System.out.println ("Method di dalam kelas Turunan");
        System.out.println("s : \""+s+"\"");
    }
}
```

Overload

```java
public class DemoOverloadTurunan {
    public static void main (String [] args){

        Turunan obj = new Turunan();

        obj.test();
        System.out.println();

        obj.test("Contoh Overload pada proses pewarisan");
    }
}
```

DemoOverloadTurunan > main >

tput - Pertemuan9 (run)

```
run:
Method di dalam kelas Induk

Method di dalam kelas Turunan
s : "Contoh Overload pada proses pewarisan"
BUILD SUCCESSFUL (total time: 0 seconds)
```

"Now Speak!"
"Woof"
"Quack"
"Meow"

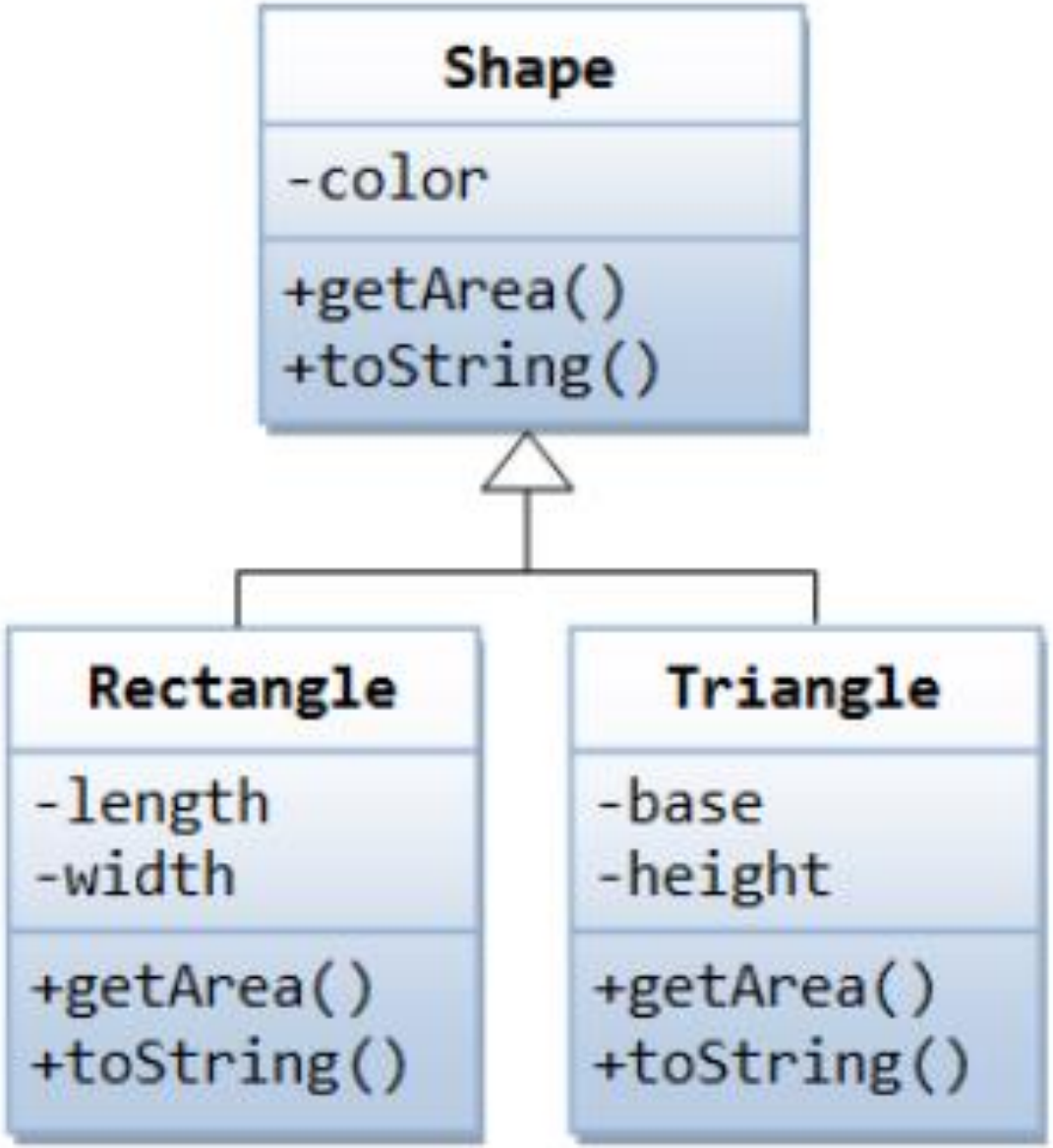
Polimorfisme Statik

- Diwujudkan dengan menggunakan overloading function, operator atau constructor
- Binding Method dilakukan secara statik
- Ciri :
  - Nama method sama
  - Parameter berbeda (tipe parameter dan/atau jumlah paramater dan/atau urutan parameter)
  - Tipe balikan tidak merupakan kriteria

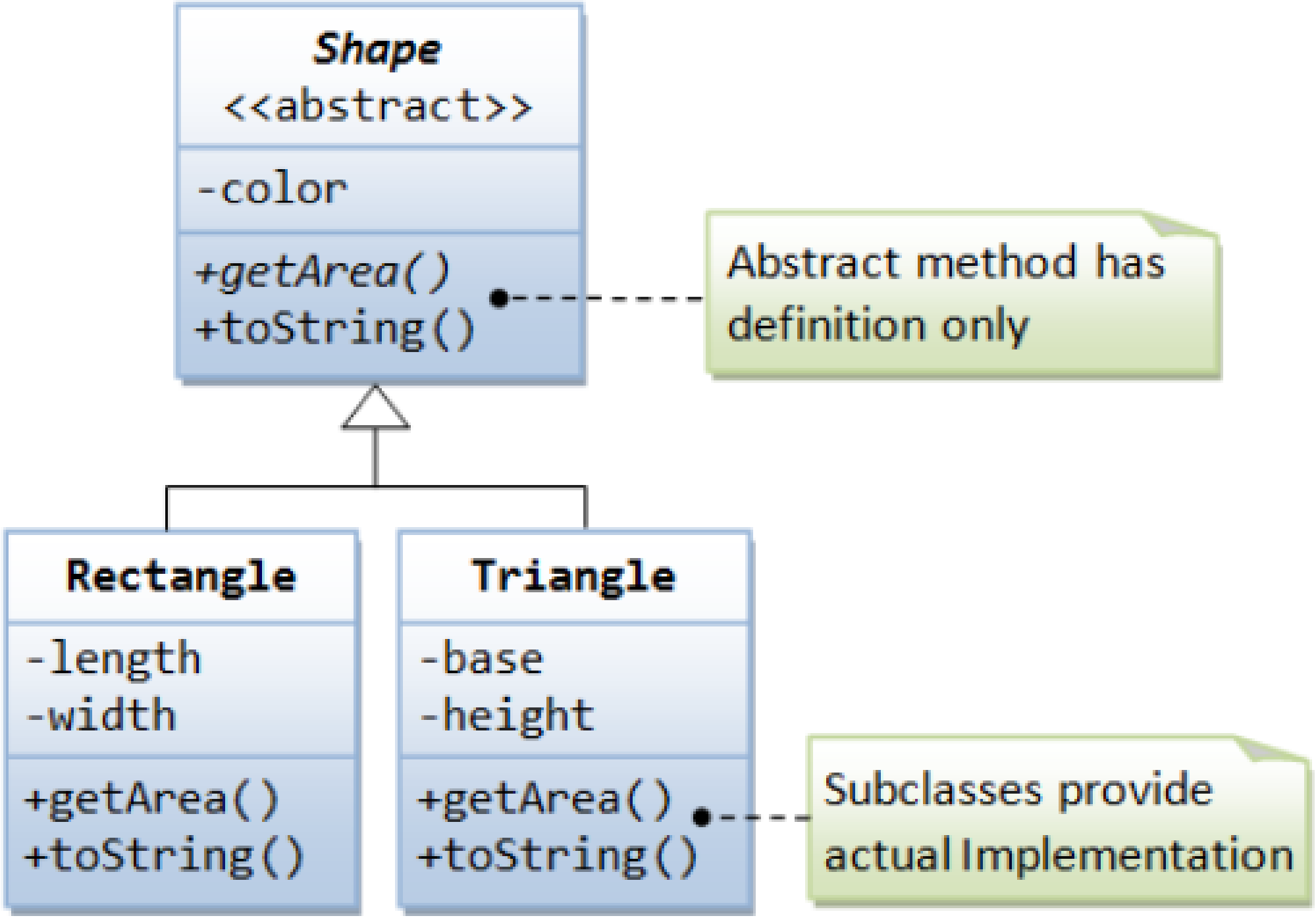
Polimorfisme Dinamik

- Overridding Method
- Interface
- Abstract Class di Java

Abstract Class



Pendefinisian kelas induk yang di dalamnya terdapat deklarasi method yang tidak memerlukan implementasi sama sekali


Shape
<<abstract>>
-color
+getArea()
+toString()
Abstract method has definition only
Rectangle
-length
-width
+getArea()
+toString()
Triangle
-base
-height
+getArea()
+toString()
Subclasses provide actual Implementation

Kelas Abstrak adalah kelas yang memiliki beberapa metode tanpa definisi lengkap (metode abstrak)

Kelas Abstrak digunakan sebagai kelas dasar untuk mendefinisikan kelas turunan

1. Tidak dapat menggunakan konstruktor kelas abstrak untuk membuat objek dari kelas abstrak
2. Hanya dapat membuat objek dari kelas turunan kelas abstrak

Kelas Abstrak

```
abstract class nama-kelas{
      badan kelas
}
```

Method Abstract

```
abstract tipe namaMethod (daftar-parameter)
```

```
abstract public class Shape {
    ......
    public abstract double getArea();
    public abstract void draw();
}
```

Method Abstract	Hanya berupa deklarasi di kelas abstrak
	Tidak memiliki method body
	Tidak bisa bertipe private
	Dimulai dengan keyword abstract
	Tidak bisa memiliki atribut final
	Sebuah kelas yang berisi satu atau lebih metode abstrak disebut kelas abstrak

```java
abstract public class Shape {
   // Private member variable
   private String color;

   // Constructor
   public Shape (String color) {
      this.color = color;
   }

   @Override
   public String toString() {
      return "Shape of color=\"" + color + "\"";
   }

   // All Shape subclasses must implement a method called getArea()
   abstract public double getArea();
}
```

Bagaimana menggunakan Kelas Abstrak?

- Buat kelas turunan dari kelas abstract tersebut dan impementasikan semua fungsi-fungsi yang dideklarasikan dikelas abstrak
- Apakah bisa semua method tidak diimplementasikan?
  - Bisa tapi kelas anak tersebut tetaplah harus abstract
- Object dapat diwujudkan pada kelas turunan dari abstract tapi tidak pada kelas abstrak

```java
public class TestShape {
   public static void main(String[] args) {
      Shape s1 = new Rectangle("red", 4, 5);
      System.out.println(s1);
      System.out.println("Area is " + s1.getArea());

      Shape s2 = new Triangle("blue", 4, 5);
      System.out.println(s2);
      System.out.println("Area is " + s2.getArea());

      // Cannot create instance of an abstract class
      Shape s3 = new Shape("green");   // Compilation Error!!
   }
}
```

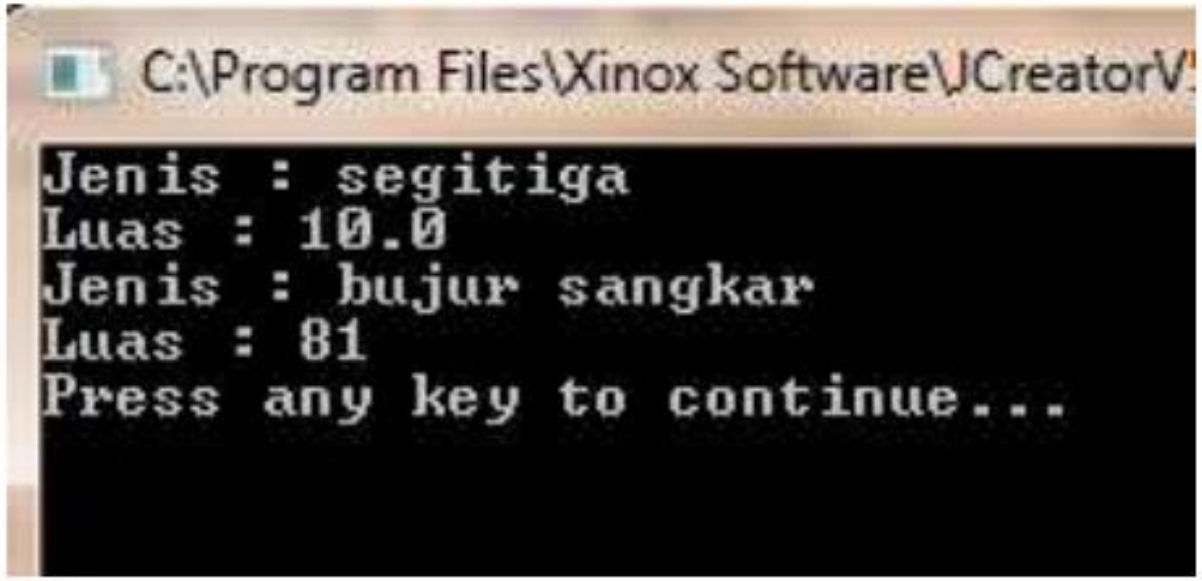
Sebagai Ringkasan, Abstract class menghasilkan sebuah template untuk pengembangan kelas ke depan yang bertujuan memberikan aturan, keseragaman nama, dan kontrak pada kelas turunannya

```
public abstract class Bentuk {
    private String jenis;
    public abstract void hitungLuas();
    public void setJenis(String j){
        this.jenis = j;
    }
    public String getJenis(){
        return this.jenis;
    }
}
```

```
class Segitiga extends Bentuk{
    private int alas;
    private int tinggi;

    public void hitungLuas(){
        System.out.println ("Luas : "+(0.5*alas*tinggi));
    }
    public Segitiga(String j,int a,int t){
        super.setJenis(j);
        this.alas = a;
        this.tinggi = t;
    }
}
```

```
class Persegi extends Bentuk{
    private int sisi;
    public void hitungLuas(){
        System.out.println ("Luas : "+(sisi*sisi));
    }
    public Persegi(String j,int s){
        super.setJenis(j);
        this.sisi = s;
    }
}
```

```
class Pemakai{
    public void Laporkan(Bentuk b){
        System.out.println ("Jenis : "+b.getJenis());
        b.hitungLuas();
    }

    public static void main(String args[]) {
        Pemakai p = new Pemakai();
        Segitiga s = new Segitiga("segitiga",5,4);
        Persegi pp = new Persegi("bujur sangkar",9);
        p.Laporkan(s);
        p.Laporkan(pp);
    }

}
```



"Now Speak!"
"Woof"
"Quack"
"Meow"

```
abstract public class Animal {
    abstract public void greeting();
}
```

```java
public class Cat extends Animal {
    @Override
    public void greeting() {
        System.out.println("Meow!");
    }
}
```

```java
public class Dog extends Animal {
    @Override
    public void greeting() {
        System.out.println("Woof!");
    }

    public void greeting(Dog another) {
        System.out.println("Wooooooooooof!");
    }
}
```

```java
public class BigDog extends Dog {
    @Override
    public void greeting() {
        System.out.println("Woow!");
    }

    @Override
    public void greeting(Dog another) {
        System.out.println("Wooooooowwwwww!");
    }
}
```

```java
public class TestAnimal {
    public static void main(String[] args) {
        // Using the subclasses
        Cat cat1 = new Cat();
        cat1.greeting();
        Dog dog1 = new Dog();
        dog1.greeting();
        BigDog bigDog1 = new BigDog();
        bigDog1.greeting();
```