

Pewarisan

{inheritance}

/*

PERTEMUAN 11

KULIAH FUNDAMEN PENGEMBANGAN APLIKASI

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

UNIVERSITAS ISLAM INDONESIA

*/



Object Oriented Programming (OOP)

Empat konsep dasar Object Oriented Programming (OOP):

1. Encapsulation
- 2. Inheritance (Pewarisan)**
3. Polymorphism
4. Abstraction

Guna Ulang Kode Program

Setelah mengenal beberapa konsep dasar dalam pemrograman berorientasi objek, bisakah kita sebutkan cara memanfaatkan baris atau blok kode program untuk digunakan di bagian program yang lain?

- Salin-tempel (X) – *tidak disarankan*
- Memanggil method: fungsi dan prosedur (*pada skala kecil, masih dalam satu kelas*)
- Hubungan antar kelas: *association, composition, aggregation.*
- Ada lagi?

**Banyak cara yang
dapat digunakan
untuk ‘*guna ulang*’
kode program
dalam paradigma
berorientasi objek**



Diagram Kelas Jenis Mobil

MobilOffroad
+ merk: String + noMesin: String + kecepatanMaksimum: Integer
+ berjalan() + mengerem() + membunyikanKlakson() + melewatiRintanganAlam()

MobilKeluarga
+ merk: String + noMesin: String + kecepatanMaksimum: Integer
+ berjalan() + mengerem() + membunyikanKlakson() + menampilkanHiburan()

Humvee
+ merk: String + noMesin: String + jenisAltileri: String
+ berjalan() + mengerem() + menembakkanMisil() + menabrakRintangan()

* Humvee: jenis kendaraan militer

Keterangan:

- Diagram kelas dari tiga objek konseptual yang berbeda
- Bagaimana implementasi ketiga diagram kelas tersebut dalam bahasa pemrograman Java?

```

public class MobilOffroad {
    String merk;
    String noMesin;
    int kecepatanMaksimum;

    void mengerem(){}

    void berjalan(){}

    void membunyikanKlakson(){}

    void melewatiRintanganAlam(){}
}

```

```

public class Humvee {

    String merk;
    String noMesin;
    String jenisAltileri;

    void mengerem(){}

    void berjalan(){}

    void menembakkanMisil(){}

    void menabrakRintangan(){}
}

```

```

public class MobilKeluarga {

    String merk;
    String noMesin;
    int kecepatanMaksimum;

    void mengerem(){}

    void berjalan(){}

    void membunyikanKlakson(){}

    void menampilkanHiburan(){}
}

```

Implementasi diagram
kelas pada program
java

MobilOffroad	MobilKeluarga	Humvee
+ merk: String + noMesin: String	+ merk: String + noMesin: String	+ merk: String + noMesin: String
+ kecepatanMaksimum: Integer	+ kecepatanMaksimum: Integer	+ jenisAltileri: String
+ berjalan() + mengerem()	+ berjalan() + mengerem()	+ berjalan() + mengerem()
+ membunyikanKlakson() + melewatiRintanganAlam()	+ membunyikanKlakson() + menampilkanHiburan()	+ menembakkanMisil() + menabrakRintangan()

Perhatikan:

1. Ada atribut yang sama pada ketiga kelas: **merk** dan **noMesin**
2. Ada method yang sama pada ketiga kelas: **berjalan()** dan **mengerem()**

Jika method **memiliki implementasi yang sama**, apa yang akan kita lakukan?

Apakah menulis *method* di salah satu kelas, kemudian **salin-tempel** ke kelas yang lain?

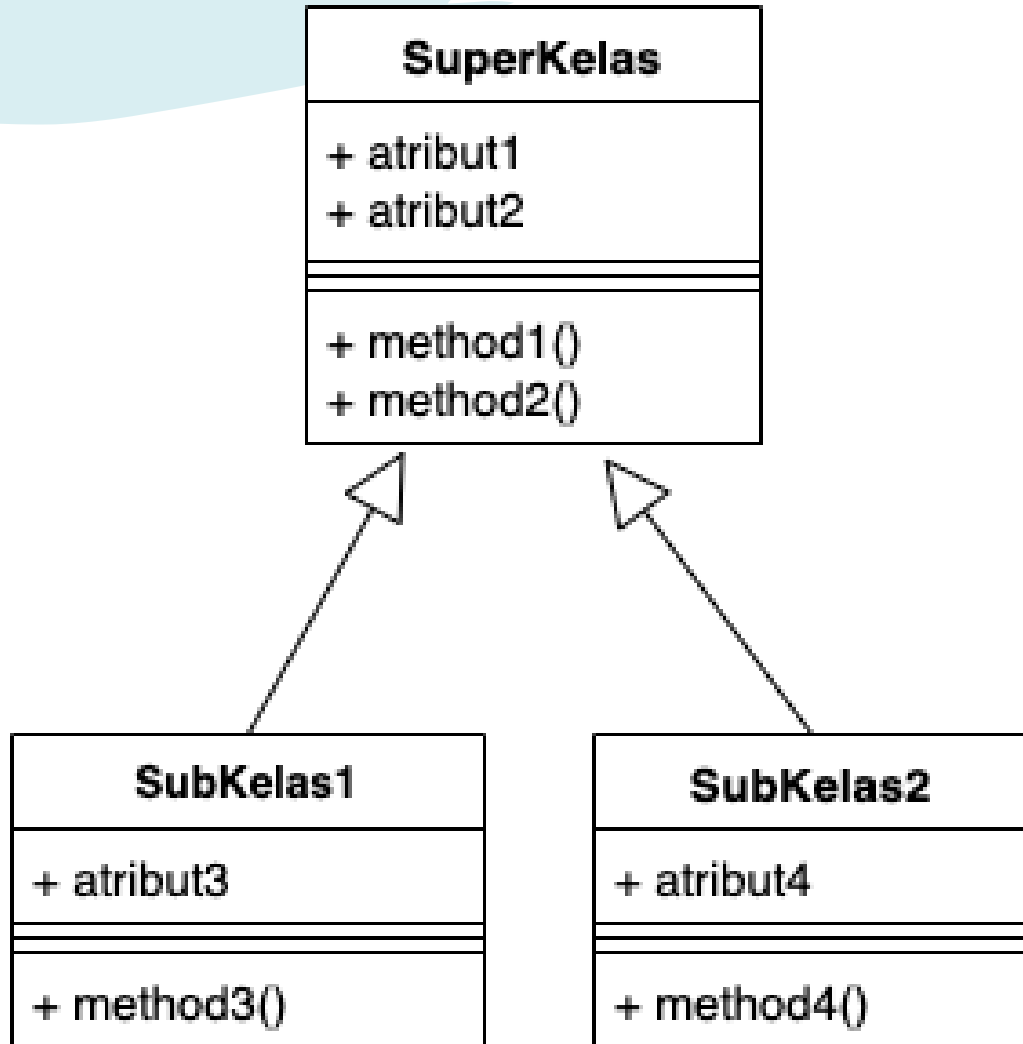
Pewarisan (inheritance)

Definisi:

- Proses **mewariskan** class member (**atribut** dan/atau **method**) dari satu kelas ke kelas lainnya.
 - Seperti konsep orang tua yang mewariskan sifat dan harta waris ke anaknya.
- Kelas yang **mewariskan** member disebut dengan **superclass/kelas induk/kelas orang tua**
- Kelas yang menerima atau **mewarisi** class member disebut **subclass/kelas anak**
- Pewarisan merupakan **salah satu prinsip penting** dalam paradigma berorientasi objek



Diagram kelas pewarisan



SuperKelas memiliki:

- **Dua atribut:** atribut1, atribut 2
- **Dua method:** method1 (), method2()

SubKelas1 memiliki:

- **Tiga atribut:** atribut1, atribut2, atribut3
- **Tiga method:** method1 (), method2(), method3()

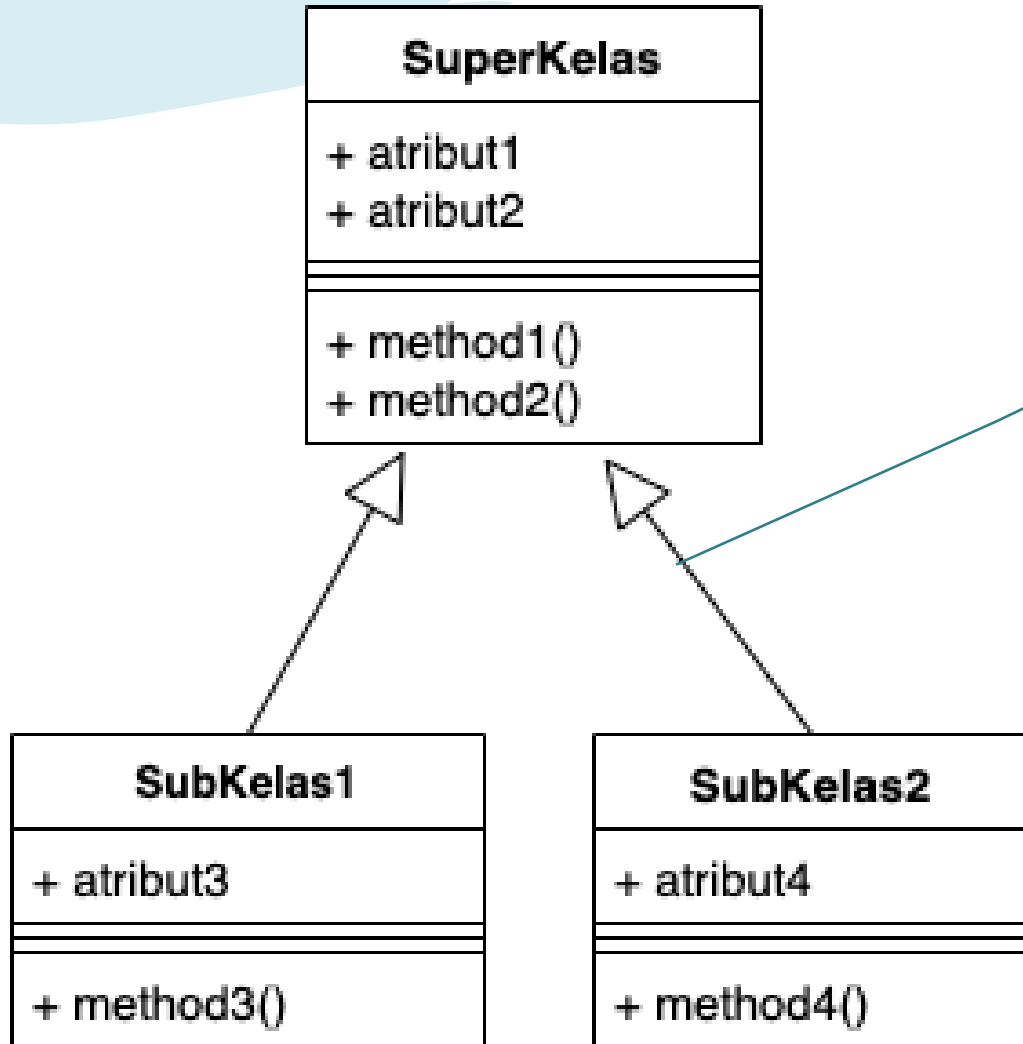
SubKelas2 memiliki:

- **Tiga atribut:** atribut1, atribut2, atribut4
- **Tiga method:** method1 (), method2(), method4()

Atribut1, atribut2, method1(), dan method2() pada **SubKelas1** dan **SubKelas2** didapatkan dari hasil **pewarisan SuperKelas**.

Subkelas **bisa menambahkan properti lain** baik atribut maupun method.

Diagram kelas pewarisan



Perhatikan simbol hubungan antara **superkelas** dengan **subkelasnya**.

Dihubungkan dengan anak panah (tanpa isi) yang mengarah ke **superkelas**.

Jangan sampai terbalik!!

Aturan pewarisan atribut dan method pada Java

Aturan ini berkaitan dengan ***access modifier*** (hak akses)

- Jika atribut atau method pada ***superclass*** memiliki hak akses
 - **PRIVATE** maka tidak bisa diwariskan pada ***subclass*** di manapun keberadaannya.
 - **DEFAULT** maka hanya diwariskan pada ***subclass*** yang berada **HANYA pada package yang sama**, tidak diwariskan pada ***subclass*** yang berbeda package/folder
 - **PROTECTED** dan **PUBLIC** maka dapat diwariskan pada semua ***subclass*** yang ada di **semua package** di dalam aplikasi
 - Perbedaannya:
 - **PROTECTED**: dapat diakses oleh kelas yang **bukan subclass** dengan syarat di dalam package yang sama
 - **PUBLIC**: dapat diakses oleh kelas yang **bukan subclass** di semua package di dalam aplikasi
 - **Catatan!** Kelas yang **bukan subclass** **harus membuat objek** dari Superclass terlebih dahulu sebelum mengakses atribut dari SuperClass

Cek Kembali aturan **access modifier** pada java

Access Modifier	Dalam satu kelas	Dalam satu <i>package</i>	Di luar <i>package</i> oleh subkelas	Di luar <i>package</i> oleh siapapun
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Makin ke atas hak akses makin kuat/terbatas. *Private* adalah hak akses paling kuat.

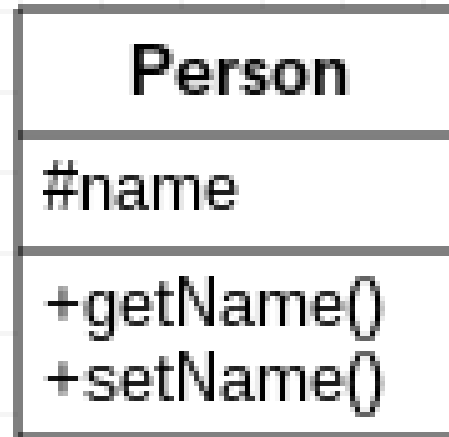




Use the **protected** modifier
*when you need to **only allow**
access to the code **within the**
package or when its **subclassed***

Pewarisan dalam pemrograman Java

- Pada class diagram (di StarUML), modifier **protected** digambarkan dengan tanda pagar (#).



Modifier Atribut	Dalam class itu sendiri	Dalam package yang sama		Berbeda package	
		Subclass	Other Class	Subclass	Other Class
Private	√				
Default	√	√	√	X	X
Protected	√	√	√	√	X
Public	√	√	√	√	√

*Class yang bukan subclass **harus membuat objek** dari Superclass terlebih dahulu sebelum mengakses atribut dari SuperClass

Pewarisan dalam pemrograman Java

- Buat **superclass** terlebih dahulu
 - Tentukan **access modifier** pada **atribut** dan **method** untuk mengatur hak akses atau “hak waris”.
- Buat **subclass** dengan menggunakan kata kunci **extends**.
 - Template kode program:

```
class NamaSubClass extends NamaSuperClass {}
```

Contoh:

```
class Mobil extends Kendaraan {  
    //isi kelas  
}
```

Pewarisan dalam pemrograman Java

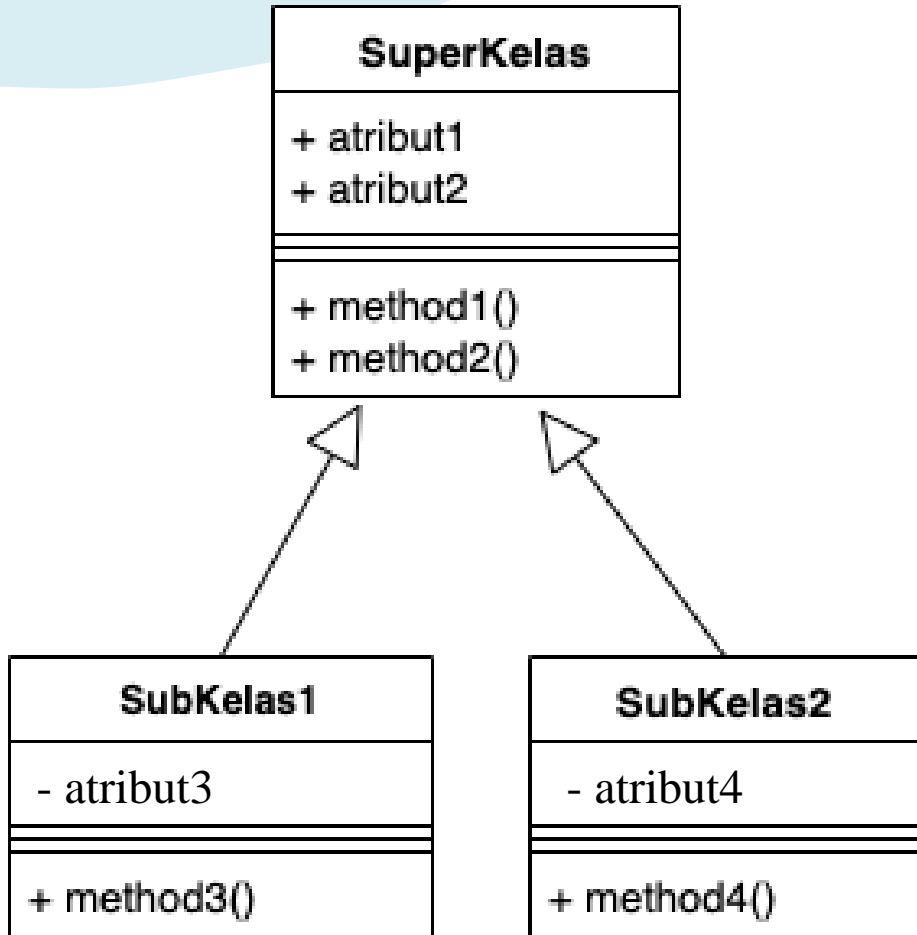
- Jika **berbeda package** maka lakukan **proses import** sebelum statemen **class**

```
import <nama_package>.<nama_class>;
```

Contoh:

```
package mobil;  
import kendaraan.Kendaraan;  
  
public class Mobil extends Kendaraan {  
    //isi kelas  
}
```

Contoh:



*SubKelas1 dibuat pada package yang berbeda untuk kepentingan praktik

```
package pewarisan;

public class SuperKelas {

    public int atribut1, atribut2;

    public void method1(){}
    public void method2(){}

}
```

superclass:
SuperKelas

```
package pewarisan;

public class SubKelas2 extends SuperKelas{
    private int atribut4;

    public void method4(){}

}
```

subclass:
SubKelas1
dan
SubKelas2

```
package DemoPewarisan;

import pewarisan.SuperKelas;

public class SubKelas1 extends SuperKelas{
    private int atribut3;

    public void method3(){}

}
```

Demonstrasi pewarisan

```
package demopewarisan;

import pewarisan.SubKelas2;
import pewarisan.SuperKelas;

public class DemoPewarisan {

    public static void main(String[] args) {
        SuperKelas super1 = new SubKelas1(); // instansiasi objek SuperKelas
        SubKelas1 sub1 = new SubKelas1(); // instansiasi objek SubKelas1
        SubKelas2 sub2 = new SubKelas2(); // instansiasi objek SubKelas2

        super1.method1(); // super kelas memanggil method1()

        sub1.method1(); //memanggil method1() yang diwarisi dari superkelas
        sub1.method3(); //memanggil methodnya sendiri, method3()

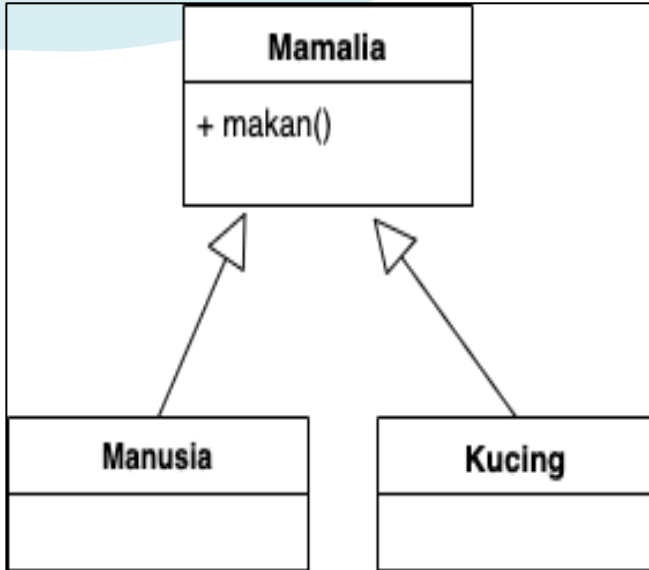
        sub2.method1(); //memanggil method yang diwarisi dari superkelas
        sub2.method4(); //memanggil methodnya sendiri, method4()
    }
}
```

Overriding

Definisi:

- **Mengubah** perilaku/implementasi **method superclass** pada *subclass*-nya.
- **Nama method** dan **parameter sama**, implementasi method yang berbeda.
- Hanya ada pada konsep pewarisan.
- **Overriding** beda dengan **overloading**:
 - **Overriding**: nama method sama, **parameter sama**, implementasi untuk tujuan berbeda. Berlaku pada **hubungan pewarisan**, antara *superclass* dengan *subclass*.
 - **Overloading**: nama method sama, **parameter beda**, implementasi bisa sama bisa beda. Berlaku pada **kelas yang sama**, bukan pada konsep pewarisan. (**lihat Kembali pada materi overloading**)

CONTOH:



* Manusia dan Kucing mewarisi method makan() dari Mamalia. Tentu saja cara makan manusia dan kucing berbeda.

```
package pewarisan;

public class Mamalia {

    public void makan(){
        System.out.println("Mamalia sedang makan");
    }

}
```

```
public class Manusia extends Mamalia{

    @Override // menambahkan anotasi/keterangan bahwa method sedang di-override
    public void makan(){
        System.out.println("Manusia makan DENGAN TANGAN");
    }

}
```

```
public class Kucing extends Mamalia{

    @Override // menambahkan anotasi/keterangan bahwa method sedang di-override
    public void makan(){
        System.out.println("Kucing makan DENGAN MULUT, tanpa tangan");
    }

}
```

Aturan overriding dan access modifier di Java (1)

Perubahan access modifier bisa dilakukan pada method yang akan di-override **hanya jika** ***access modifier baru*** di subclass **lebih lemah/rendah** daripada access modifier di superclass.

src > pewarisan > J Mamalia.java > ...

```
1 package pewarisan;
2
3 public class Mamalia {
4
5 >     public void makan(){...
8
9     protected void minum(){} // akses modifier = protected
10 }
```

Ada error pada kode program baris ke-5 yang menunjukkan bahwa perubahan access modifier pada saat override **tidak bisa dilakukan** karena access modifier **LEBIH KUAT** daripada milik superkelas (**Mamalia**).

src > pewarisan > J Manusia.java > ...

```
1 package pewarisan;
2
3 public class Manusia extends Mamalia {
4     @Override
5     public void minum(){ // akses modifier baru = public (lebih lemah dari protected)
6         System.out.println(x:"Manusia minum DENGAN TANGAN");
7     }
8 }
```

src > pewarisan > J Kucing.java > ...

```
1 package pewarisan;
2
3 public class Kucing extends Mamalia {
4     @Override
5     void minum(){ // akses modifier baru = default (lebih kuat dari protected)
6         System.out.println(x:"Kucing minum DENGAN MULUT");
7     }
8 }
```

Konstruktor pada pewarisan (1)

- Konstruktor dari superclass **TIDAK TERMASUK** bagian yang **diwariskan** pada subclass.
- **Konstruktor** superclass **SELALU** dideklarasikan baik secara implisit maupun eksplisit pada **konstruktor subclass**.
- **Konstruktor** superclass **SELALU** dipanggil saat objek dari subclass diciptakan.

Konstruktor pada pewarisan (2)

- Jika **konstruktor subclass tidak dideklarasikan** secara eksplisit, yang terjadi adalah:
 - Secara **implisit default constructor** pada subclass **berisi pemanggilan default konstruktor pada superclass**.
 - Ketika **objek** dari subclass diciptakan, yang dipanggil adalah **default constructor** superclass, baru kemudian **default constructor** dari subclass-nya.

*yang dimaksud dengan default constructor adalah **konstruktor tanpa parameter**

Contoh 1

```
public class Mamalia {  
    String jenis;  
  
    public Mamalia(){ //default konstruktor Mamalia  
        System.out.println("Mamalia baru telah lahir");  
    }  
  
    public Mamalia(String jenis){ //overload konstruktor  
        this.jenis = jenis;  
        System.out.println("Mamalia "+jenis+" baru telah lahir");  
    }  
}
```

Konstruktor pada kelas Kucing tidak di deklarasikan

```
public class Kucing extends Mamalia{  
    // konstruktor kucing tidak di deklarasikan  
    // secara eksplisit  
}
```

```
public class DemoMamalia {  
    public static void main(String[] args) {  
        Kucing kucing1 = new Kucing();  
    }  
}
```

Dari output program bisa dilihat bahwa: pada saat objek kucing diciptakan, konstruktor Mamalia ikut dipanggil juga

```
Output - kuliahFPA (run) x  
FUN:  
Mamalia baru telah lahir  
BUILD SUCCESSFUL (total time: 0 seconds)
```


Konstruktor pada pewarisan (3)

- Ketika konstruktor `subclass` dideklarasikan sedangkan konstruktor `superclass` **TIDAK** dideklarasikan di `subclass` maka `default constructor superclass` **tetap akan dipanggil**.

Contoh 2

```
public class Mamalia {  
    String jenis;  
  
    public Mamalia(){ //default konstruktor Mamalia  
        System.out.println("Mamalia baru telah lahir");  
    }  
  
    public Mamalia(String jenis){ //overload konstruktor  
        this.jenis = jenis;  
        System.out.println("Mamalia "+jenis+" baru telah lahir");  
    }  
}
```

```
public class Simpanse extends Mamalia{  
    public Simpanse(){  
        System.out.println("simpanse tercipta");  
    }  
}
```

```
public class DemoMamalia {  
    public static void main(String[] args) {  
        Simpanse simpanse1 = new Simpanse();  
    }  
}
```

Konstruktor pada kelas Simpanse dideklarasikan, tapi konstruktor Mamalia tidak dideklarasikan.

Dari output program bisa dilihat bahwa: pada saat objek simpanse diciptakan, konstruktor Mamalia dipanggil terlebih dahulu, baru kemudian konstruktor Simpanse

Output - kuliahFPA (run)

```
run:  
Mamalia baru telah lahir  
simpanse tercipta  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Konstruktor pada pewarisan (4)

- Jika **konstruktor** *superclass* **dideklarasikan** secara **eksplisit** pada konstruktor *subclass*
 - Deklarasi **harus dilakukan tepat di bawah** deklarasi nama konstruktor *subclass*.
 - **Tidak boleh** didahului statement program apapun karena **pemanggilan konstruktor** HARUS merupakan **statemen pertama** dalam sebuah konstruktor
- Gunakan kata kunci **super()** untuk menggantikan penyebutan konstruktor *superclass*.

Contoh 3

```
public class Mamalia {  
    String jenis;  
  
    public Mamalia(){ //default konstruktor Mamalia  
        System.out.println("Mamalia baru telah lahir");  
    }  
  
    public Mamalia(String jenis){ //overload konstruktor  
        this.jenis = jenis;  
        System.out.println("Mamalia "+jenis+" baru telah lahir");  
    }  
}
```

```
public class Manusia extends Mamalia{  
    public Manusia(){  
        super("manusia");  
        System.out.println("Alhamdulillah");  
    }  
}
```

```
public class DemoMamalia {  
    public static void main(String[] args) {  
        Manusia man1 = new Manusia();  
    }  
}
```

Konstruktor pada kelas Manusia dideklarasikan dan sekaligus mendeklarasikan overload konstruktor Mamalia tepat dibawahnya.

Dari output program bisa dilihat bahwa: pada saat objek manusia diciptakan, (overload) konstruktor Mamalia dipanggil sesuai dengan deklarasi pada konstruktor Manusia

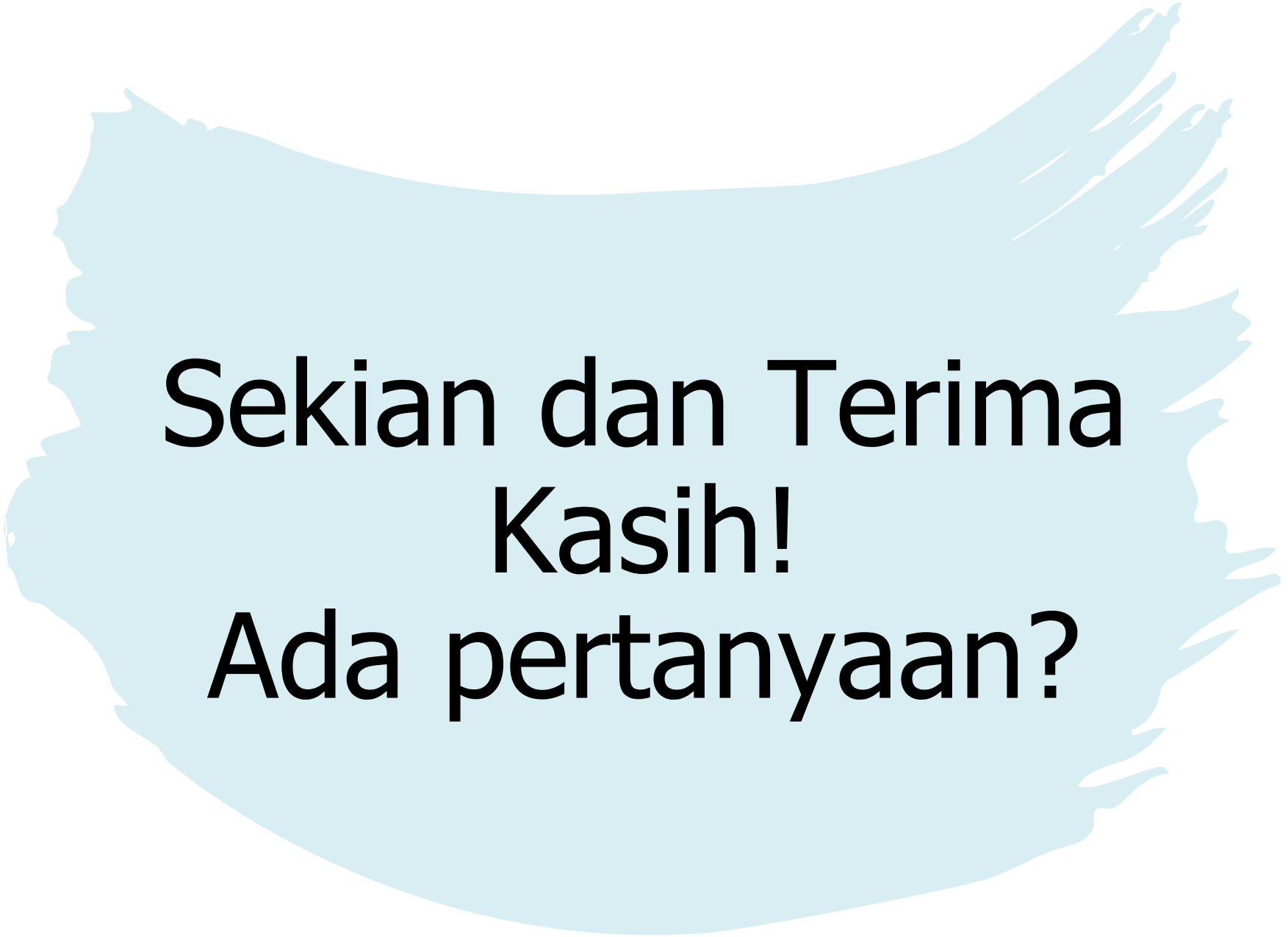
```
Output - kuliahFPA (run) x  
Run:  
Mamalia manusia baru telah lahir  
Alhamdulillah  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kata kunci **super**

- Untuk merujuk pada *member* dari **parent class**.
- Mirip dengan kata kunci **this** yang digunakan untuk merujuk pada member dari class itu sendiri.
- Format penulisan:
 - **super.attribute**
Untuk merujuk pada attribut pada parent class
 - **super.method()**
Untuk merujuk pada method pada parent class
 - **super()**
Untuk merujuk pada constructor pada parent class

Contoh kode program bisa dilihat di sini:

<https://www.javatpoint.com/super-keyword>

A light blue brushstroke background with a textured, hand-painted appearance, featuring various shades of blue and white strokes.

**Sekian dan Terima
Kasih!
Ada pertanyaan?**