

# Bahasa Pemrograman 2

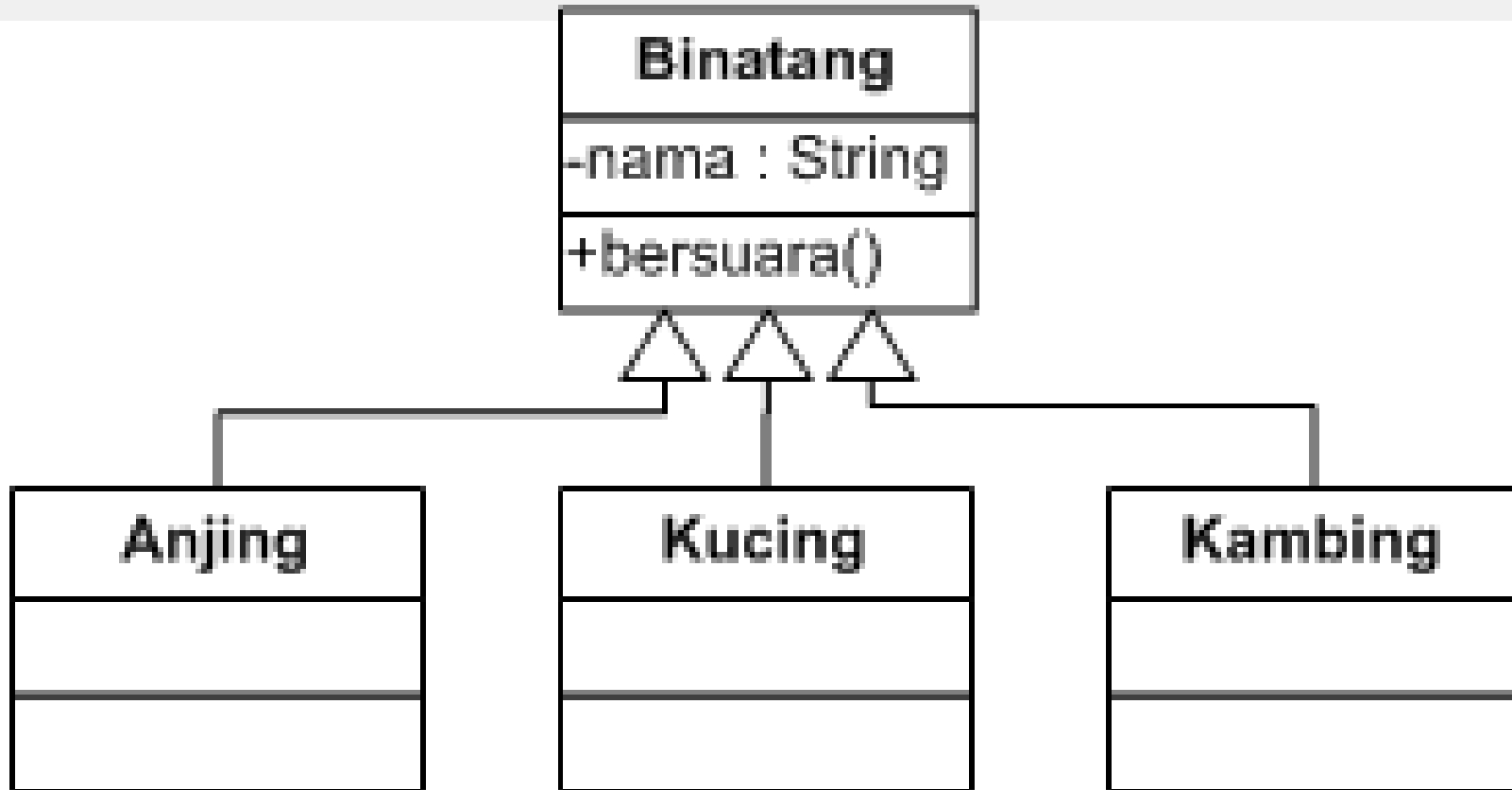
**Polimorfisme dan Binding**

anton@ukdw.ac.id

# Polymorphism

- Polymorphism = Poly + Morphos
- Poly = banyak, Morphos = bentuk
- Kemampuan obyek-obyek yang berbeda untuk memberi respons terhadap permintaan yang sama, *sesuai dengan cara masing-masing obyek*
- Polymorfisme dapat diterapkan secara:
  - Overriding Method
  - Overloading Method

# Polymorphism



# Review: 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**.

# Review: Overriding

- Deklarasi method pada subclass harus **sama** dengan yang terdapat di super class.  
Kesamaan pada:
  - Nama
  - Return type
  - Daftar parameter (jumlah, tipe, dan urutan)
- Method pada parent class disebut **overriden** method
- Method pada subclass disebut **overriding** method.

# Review: Overriding

- Class Anjing melakukan overriding terhadap method **bersuara()** yang diturunkan dari class Binatang
- Dengan overriding, suatu method yang didapatkan oleh subclass bisa melakukan **hal yang berbeda** dengan method asli pada superclassnya

# Review: Overriding

- Mode akses overriding method **harus sama** atau lebih **luas** dari pada overridden method.
- Subclass hanya boleh meng-override method superclass **satu kali saja**, tidak boleh ada lebih dari satu method pada kelas yang sama yang sama persis.
  - Jika lebih dari satu berarti ada 2 method yang sama persis dalam satu kelas

# Review: Overriding

- Contoh:

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

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

# Review: Overriding

- Dalam metode overriding, kita dapat memanggil metode overridden.
- Menggunakan keyword:  
**`super.namaMethod( )`**
- **super** menandakan konstruktor sekaligus superclass
  - Konstruktor disesuaikan dgn parameteranya
  - Superclass bisa memanggil methodnya

# Review: Overriding

- Contoh:

```
class Mahasiswa {  
    public void perkenalan() {  
        System.out.print("Objek mahasiswa");  
    }  
}
```

```
class MhsSI extends Mahasiswa {  
    public void perkenalan() {  
        super.perkenalan();  
        System.out.println("SI");  
    }  
}
```

# Review: Overriding

- Contoh:

```
class CobaMhs
{
    public static void main(String[] args)
    {
        Mahasiswa baru = new MhsSI();
        baru.perkenalan();
    }
}
```

# Review: Overloading Method

- Menuliskan kembali **method** dengan nama yang sama pada suatu class.
- Tujuan : memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang **mirip**.

# Overloading

Aturan overloading:

- Nama method harus sama
- Daftar parameter harus berbeda
- Return type boleh sama, juga boleh berbeda

# Overloading Method

- Contoh:

```
public void println (char c);
```

```
public void println (String s);
```

```
public void println (int i);
```

```
public void println (String s, double d);
```

# Contoh

```
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
```

```
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!  
    }  
}
```

# Polimorfisme

- Berhubungan dengan **abstract class**
- Kelas abstract “**memaksa**” kelas anak mengoverride method yg abstract dikelas induk sehingga kelas anak dapat “**bertingkah laku**” seperti yang diinginkannya!
  - *Berbeda* dengan induknya
  - Namun masih “**sejenis dengan induknya**”

# Polimorfisme

- Sifat “**sejenis**” dengan induknya itulah yang dapat digunakan untuk membuat kelas “**umum**” yang dapat menggunakan kelas yang bertipe induk namun bertingkah laku sesuai dengan kelas anaknya
- Contoh

```
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);
    }
}

```

```

C:\Program Files\Xinox Software\JCreatorV...
Jenis : segitiga
Luas : 10.0
Jenis : bujur sangkar
Luas : 81
Press any key to continue...

```

# Proses binding obyek oleh compiler

- Binding = pengikatan obyek oleh compiler
- Static binding
- Dyanamic binding

# Static Binding

- Binding dilakukan saat **compile-time**
- Overloading berarti ada beberapa method dengan nama yang sama.
  - Method yang dijalankan tergantung dari parameter yang diberikan
- Pemilihan method yang dijalankan ditentukan saat **compile time** (static binding)

# Static Binding

Contoh:

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

```
println("PBO");  
println(2000);  
println("a");  
println('b');  
println("Data", 2.0);
```

# Dynamic Binding

- Binding dilakukan saat **run-time** (dynamic)
- Penentuan method mana yang dijalankan dilakukan saat **run-time**

Contoh:

```
void kill (Mortal m) {  
    m.killed();  
}
```

Misalnya orang dan tanaman masuk dalam class Mortal, maka method killed() yang dipanggil tergantung dari siapa pemanggilnya.

# Contoh

```
class Mortal{
    public void showMortal(){ System.out.println("Mortal"); }
    public void killed(){ }
}

class Tree extends Mortal{
    public void showTree() { System.out.println("Tree"); }
    public void killed() { System.out.println("Tree killed"); }
}

class Animal extends Mortal{
    public void showAnimal() { System.out.println("Animal"); }
    public void killed() { System.out.println("Animal killed"); }
}
```

```
E:\Documents\Dosen\pbo\latihan>java Utama
Tree
Animal
Animal killed
Tree killed
```

```
public class Utama{
    public void kill(Mortal m){
        m.killed();
    }
    public static void main(String args[]){
        Tree t = new Tree();
        t.showTree();
        Animal a = new Animal();
        a.showAnimal();
        Utama u = new Utama();

        u.kill(a);
        u.kill(t);

    }
}
```

# Dynamic Binding

- Dynamic binding dapat dilakukan dengan 2 cara:
  - Overriding Inheritance Method
  - Overriding Interface Method

# Overriding Inheritance Method



# Polymorphism

```
class Binatang {  
    private String name;  
    public Binatang(String n) {  
        name = n;  
    }  
    public String getName() {  
        return name;  
    }  
  
    public void bersuara() {  
        System.out.println("Binatang bersuara");  
    }  
}
```

# Polymorphism

```
class Anjing extends Binatang {  
    Anjing(String n) {  
        super(n);  
    }  
  
    public void bersuara() {  
        System.out.println("Guk guk guk");  
    }  
}
```

# Polymorphism

```
class Kucing extends Binatang {  
    Kucing(String n) {  
        super(n);  
    }  
  
    public void bersuara() {  
        System.out.println("meong meong");  
    }  
}
```

# Penentuan Polimorfisme

- Penentuan “siapa” obyek sesungguhnya yang diacu oleh kompuler dapat diakses menggunakan **casting** sesuai dengan tipe data class anak yang sebenarnya!

# Contoh Casting Polymorfisme

```
class A extends Object{  
    public void m(){  
        System.out.println("m in class A");  
    }  
}
```

```
class B extends A{  
    public void m(){  
        System.out.println("m in class B");  
    }  
}
```

```
public class CobaPoly{  
    public static void main(String[] args){  
        Object var = new B();  
        ((B) var).m();  
    }  
}
```

# Aturan interface

- Suatu interface dapat mengekstens interface lain
- Tujuannya: memperluas kemampuan interface
- Interface hanya bisa extends interface saja
  - Tidak bisa extends class & abstract class

# Overriding Interface Method

Contoh:

```
interface I1{  
    public void p();  
}
```

```
interface I2 extends I1{  
    public void q();  
}
```

# Overriding Interface Method

```
class A extends Object{  
    public String toString(){  
        return "toString in A";  
    }  
    public String x(){  
        return "x in A";  
    }  
}
```

# Overriding Interface Method

```
class B extends A implements I2{  
    public void p(){ System.out.println ("p() in B"); }  
    public void q(){ System.out.println ("q() in B"); }  
}
```

```
class C extends Object implements I2{  
    public void p(){ System.out.println ("p() in C"); }  
    public void q(){ System.out.println ("p() in C"); }  
}
```

```
public class demoInterface {  
  
    public static void main(String[] args) {  
  
        A a = new A();  
        System.out.println (a.x());  
  
        C c = new C();  
        c.p();  
        c.q();  
  
        B b = new B();  
        b.p();  
        b.q();  
  
    }  
}
```



```
x in A  
p() in C  
p() in C  
p() in B  
q() in B  
Press any key to continue...
```

# NEXT

- Package & Class-class penting