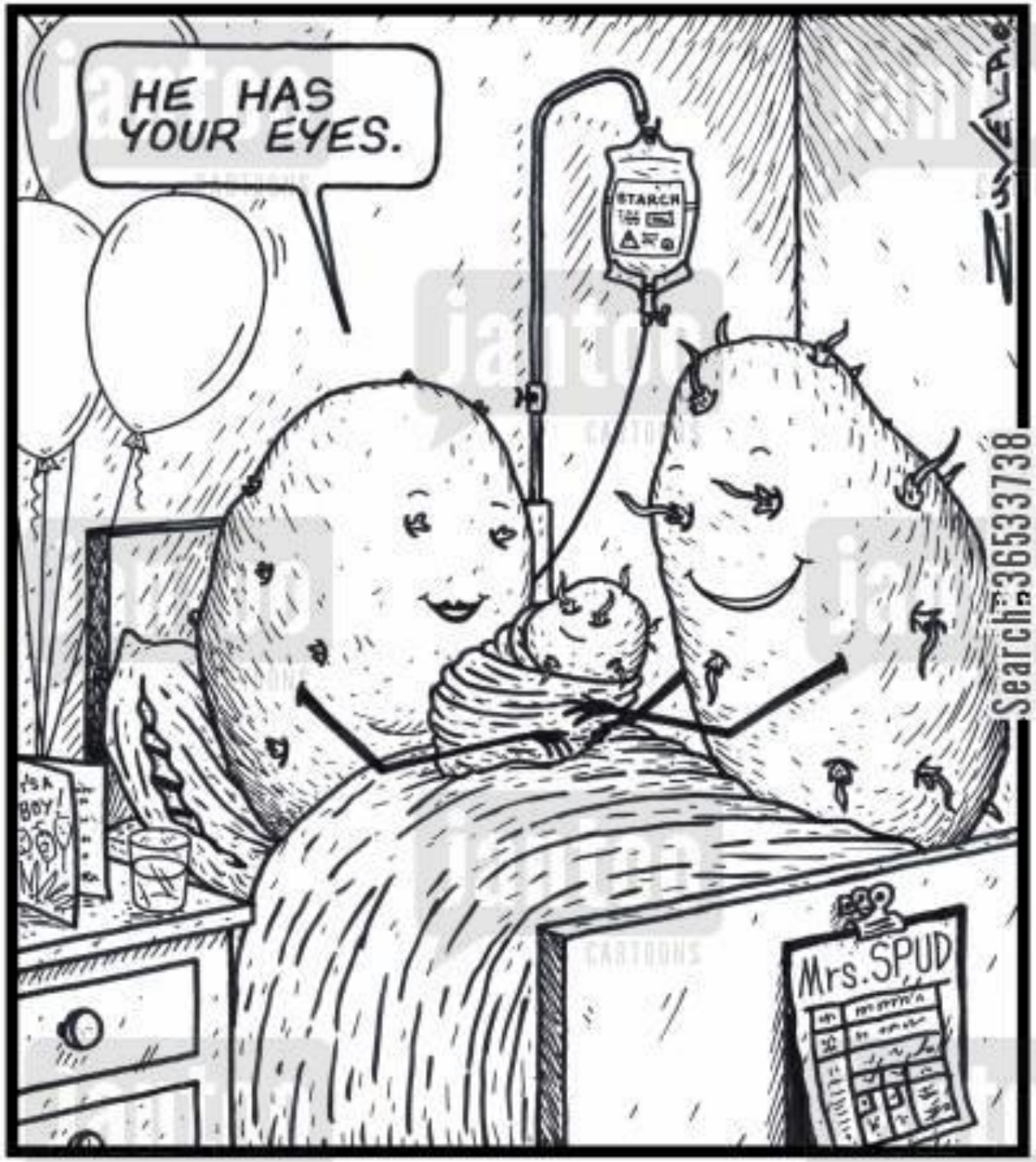
THE INHERITANCE

Brigida Arie Minartiningtyas, M.Kom.


HE HAS YOUR EYES.
STARCH
Mrs. SPUD
ZUVELA
Search: 36533738

Grouping

Sampai saat ini kita telah melakukan pengelompokan terhadap objek objek dalam kelas. Pada hari ini kita akan melakukan pengelompokan dengan sudut pandang yang lain

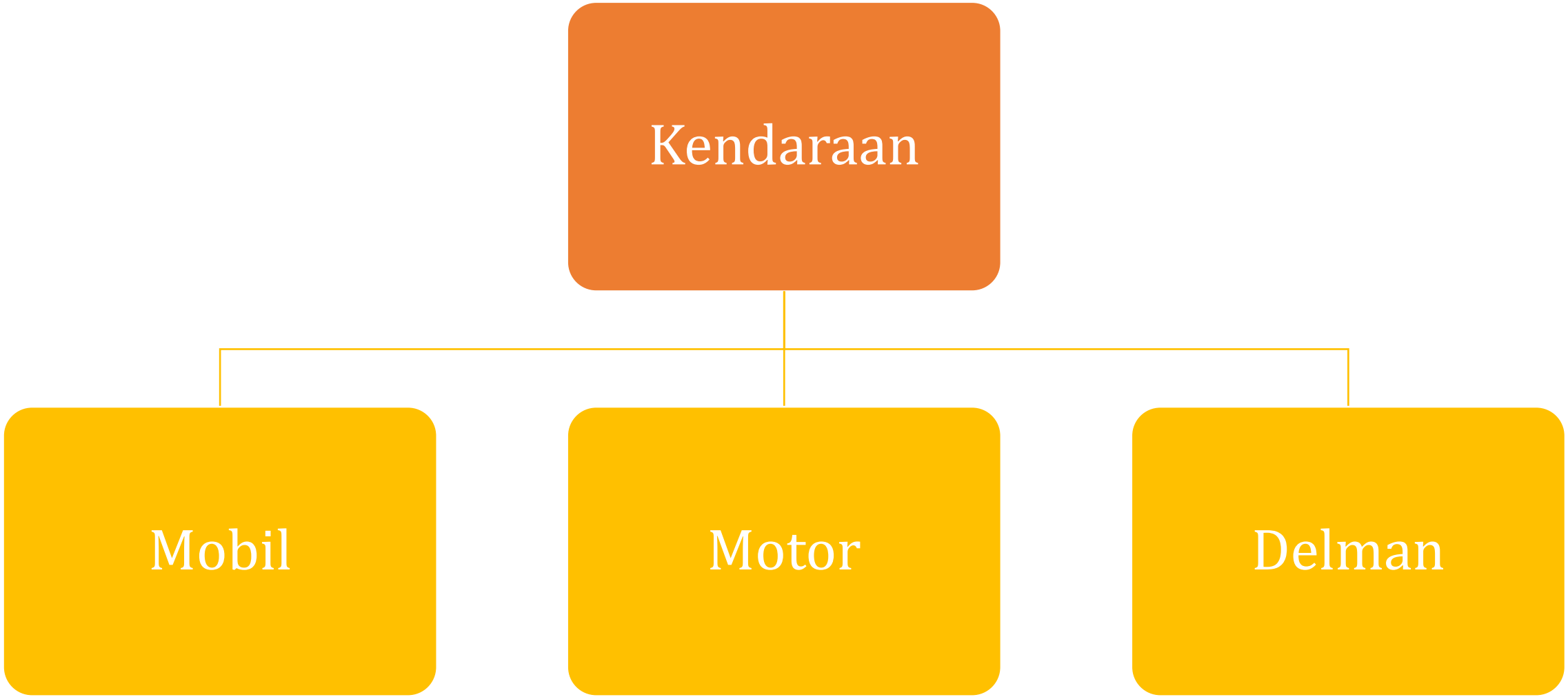
Selain melakukan katagorisasi terhadap objek yang memiliki sekumpulan atribut dan perilaku yang sama, manusia sering melakukan pengelompokan terhadap objek yang memiliki kesamaan atas beberapa (**sebagian**) atribut /perilaku

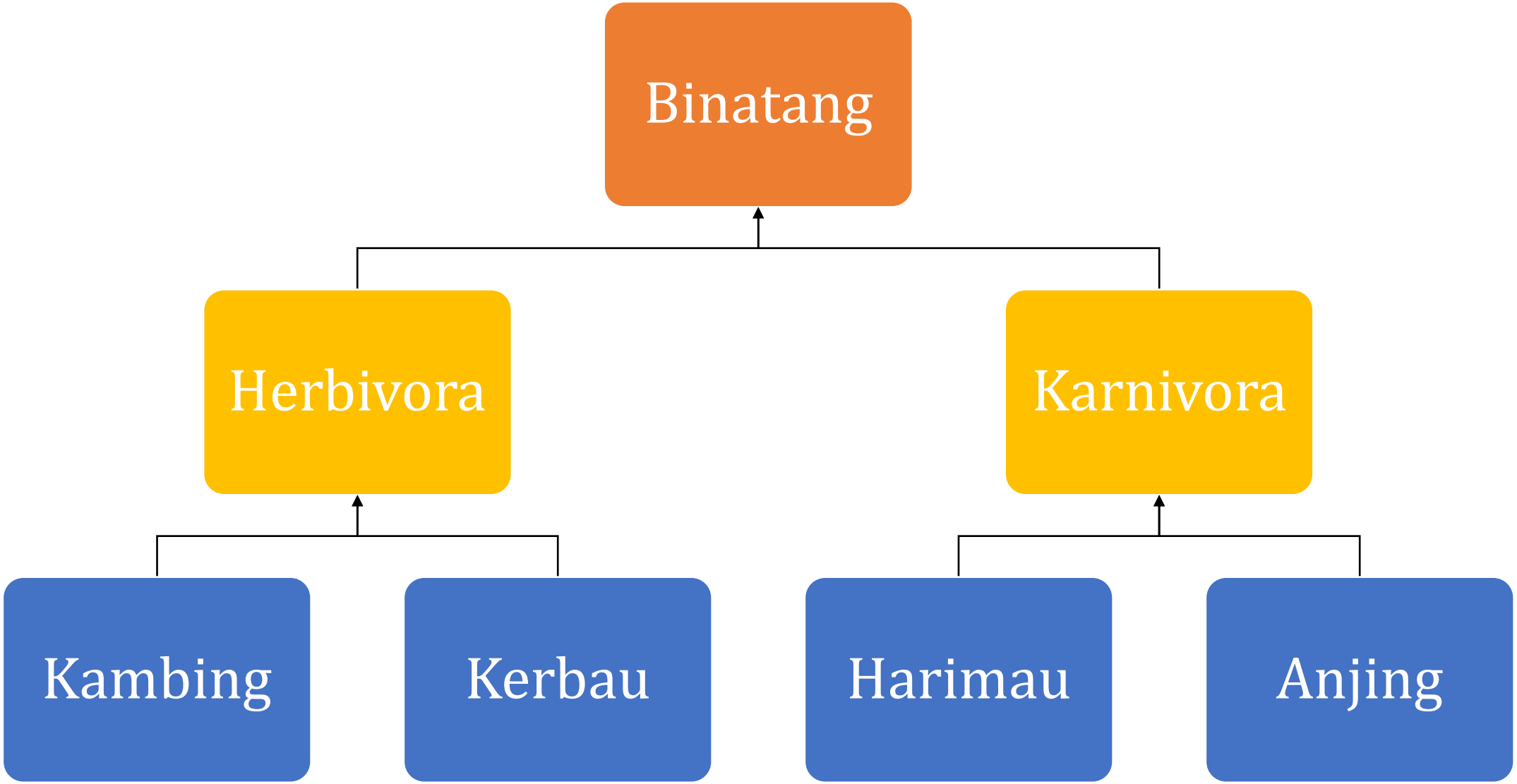
Is-a Relationship

Contoh :

Pengelompokan atas kendaraan bermotor, kemudian meng-grupkannya berdasarkan suatu tipe atau jenis (mobil, truk, sepeda motor, dll.)

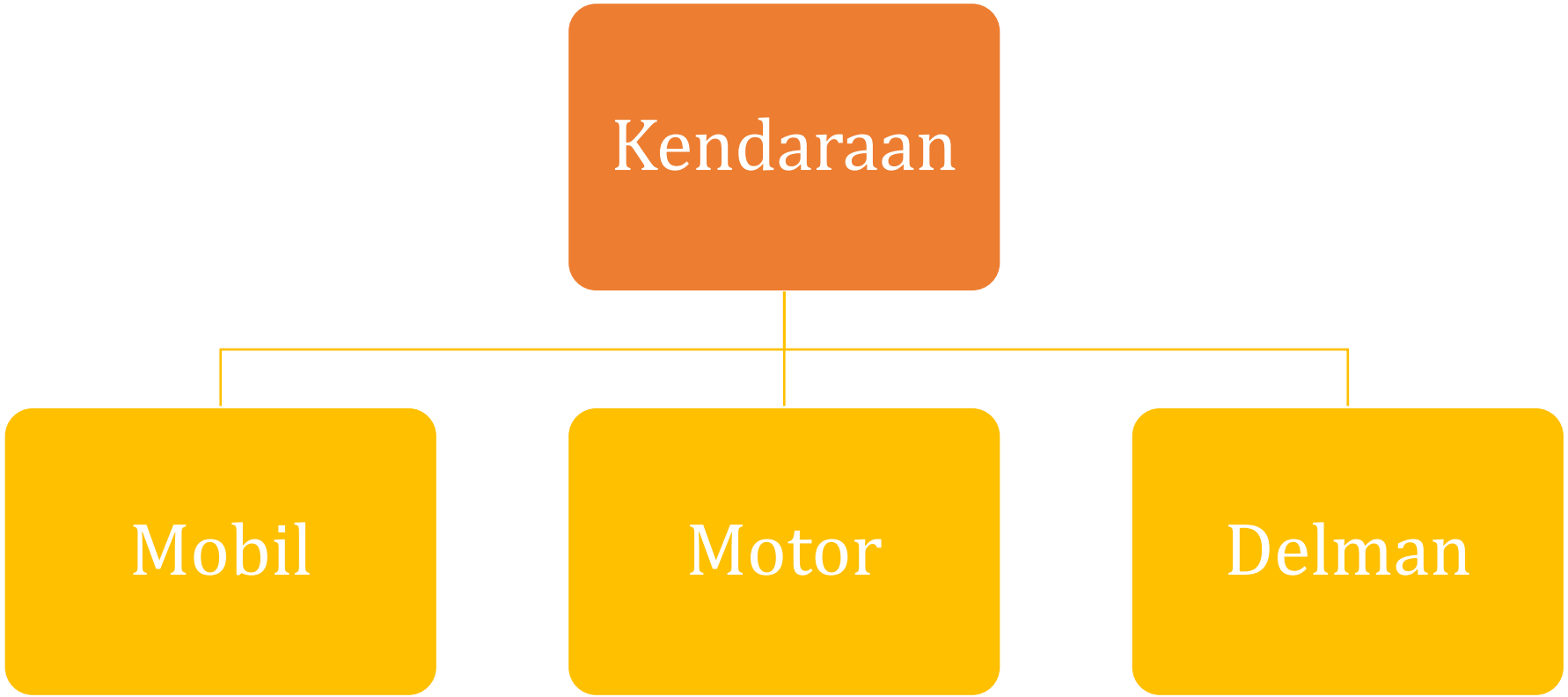
Memiliki atribut yang serupa tetapi memiliki atribut yang berbeda sebagai penciri khusus


Kendaraan
Mobil
Motor
Delman

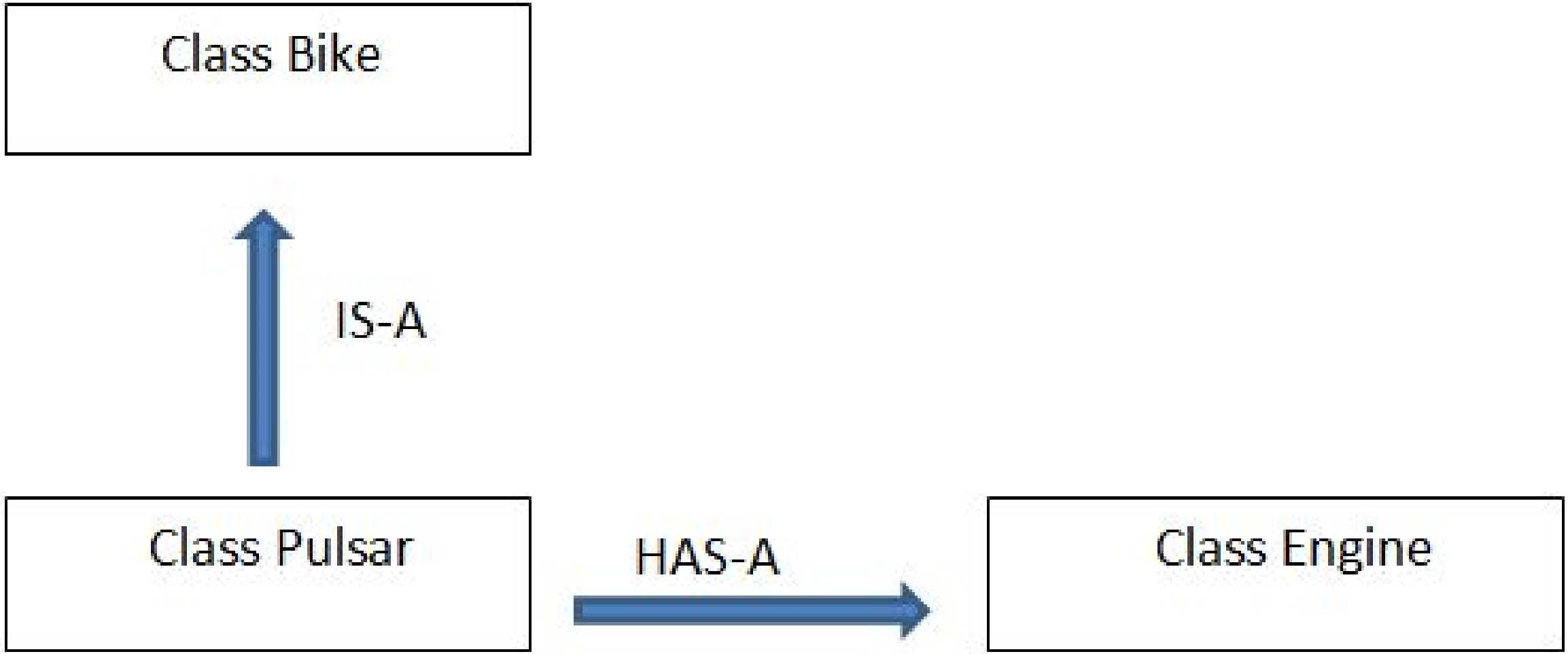


Java mengijinkan untuk mendefinisikan suatu kelas yang bersifat generik. Selanjutnya, kelas tersebut dapat diturunkan lagi menjadi kelas baru dengan sifat yang lebih spesifik

Objek yang dikelompokkan dalam satu kelas men-share sekumpulan atribut dan perilaku



Keterkaitan antar kelas dalam relasi "is-a" berasal dari kenyataan bahwa sub kelas memiliki atribut dan perilaku yang dimiliki kelas induk, ditambah atribut dan perilaku yang dimiliki oleh sub kelas tersebut


Class Bike
IS-A
Class Pulsar
HAS-A
Class Engine

Inheritance

SuperClass	SubClass
• Kelas Dasar atau Kelas Induk • Merupakan kelas yang lebih general dalam relasi “is-a”	• Kelas Turunan atau Kelas Anak • Merupakan kelas yang lebih spesifik dalam relasi “is-a”

Inheritance

Objek yang dikelompokkan dalam sub kelas memiliki atribut dan perilaku kelas induk, dan juga atribut dan perilaku tambahan

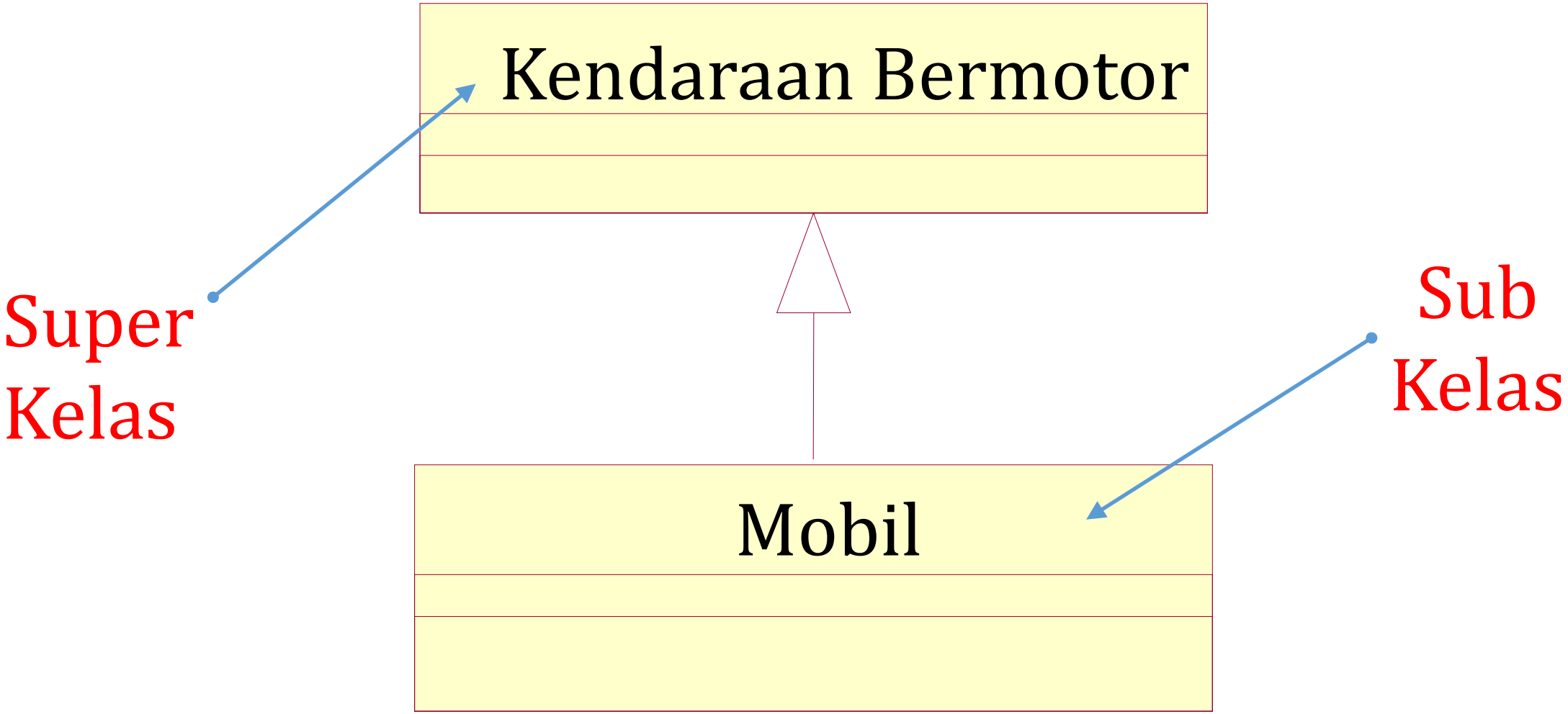
Jadi, kumpulan atribut dan perilaku sub kelas lebih besar dari super kelas-nya

Inheritance

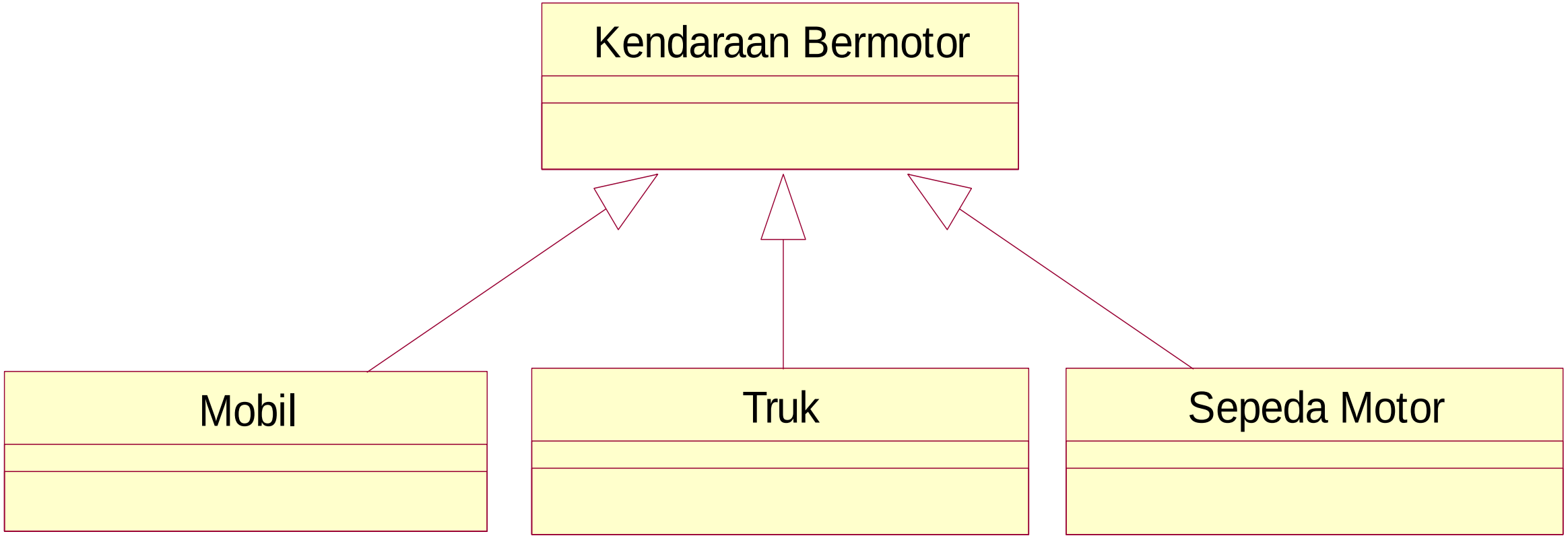
Relasi “is-a” antar superclass dan subclasses-nya disebut dengan **pewarisan** atau **inheritance**

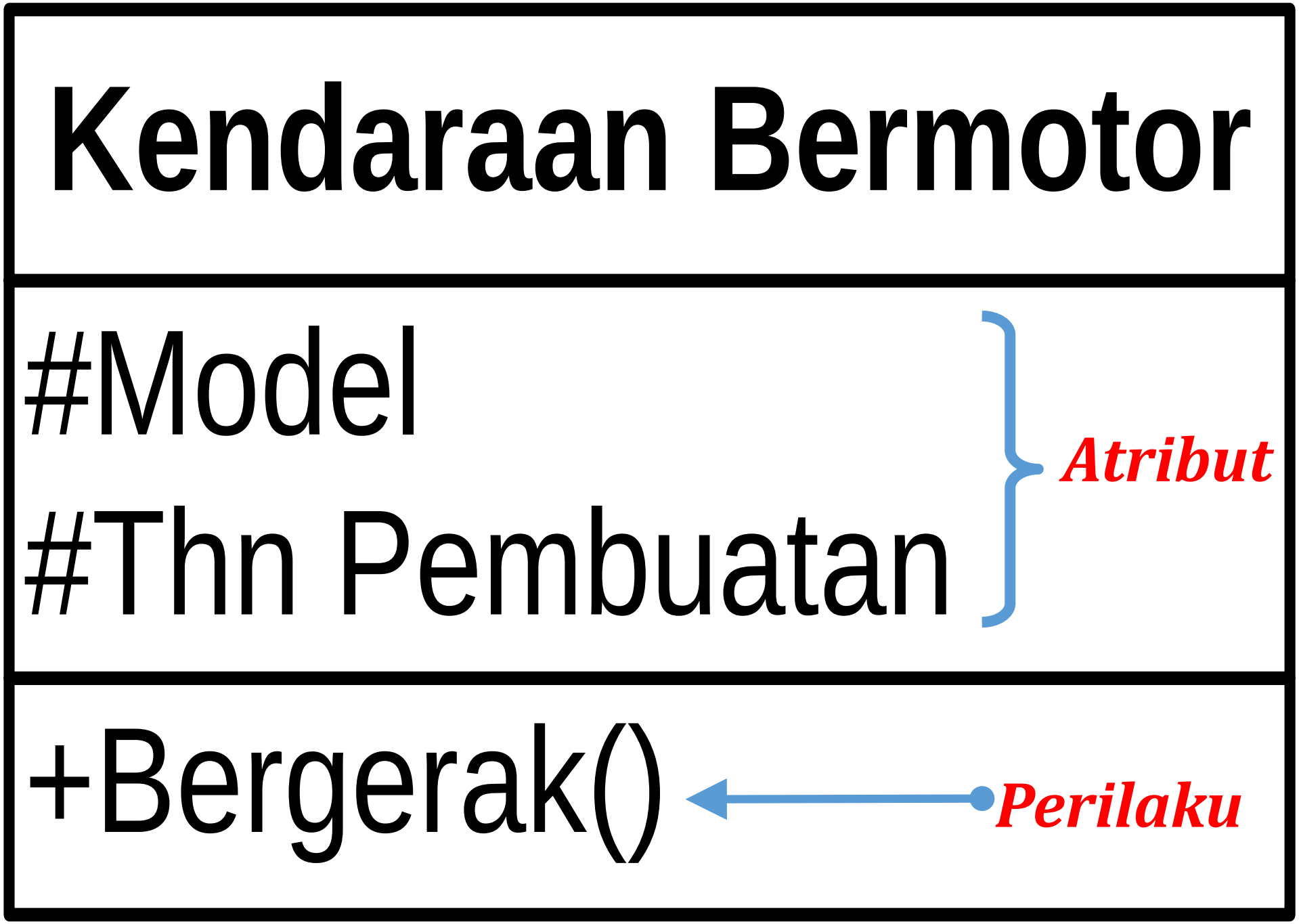
Kita mengatakan subclass “mewarisi” suatu superclass (atau juga bisa dikatakan sebuah subclass “turunan dari” suatu superclass)

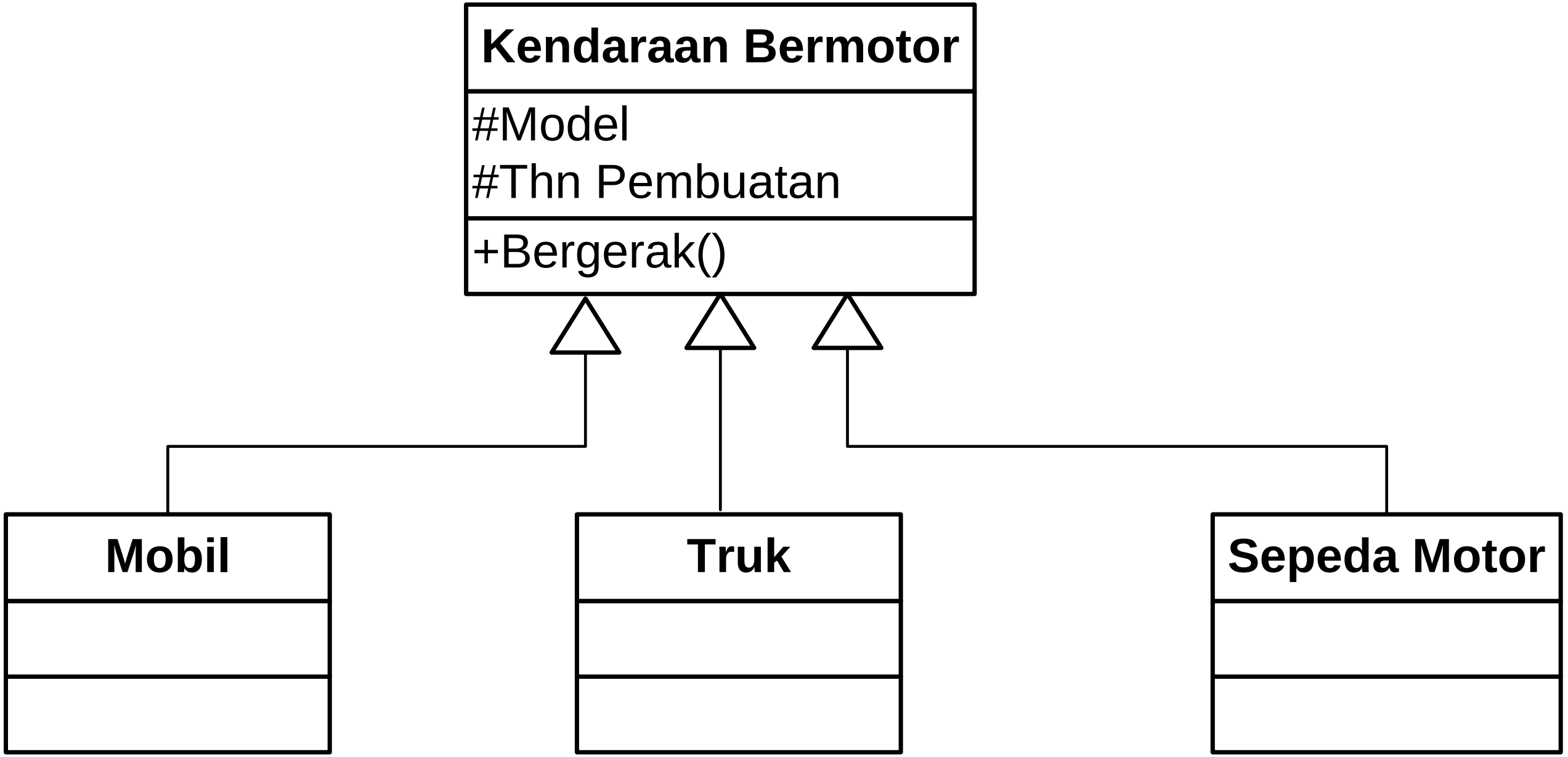
Object Relationship Diagram

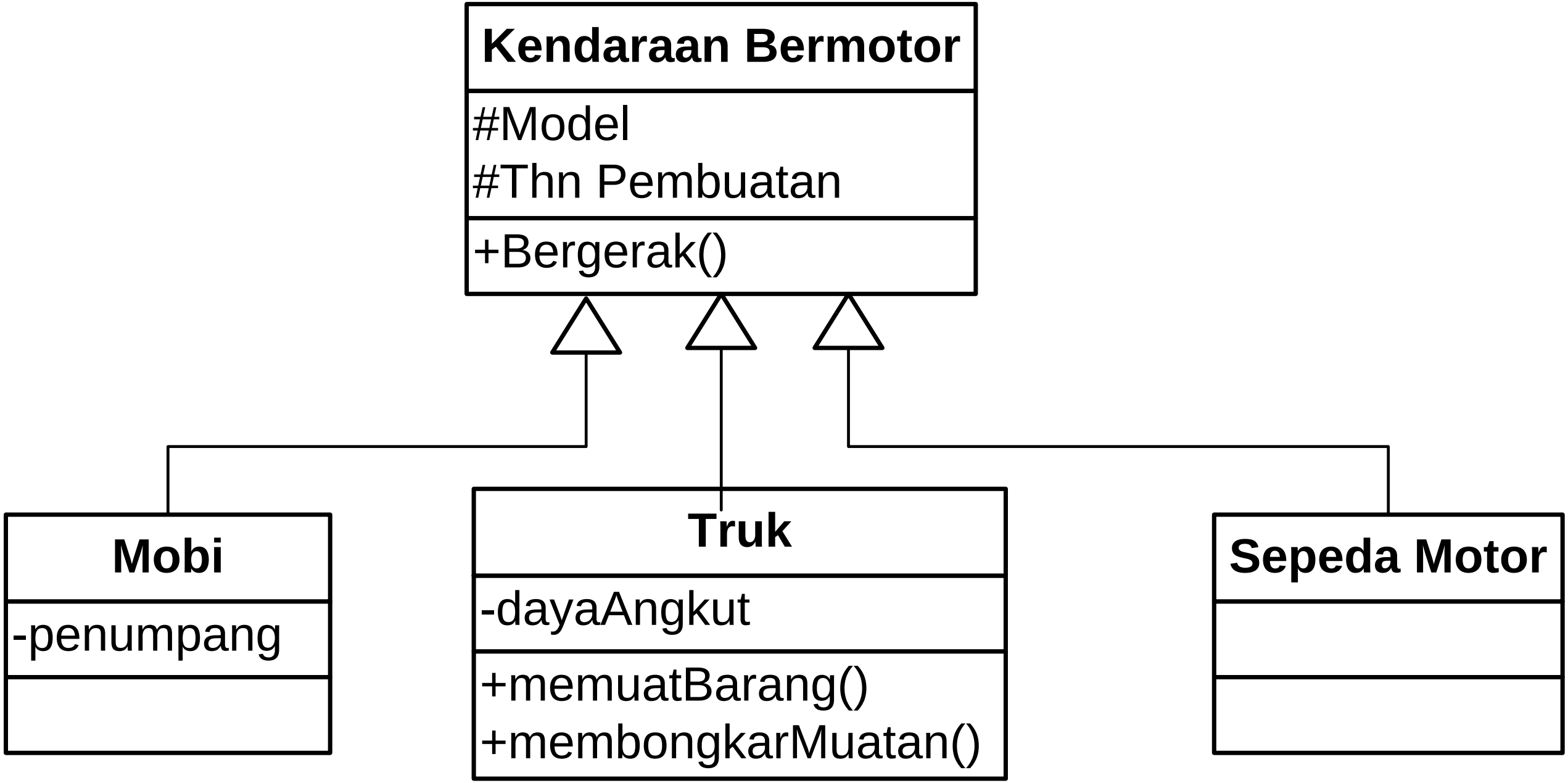


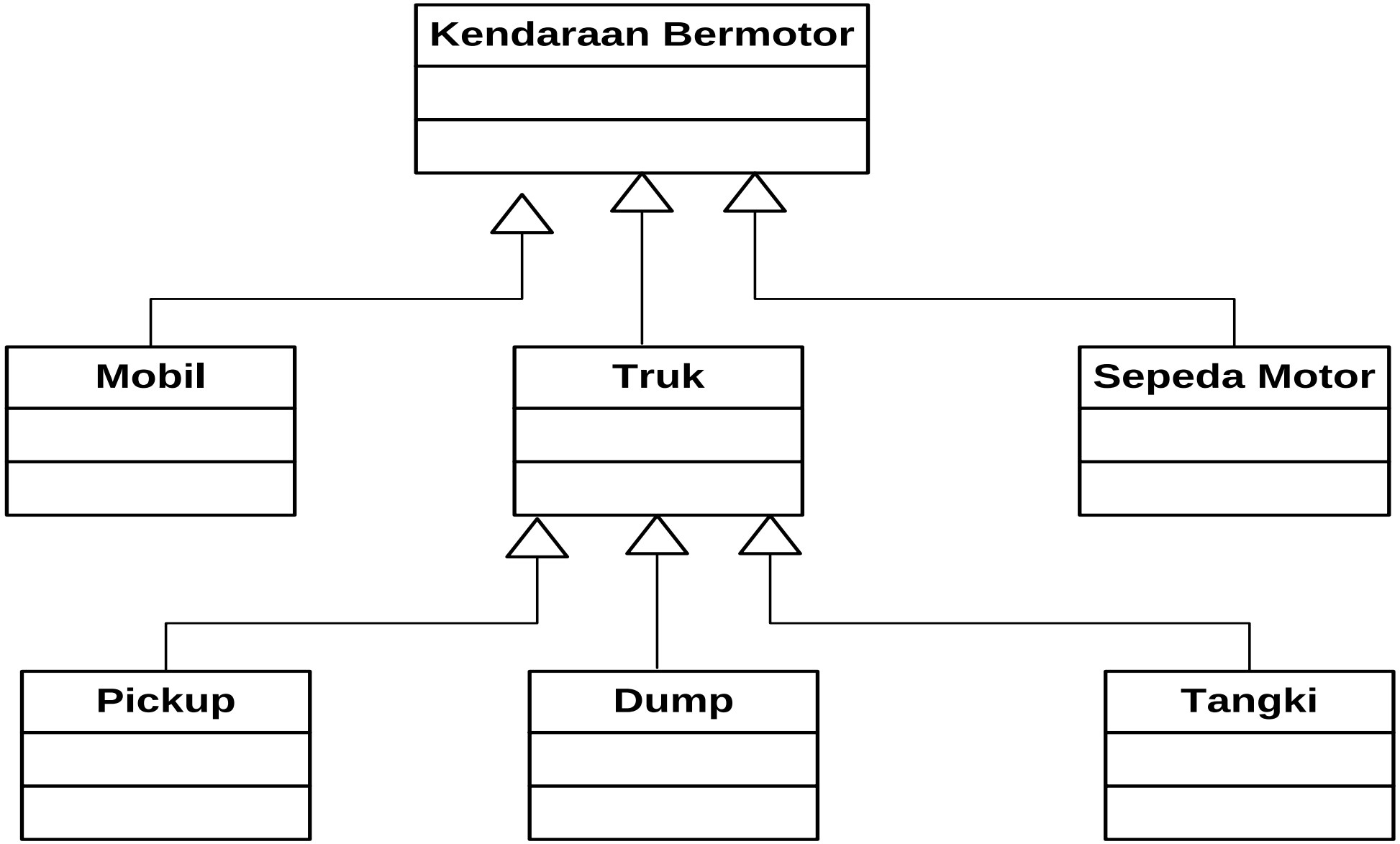
Superclass memiliki beberapa subclass




Kendaraan Bermotor
#Model
#Thn Pembuatan
Atribut
+Bergerak()
Perilaku


Kendaraan Bermotor
#Model
#Thn Pembuatan
+Bergerak()
Mobil
Truk
Sepeda Motor


Kendaraan Bermotor
#Model
#Thn Pembuatan
+Bergerak()
Mobi
-penumpang
Truk
-dayaAngkut
+memuatBarang()
+membongkarMuatan()
Sepeda Motor



Kelas turunan dapat diturunkan lagi menjadi kelas turunan lainnya

Membuat SubClass

class nama-subclass extends nama-superclass

```
class Goalkeeper extends SoccerPlayer {......}
class MyApplet extends java.applet.Applet {.....}
class Cylinder extends Circle {......}
```

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

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

    public int getA(){
        return a;
    }
}
```

SuperClass

```java
public class B extends A{
    private int b;

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

    public int getB(){
        return b;
    }
}
```

SubClass

```java
public class DemoKelasTurunan1 {
    public static void main (String [] args){
        B obj = new B ();

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

        System.out.println("Nilai a : "+obj.getA());
        System.out.println("Nilai b : "+obj.getB());
    }
}
```

DemoKelasTurunan1 > main >

utput - Pertemuan9 (run)

```
run:
Nilai a : 100
Nilai b : 200
BUILD SUCCESSFUL (total time: 1 second)
```

Hanya memanggil subclass, tapi method di superclass juga dapat dipanggil

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

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

    protected int getA(){
        return a;
    }
}
```

SuperClass

```java
public class B_protected extends A_protected {
    private int b;

    B_protected (int nilaiA, int nilaiB){
        //a = nilaiA;
        setA(nilaiA);
        b = nilaiB;
    }

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

SubClass

```java
public class C {
    private int c;
    public void setC(int nilai){
        //setA(10);
        c = nilai;
    }

    public int getC(){
        return c;
    }

    public void showC(){
        //System.out.println("Nilai a : "+getA());
        System.out.print("Nilai c : "+c);
    }
}
```

Method setA dan getA tidak dapat diakses oleh lingkungan luar

```java
public class DemoProtected1 {
    public static void main (String [] args){

        B_protected obj = new B_protected (40,50);

        obj.showData();

        obj.setA(100);
        System.out.println("Nilai a : " +obj.getA());
    }
}
```

Method setA di kelas induk dapat diakses oleh objek yang terbentuk dari kelas B(kelas turunan)

DemoProtected1 > main >

out - Pertemuan9 (run)

```
run:
Nilai a : 40
Nilai b : 50
Nilai a : 100
BUILD SUCCESSFUL (total time: 0 seconds)
```

Menggunakan Kata Kunci SUPER

```
super (daftar-parameter)
```

Constructor yang terdapat pada kelas induk dapat dipanggil dari kelas turunan menggunakan kata kunci **super**

```java
public class Kotak {
    protected double panjang;
    protected double lebar;
    protected double tinggi;

    Kotak(){
        panjang = lebar = tinggi = 0;
    }

    Kotak(int p, int l, int t){
        panjang = p;
        lebar = l;
        tinggi = t;
    }

    public double hitungvolume(){
        return (panjang*lebar*tinggi);
    }
}
```

```java
public class KotakPejal extends Kotak {
    private double berat;

    KotakPejal(int p, int l, int t, int b){
        super (p,l,t);

        berat = b;
    }

    public double getBerat(){
        return berat;
    }
}
```

Memanggil konstruktor kelas Kotak

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

        KotakPejal k = new KotakPejal (6,5,4,2);

        System.out.println ("Volume k : "+k.hitungvolume());
        System.out.println("Berat k  : "+k.getBerat());
    }
}
```

DemoSuper > main >

tput - Pertemuan9 (run)

```
run:
Volume k : 120.0
Berat k  : 2.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public class A_super {
    protected int a;
}
```

```java
public class B_super extends A_super{
    private int a;

    B_super(int nilai1, int nilai2){
        super.a = nilai1;
        a = nilai2;
    }

    public void tampilkanNilai(){
        System.out.println("Nilai a di dalam kelas A : "+super.a);
        System.out.println("Nilai a di dalam kelas B : "+a);
    }
}
```

Menimpa a di kelas A

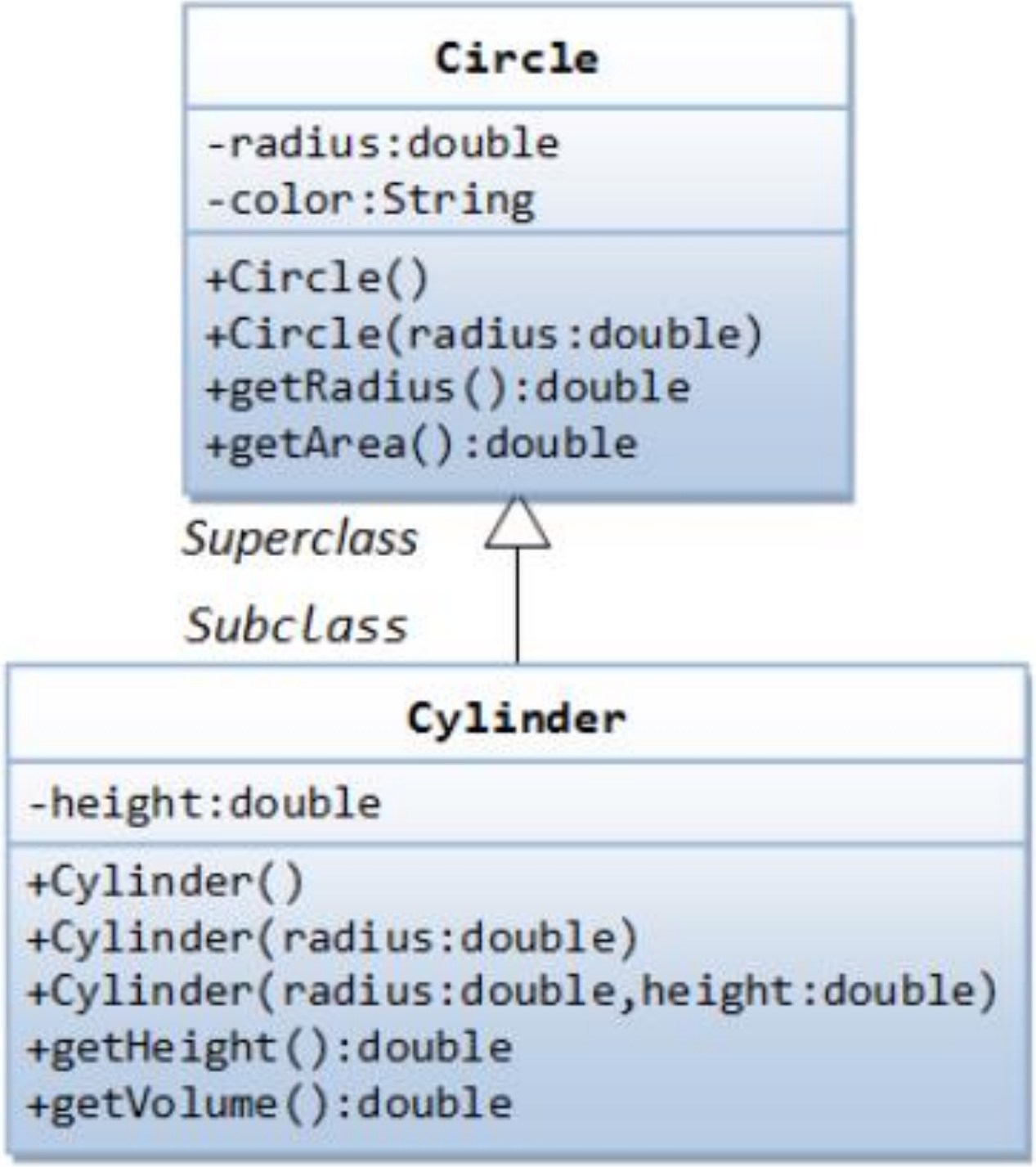
Mengakses anggota kelas induk (A_super)

```java
public class DemoSuper3 {
    public static void main (String [] args){

        B_super obj = new B_super (121, 212);
        obj.tampilkanNilai();
    }
}
```

DemoSuper3 > main > obj

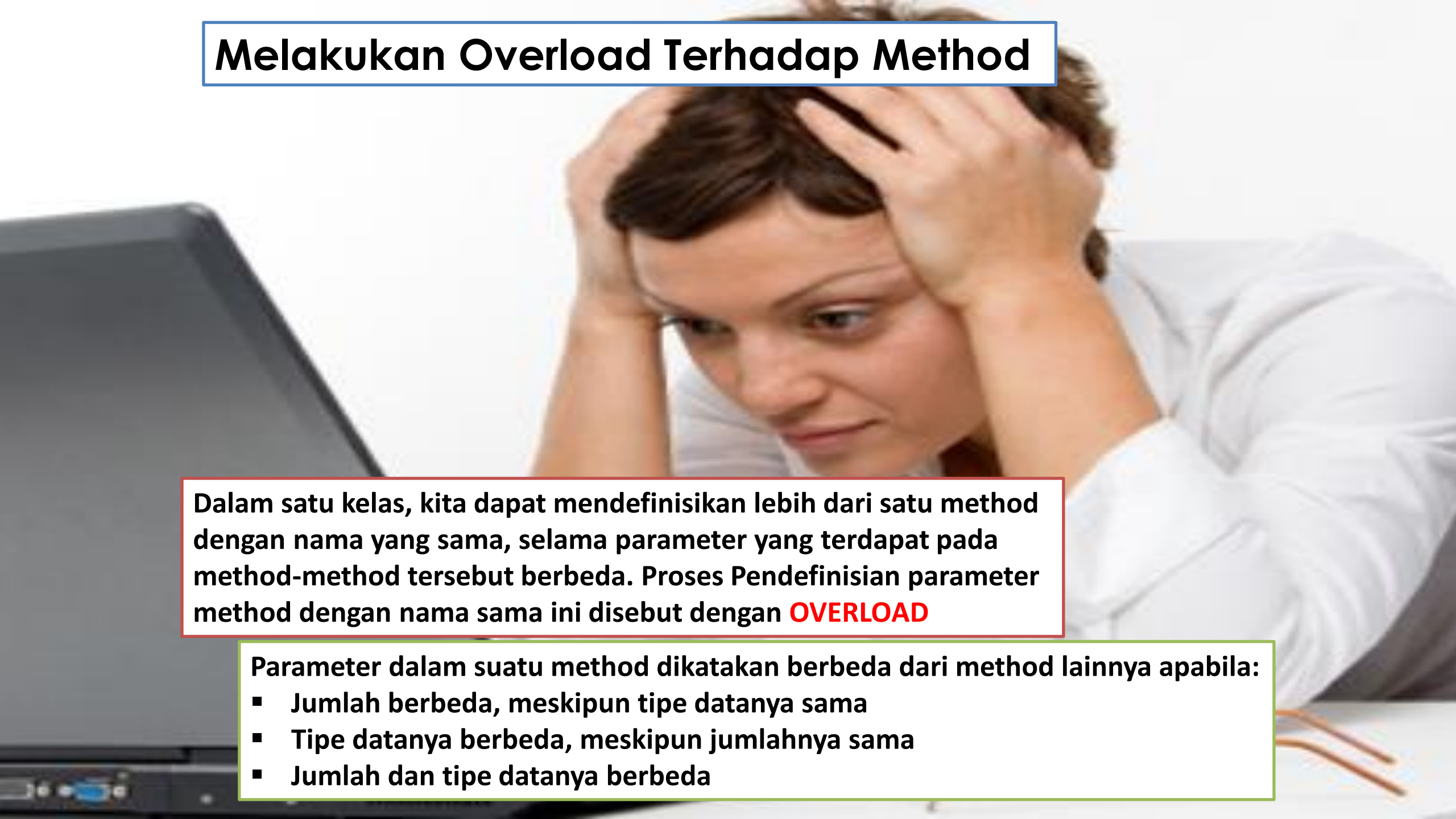
tput - Pertemuan9 (run)

```
run:
Nilai a di dalam kelas A : 121
Nilai a di dalam kelas B : 212
BUILD SUCCESSFUL (total time: 0 seconds)
```


Circle
-radius:double
-color:String
+Circle()
+Circle(radius:double)
+getRadius():double
+getArea():double
Superclass
Subclass
Cylinder
-height:double
+Cylinder()
+Cylinder(radius:double)
+Cylinder(radius:double,height:double)
+getHeight():double
+getVolume():double

```java
// Define Cylinder class, which is a subclass of Circle
public class Cylinder extends Circle {
   private double height;   // Private member variable

   public Cylinder() {      // constructor 1
      super();              // invoke superclass' constructor Circle()
      height = 1.0;
   }
   public Cylinder(double radius, double height) {  // Constructor 2
      super(radius);        // invoke superclass' constructor Circle(radius)
      this.height = height;
   }

   public double getHeight() {
      return height;
   }
   public void setHeight(double height) {
      this.height = height;
   }
   public double getVolume() {
      return getArea()*height;   // Use Circle's getArea()
   }
}
```

Melakukan Overload Terhadap Method



Dalam satu kelas, kita dapat mendefinisikan lebih dari satu method dengan nama yang sama, selama parameter yang terdapat pada method-method tersebut berbeda. Proses Pendefinisian parameter method dengan nama sama ini disebut dengan **OVERLOAD**

Parameter dalam suatu method dikatakan berbeda dari method lainnya apabila:

- Jumlah berbeda, meskipun tipe datanya sama
- Tipe datanya berbeda, meskipun jumlahnya sama
- Jumlah dan tipe datanya berbeda

```java
public class Pembagian {

    int bagi (int a, int b){
        return a/b;
    }

    double bagi (double a, double b){
        return a/b;
    }
}
```

Method yang dipanggil adalah method yang daftar parameternya ***paling sesuai*** *dengan tipe argumen yang dilewatkan*

```java
public class DemoOverload1 {
    public static void main (String [] args){

        Pembagian b = new Pembagian();

        int x = b.bagi (10,4);
        double y = b.bagi(10.0, 4.0);

        System.out.println("Hasil bagi tipe int = " +x);
        System.out.println("Hasil bagi tipe double = " +y);
    }
}
```

DemoOverload1 > main >

put - Pertemuan6 (run)

```
run:
Hasil bagi tipe int = 2
Hasil bagi tipe double = 2.5
BUILD SUCCESSFUL (total time: 1 second)
```

```java
public class myclass {

    void tulis (){
        System.out.println("Tanpa Parameter");
    }

    void tulis (double d){
        System.out.println("Satu parameter dengan nilai argumen : " +d);
    }
}
```

```java
public class DemoOverload2 {
    public static void main (String [] args){

        myclass c = new myclass ();

        c.tulis();
        c.tulis(4);
    }
}
```

DemoOverload2 > main >

put - Pertemuan6 (run)

```
run:
Tanpa Parameter
Satu parameter dengan nilai argumen : 4.0
BUILD SUCCESSFUL (total time: 1 second)
```

Dikonversi menjadi tipe double

```java
public class myclass2 {

    void tulis (int a, String s){
        System.out.println("int : " +a+ ", String : \""+s+"\"");
    }

    void tulis (String s, int a){
        System.out.println("String : \"" +s+ "\", int : "+a+"");
    }
}
```

Meskipun jumlah dan tipe dari parameter yang terdapat pada beberapa method sama, tapi jika urutannya berbeda, maka kedua method tersebut juga akan dianggap berbeda

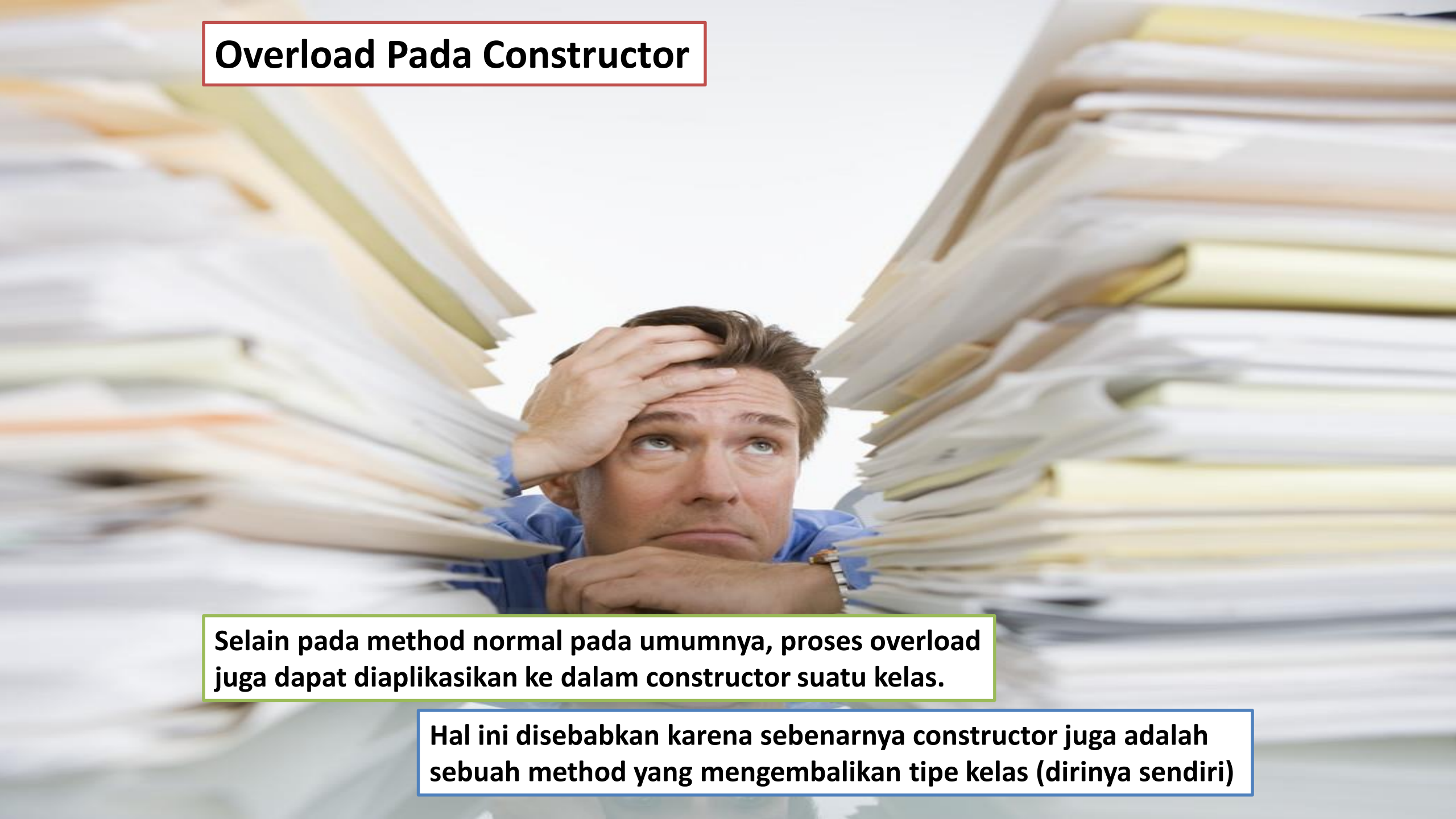
```java
public class DemoOverload3 {
    public static void main (String [] args){

        myclass2 c = new myclass2();

        c.tulis(4, "Contoh method pertama");
        c.tulis("Contoh method kedua", 8);
    }
}
```

DemoOverload3 > main >

put - Pertemuan6 (run)

```
run:
int : 4, String : "Contoh method pertama"
String : "Contoh method kedua", int : 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

Overload Pada Constructor



Selain pada method normal pada umumnya, proses overload juga dapat diaplikasikan ke dalam constructor suatu kelas.

Hal ini disebabkan karena sebenarnya constructor juga adalah sebuah method yang mengembalikan tipe kelas (dirinya sendiri)

```java
public class Kotak {
    double panjang;
    double lebar;
    double tinggi;

    Kotak(){
        panjang = 0;
        lebar = 0;
        tinggi = 0;
    }

    Kotak (double p, double l, double t){
        panjang = p;
        lebar = l;
        tinggi = t;
    }
    Kotak (double sisi){
        panjang = lebar = tinggi = sisi;
    }

    double hitungVolume(){
        return (panjang*lebar*tinggi);
    }
}
```

Tanpa parameter

Tiga parameter

Satu parameter

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

        Kotak k1, k2, k3;

        k1 = new Kotak();
        k2 = new Kotak(10);
        k3 = new Kotak(4,3,2);

        System.out.println("Volume k1 = " +k1.hitungVolume());
        System.out.println("Volume k1 = " +k2.hitungVolume());
        System.out.println("Volume k1 = " +k3.hitungVolume());
    }
}
```

DemoOverloadConstructor > main >

put - Pertemuan6 (run)

```
run:
Volume k1 = 0.0
Volume k1 = 1000.0
Volume k1 = 24.0
BUILD SUCCESSFUL (total time: 1 second)
```

Objek Sebagai Parameter

Dalam Java, objek juga dapat berperan sebagai parameter dari sebuah method

```java
public class Kotak2 {
    double panjang;
    double lebar;
    double tinggi;

    Kotak2 (double p, double l, double t){
        panjang = p;
        lebar = l;
        tinggi = t;
    }

    double hitungVolume(){
        return (panjang*lebar*tinggi);
    }

    boolean sama (Kotak2 k){
        if ((k.panjang == this.panjang)&&(k.lebar == this.lebar)&&(k.tinggi == this.tinggi)){
            return true;
        }
        else {
            return false;
        }
    }
}
```

```java
public class DemoParamObjek1 {
    public static void main (String [] args){

        Kotak2 k1,k2,k3,k4;

        k1 = new Kotak2(4,3,2);
        k2 = new Kotak2(6,5,4);
        k3 = new Kotak2(4,3,2);
        k4 = new Kotak2(6,5,4);

        System.out.println("k1 == k2 : " +k1.sama(k2));
        System.out.println("k1 == k2 : " +k1.sama(k3));
        System.out.println("k1 == k2 : " +k2.sama(k4));
    }
}
```

put - Pertemuan6 (run)

```
run:
k1 == k2 : false
k1 == k2 : true
k1 == k2 : true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method dengan parameter objek k

Objek yang dilewatkan pada umumnya digunakan untuk proses inisialisasi nilai dari objek baru yang akan dibentuk

```java
public class Kotak3 {
    double panjang;
    double lebar;
    double tinggi;

    Kotak3(double p, double l, double t){
        panjang = p;
        lebar = l;
        tinggi = t;
    }

    Kotak3 (Kotak3 k){
        this.panjang = k.panjang;
        this.lebar = k.lebar;
        this.tinggi = k.tinggi;
    }

    double hitungVolume (){
        return (panjang*lebar*tinggi);
    }

    boolean sama (Kotak3 k){
        if ((k.panjang == this.panjang)&&(k.lebar == this.lebar)&&(k.tinggi == this.tinggi)){
            return true;
        }
        else {
            return false;
        }
    }
}
```

```java
public class DemoParamObjek2 {
    public static void main (String [] args){

        Kotak3 k1,k2;

        k1 = new Kotak3 (4,3,2);
        k2 = new Kotak3 (k1);

        System.out.println("k1 == k2 : " +k1.sama(k2));
        System.out.println("Volume k1 = " +k1.hitungVolume());
        System.out.println("Volume k2 = " +k1.hitungVolume());
    }
}
```

DemoParamObjek2 > main >

put - Pertemuan6 (run)

```
run:
k1 == k2 : true
Volume k1 = 24.0
Volume k2 = 24.0
BUILD SUCCESSFUL (total time: 0 seconds)
```