

PERTEMUAN 13

PERANCANGAN BERORIENTASI OBJEK UNTUK MULTI KELAS

POKOK BAHASAN

1. Pendahuluan
2. Hubungan Antar Kelas
3. Inheritansi
4. Polimorfisme

PENDAHULUAN

- Keunggulan utama bahasa pemrograman berorientasi objek adalah pada kemampuan dalam membangun program yang besar.
- Dalam merancang program yang menggunakan beberapa kelas perlu mempertimbangkan hubungan antar kelas.



NOTASI

- Pendekatan perancangan berorientasi objek dan pemrograman berorientasi objek telah menjadi metodologi yang stabil.
- Notasi yang digunakan untuk perancangan berorientasi objek adalah UML (Unified Modelling Language) dari Rumbaugh, Booch, dan jacobson.

HUBUNGAN ANTAR KELAS

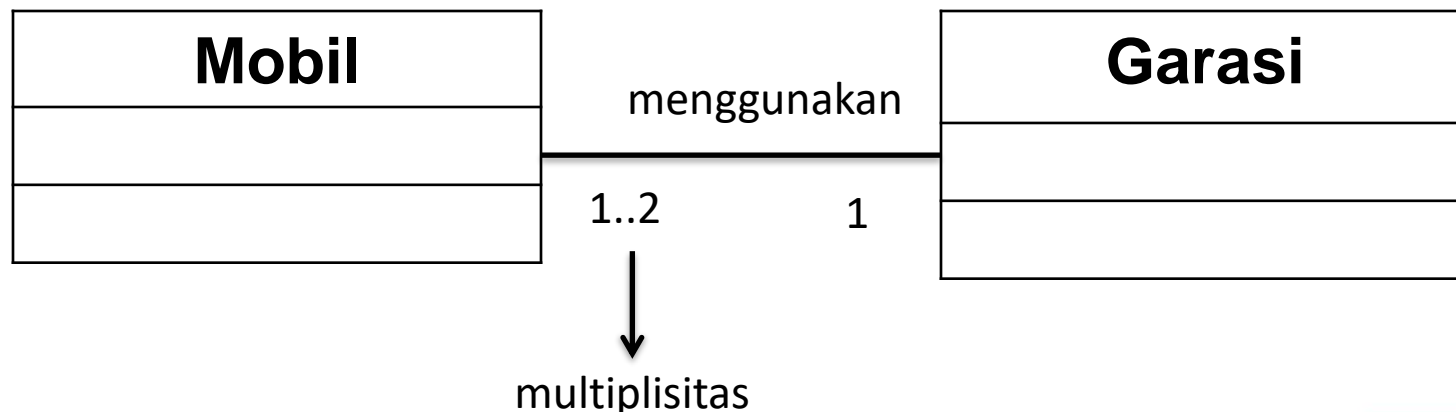
1. Hubungan yang paling sederhana adalah hubungan antar dua kelas yang independen satu sama lain. Sebuah kelas dapat menggunakan layanan yang lain disebut dengan asosiasi.
2. Sebuah kelas mungkin dibuat oleh kelas lain atau mengandung kelas lain yang merupakan bagian dari dirinya sendiri. Hubungan kolektif dapat berupa agregasi atau komposisi.
3. Kelas memungkinkan mewarisi seluruh atribut dan operasi dari kelas induk, memiliki nama yang unik, atribut dan operasi yang berbeda dengan kelas induk. Bentuk hubungan antara kelas induk dan anak adalah generalisasi.

ASOSIASI

- Asosiasi menggambarkan interaksi yang mungkin terjadi antara suatu objek dengan objek yang lain.
- Asosiasi memungkinkan suatu kelas untuk menggunakan atau mengetahui atribut atau operasi yang dimiliki oleh kelas lain.
- Ada 2 jenis asosiasi :
 - Asosiasi dua arah  atau
 - Asosiasi satu arah (pasif) 

ASOSIASI (lanjutan)

- Contoh : kelas mobil dan kelas garasi merupakan kelas independen. Mobil kadang menggunakan layanan kelas garasi seperti parkir. Objek yang diinstansiasi dari kedua kelas dapat saling berinteraksi melalui pengiriman atau penerimaan pesan

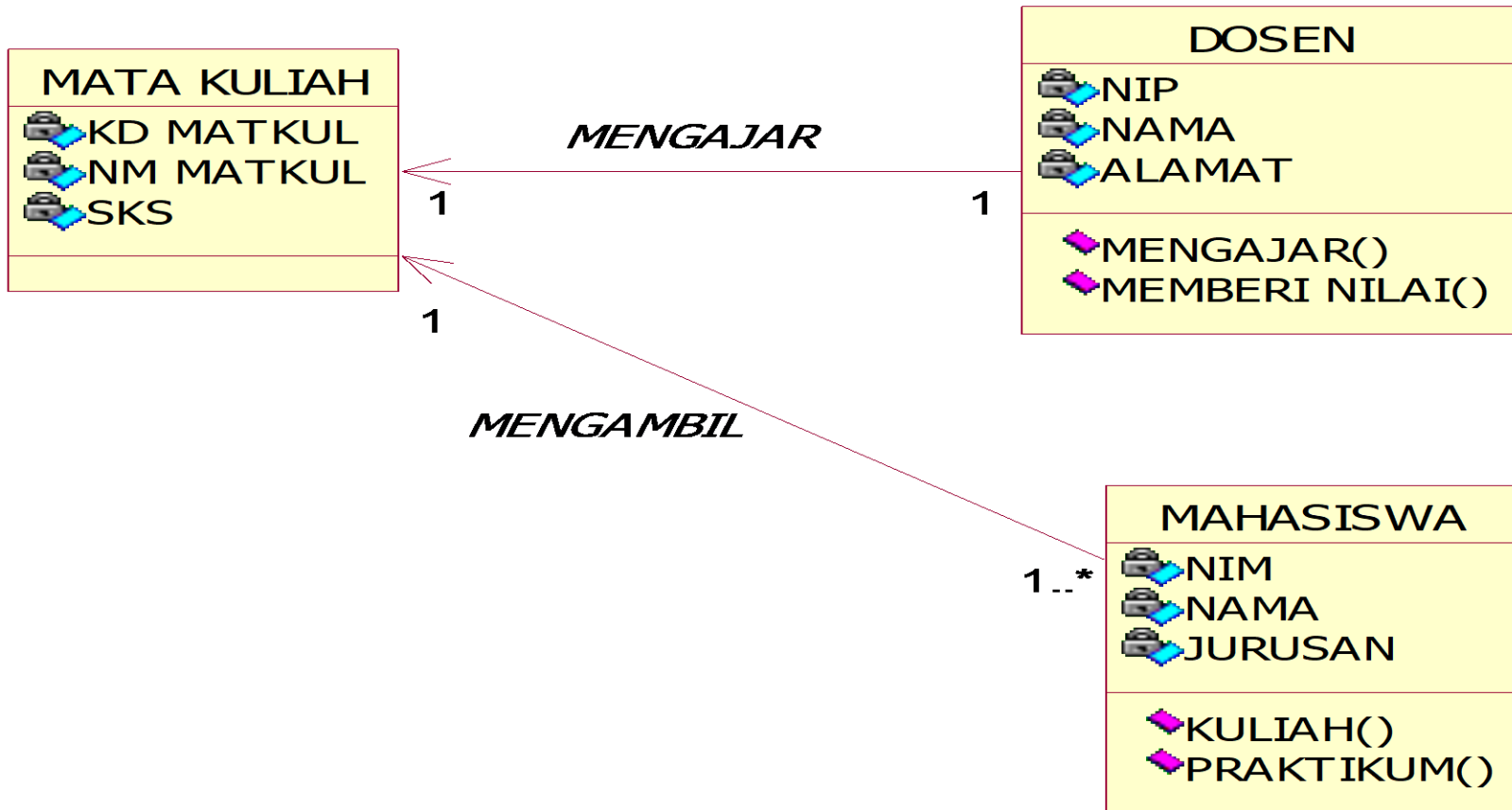


ASOSIASI (lanjutan)

- Pada asosiasi terdapat multiplisitas.
- Multiplisitas adalah jumlah banyaknya obyek sebuah kelas yang berelasi dengan sebuah obyek lain pada kelas lain yang berasosiasi dengan kelas tersebut.

Tipe	Notasi UML	Keterangan
Exactly	1 or blank	Seorang Karyawan bekerja pada satu dan hanya satu departemen
Zero or 1	0..1	Seorang Karyawan memiliki satu suami/istri atau tidak punya suami/istri
Zero or More	0..* or *	Customer dapat tidak melakukan pembayaran sampai beberapa kali
1 or More	1..*	Universitas menawarkan paling sedikit 1 matakuliah sampai beberapa matakuliah
Specific range	7..9	Tim memiliki pertandingan terjadwal sebanyak 7, 8, atau 9 pertandingan

ASOSIASI (lanjutan)

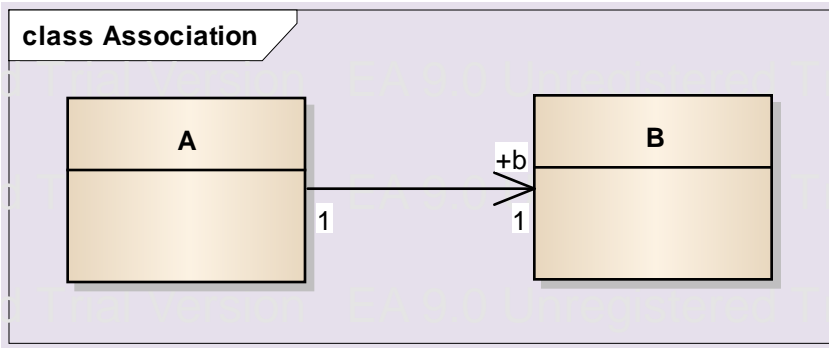


ASSOCIATIONS


```
public class A {  
    public B b;
```

```
    public A() {  
        }  
    }
```

```
public class B {  
    public B() {  
        }  
    }
```



AGREGASI

- Agregasi adalah hubungan suatu kelas yang merupakan bagian dari kelas lain namun bersifat tidak wajib.
- Simbol : 




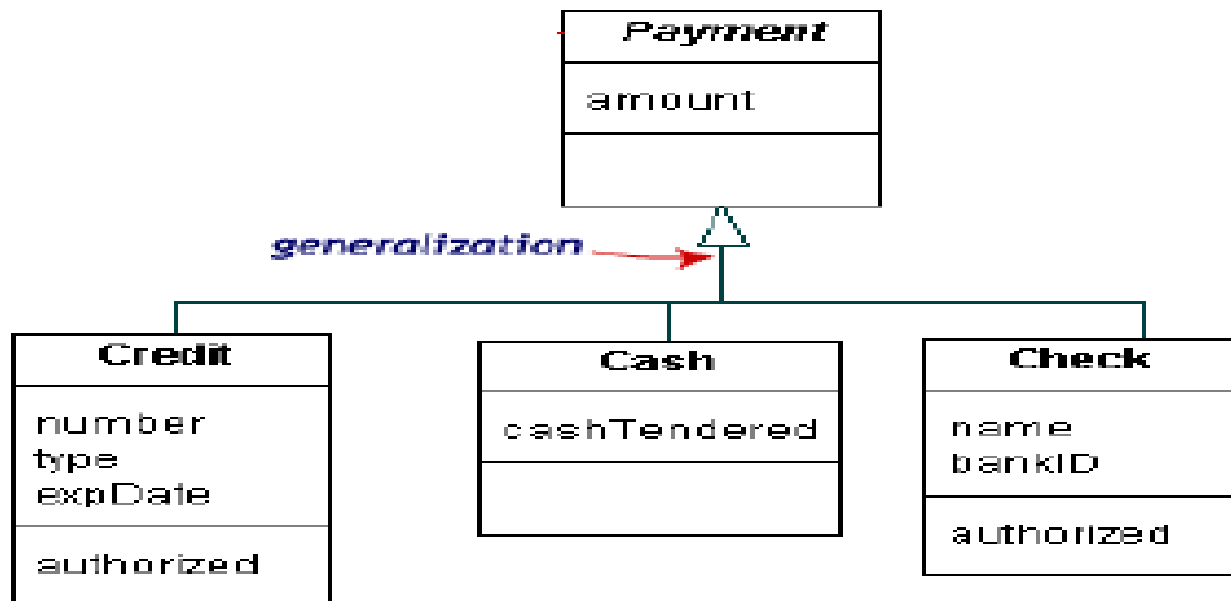
KOMPOSISI

- Komposisi adalah hubungan suatu kelas yang merupakan bagian yang wajib dari kelas lain.
- Simbol :

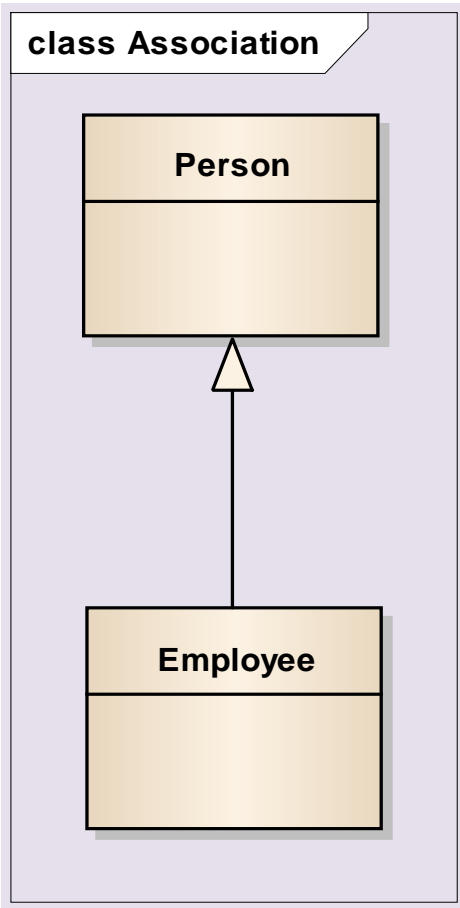


GENERALISASI

- Generalisasi diperlukan untuk memperlihatkan hubungan pewarisan antar objek atau kelas.
- Simbol : 



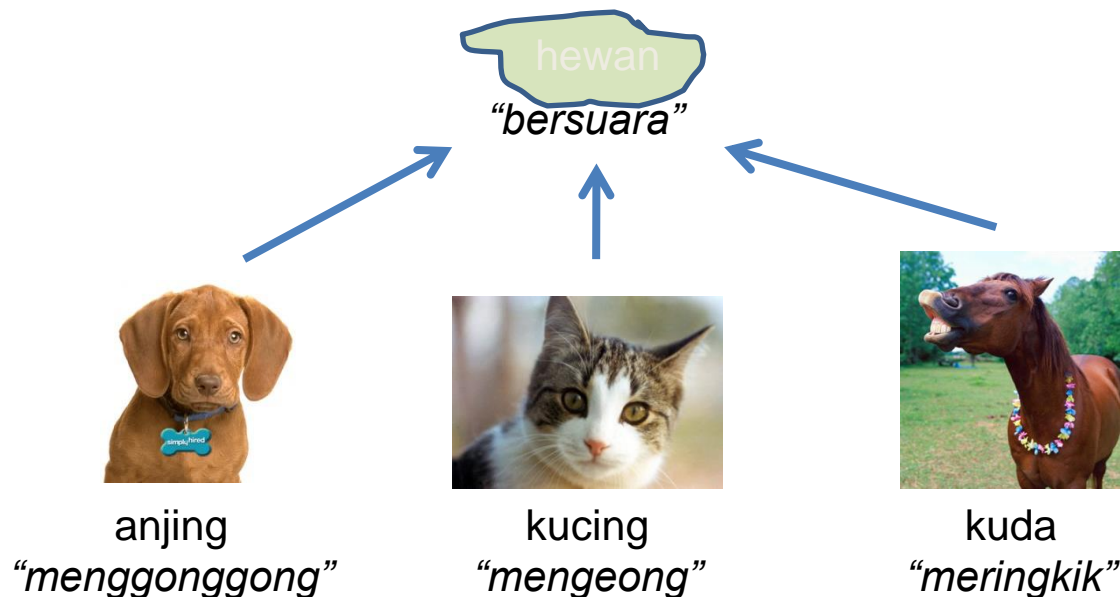
GENERALISASI



```
public class Person {  
  
    public Person() {  
  
    }  
  
}  
  
public class Employee  
    extends Person {  
  
    public Employee() {  
  
    }  
  
}
```

POLYMORPHISM

- Polymorphism adalah kemampuan untuk mempunyai beberapa bentuk yang berbeda.
- Polymorphism diimplementasikan dengan mekanisme inheritance dan overriding



CONTOH POLYMORPHISM

```
class Animal {  
    public void animalSound() {  
        System.out.println("Binatang bersuara");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: bark bark");  
    }  
}  
  
class Cat extends Animal {  
    public void animalSound() {  
        System.out.println("The cat says: Meaowww");  
    }  
}  
  
class Horse extends Animal {  
    public void animalSound() {  
        System.out.println("The Horse says: Hieeee..");  
    }  
}
```

Polymorphism menggunakan keyword “**extend**”


```
public class TestAnimalSound {  
  
    public static void main(String args[]) {  
        Animal a = new Animal(); // Animal reference and object  
        Animal b = new Dog(); // Animal reference but Dog object  
  
        a.animalSound(); // runs the method in Animal class  
        b.animalSound(); // runs the method in Dog class  
    }  
}
```

Output

Binatang bersuara
The Dog Says: bark bark

POLYMORPHISMS

OVERLOADING

Method Overloading adalah sebuah kemampuan yang membolehkan sebuah class mempunyai 2 atau lebih method dengan nama yang sama, yang membedakan adalah parameternya.

Pada method overloading perbedaan parameter mencakup :

- Jumlah parameter
- Tipe data dari parameter
- Urutan dari tipe data parameter

Method Overloading juga dikenal dengan sebutan Static Polymorphism. Berikut ini contoh Class yang melakukan Overloading.

* PHP tidak mendukung Overloading

OVERRIDING

Method overriding merupakan **method yang parent class yang ditulis kembali oleh subclass**. Aturan dari method overriding pada Java :

- Parameter yang terdapat pada method overriding di subclass harus sama dengan parameter yang terdapat pada parent class.
- Aturan hak akses (visibility), hak akses method overriding di subclass tidak boleh lebih ketat di bandingkan dengan hak akses method pada parent class.

OVERLOADING

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

CONTOH OVERLOADING

```
public class Calculation {  
    void sum(int a,int b) {  
        System.out.println(a+b);  
    }  
    void sum(int a,int b,int c) {  
        System.out.println(a+b+c);  
    }  
  
    public static void main(String args[]){  
        Calculation cal = new Calculation();  
        cal.sum(20,30,60); // perhatikan nama class sama  
        cal.sum(20,20); //beda jumlah yang dijumlahkan tapi nama sama  
    }  
}
```

```
public class ContohOverloading {
    public void jumlah (int a, int b){ System.out.println("Jumlah 2
    angka =" + (a + b));
}
```

```
//overloading perbedaan jumlah parameter
public void jumlah (int a, int b, int c){
    System.out.println("Jumlah 3 angka =" + (a + b + c));
}
```

```
//overloading perbedaan tipe data parameter
public void jumlah(double a, int b){
    System.out.println("Jumlah 2 angka (double+int) =" + (a + b));
}
```

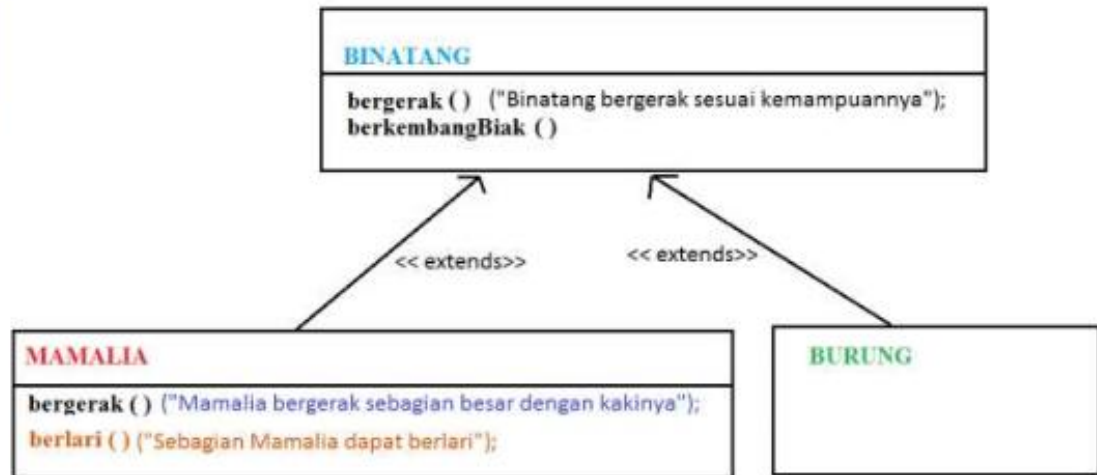
```
//overloading perbedaan urutan tipe data parameter public void
jumlah (int b, double a){
    System.out.println("Jumlah 2 angka (int+double) =" + (a + b));
}
}
```

Contoh overloading di main program, **co** adalah object baru dari bentuk class **ContohOverloading** yang sudah ada

```
public class PenggunaanOverloading {
    public static void main(String[] args) {
        ContohOverloading co = new ContohOverloading();
        co.jumlah(83,32);
        co.jumlah(34,454,432);
        co.jumlah(34.43,34);
        co.jumlah(28,33.23);
    }
}
```


OVERRIDING

Extends = Turunan; anak (MAMALIA) memiliki sifat parent (BINATANG)



```

public class Binatang {
    public void bergerak(){
        System.out.println("Binatang bergerak sesuai kemampuannya");
    }

    public void berkembangBiak(){
        System.out.println("Binatang berkembang biak sesuai kemampuannya");
    }
}

public class Mamalia extends Binatang {
    //overriding method parent class
    public void bergerak(){
        System.out.println("Mamalia bergerak sebagian besar dengan kakinya");
    }
    public void berlari(){
        System.out.println("Sebagian Mamalia dapat berlari");
    }
}
    
```

```
public class PenggunaanOverriding {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Binatang b = new Binatang();  
        Mamalia m = new Mamalia();  
        Binatang bm = new Mamalia();  
  
        b.bergerak();    // Binatang bergerak sesuai kemampuannya  
        m.bergerak();    // Mamalia bergerak sebagian besar dengan kakinya  
        bm.bergerak();   // Binatang bergerak sesuai kemampuannya + Mamalia bergerak sebagian besar dengan kakinya  
        bm.berkembangBiak(); // Binatang berkembang biak sesuai kemampuannya  
    }  
}
```

bm override object dimana object tersebut memiliki sifat dari parent object
Child object = mamalia
Parent object = binatang

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.

CONTOH OVERRIDING

```
class Animal {  
    public void animalSound() {  
        System.out.println("Binatang bersuara");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        super.animalSound(); // invokes the super class method  
        System.out.println("The Dog says: bark bark");  
    }  
}  
  
public class TestAnimalSound {  
  
    public static void main(String args[]) {  
        Animal b = new Dog(); // Animal reference but Dog object  
        b.animalSound(); // runs the method in Dog class  
    }  
}
```

Output

```
Binatang bersuara  
The Dog says: bark bark
```

Overriding menggunakan keyword **“super”**

Contoh Overriding Python

```
1 class Kendaraan: #kelas Induk
2     def berjalan(self):
3         print('berjalan..')
4
5 class Mobil(Kendaraan): #kelas anak
6     def berjalan(self): #fungsi sama dengan induk
7         print('Berjalan dengan cepat..') #override fungsi induk
8
9 sepeda = Kendaraan() #create object 'sepeda' dari class induk
10 sedan = Mobil() #create object 'sedan' dari class anak
11
12 sepeda.berjalan() #panggil fungsi dari class
13 sedan.berjalan() #panggil fungsi dari class
14
15 #fungsi yang dipanggil sama, tetapi hasil luaran fungsi berbeda
16
```

Hasil running code program →

```
berjalan..
Berjalan dengan cepat..
> |
```