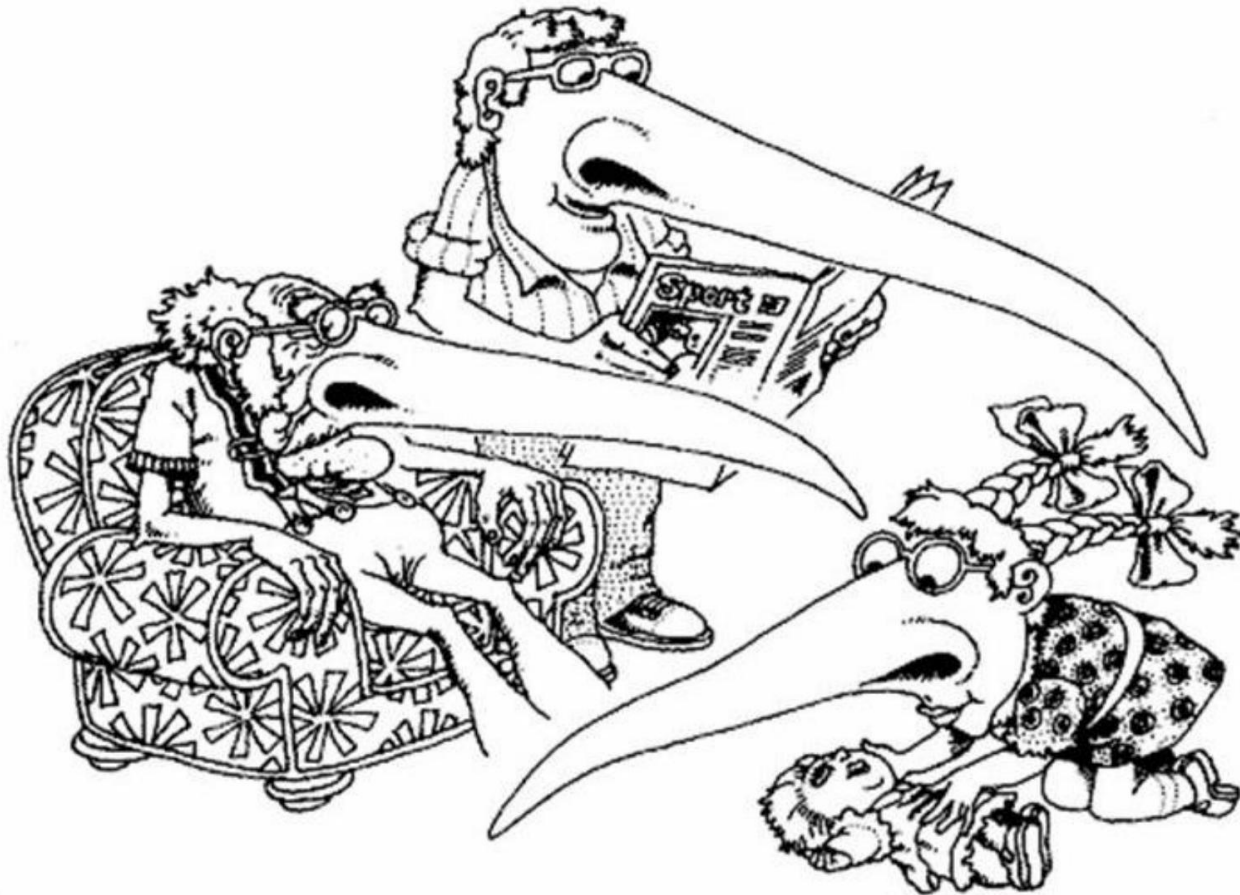


# Pemrograman Berorientasi Objek



**Warisan**

## Apa itu Warisan?



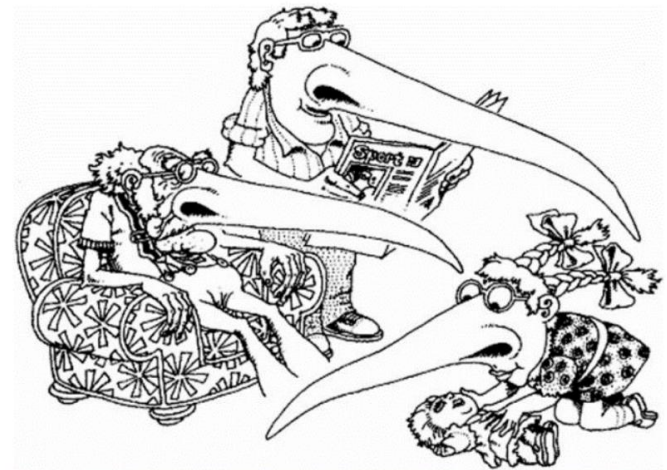
# Warisan

- ▶ Kemampuan kelas Java untuk 'meneruskan' semua atau beberapa bidang, atribut, dan metode mereka ke kelas lain yang akan disebut sebagai kelas anak mereka
- ▶ Kelas anak akan dapat mengenali bidang dan metode yang diwarisi, dan dapat menggunakannya tanpa mendeklarasikan atau mendefinisikan lagi



# Warisan

- Pewarisan adalah implementasi hubungan kelas generalisasi-spesialisasi atau hubungan "adalah-a"
- Singkatnya: ini adalah mekanisme dimana satu kelas memperoleh properti dari kelas lain.



# Warisan

- Dilambangkan dengan tanda panah (kepala berongga dan segitiga)

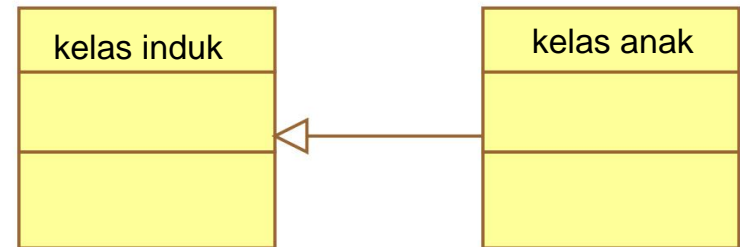


- Kelas yang mewariskan atribut dan metode disebut
  - kelas induk, kelas super, atau kelas dasar
- Kelas yang mewarisi atribut dan metode disebut
  - Kelas anak, subkelas, atau kelas turunan

# Jenis-jenis Warisan

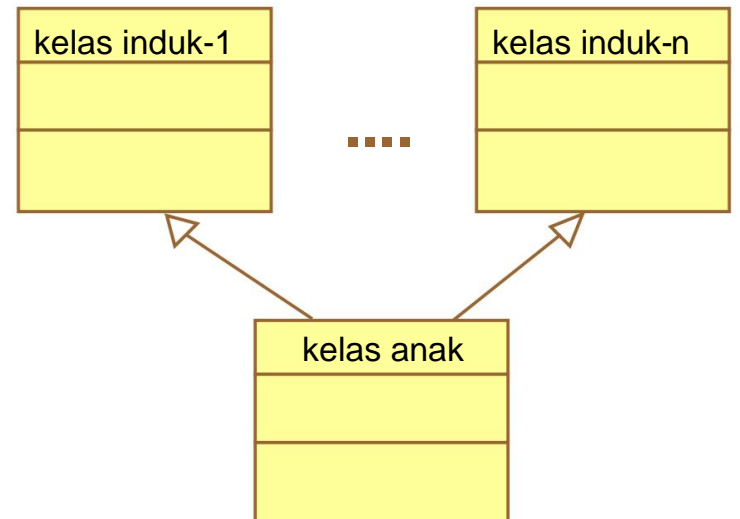
## ➤ Warisan Tunggal

- Kelas anak mewarisi dari 1 kelas induk



## ➤ Warisan Ganda

- Kelas anak mewarisi banyak kelas induk



# Pewarisan Kelas Java

- ▶ Hanya pewarisan tunggal
- ▶ Menggunakan kata kunci extends

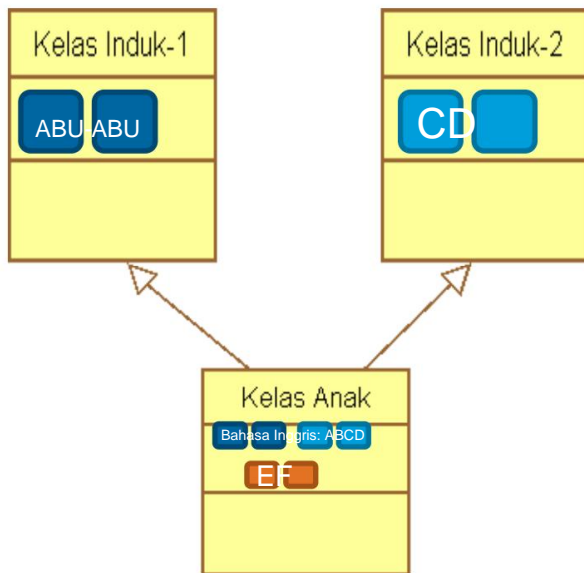
```
kelas ChildClass memperluas ParentClass{  
    // atribut kelas anak // metode  
    kelas anak  
}
```

- ▶ Semua atribut dan metode non-pribadi dilewatkan pada
- ▶ Untuk mengakses properti induk
  - Gunakan kata kunci "super"

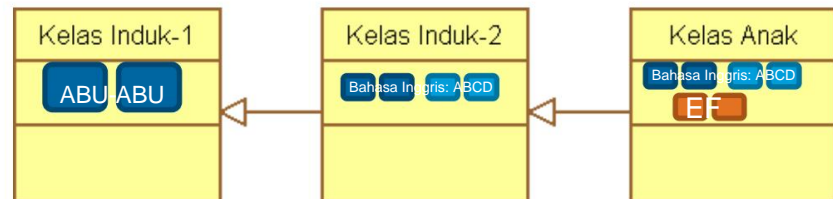


# Pewarisan Ganda di Java

- Pewarisan Berganda ditangani menggunakan pewarisan tunggal bertahap



Pewarisan Ganda Basis



Pewarisan Ganda menggunakan  
Pewarisan Tunggal Bertahap



# Contoh - Pewarisan Metode

```
kelas publik Parent{  
    metode publik StringInduk(){  
        kembalikan "ini adalah metode Induk";  
    }  
}
```

```
kelas publik Anak memperluas Orang Tua { publik  
    String metodeAnak() {  
        kembalikan "ini adalah metode Anak";  
    }  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Induk p = new Induk();  
        Anak c = new Anak();  
  
        System.out.println(p.indukmetode());  
        System.out.println(c.indukmetode());  
  
        Sistem.keluar.println(c.anakmetode());  
        Sistem.keluar.println(p.anakmetode());  
    }  
}
```

- > Ini adalah metode Induk
- > Ini adalah metode Induk
- > Ini adalah metode Anak
- > **kesalahan: tidak dapat menemukan simbol**


Anak dapat mengakses  
metode publik  
Orang tua, tapi bukan wakil  
sebaliknya

# Contoh – Pewarisan Variabel

```
kelas publik Parent{ publik int  
    publicInt; privat int privateInt; int  
    defaultInt;  
  
}
```

```
kelas publik Anak memperluas Induk { publik void  
    methodChild() { publicInt = 10; privateInt =  
        20; defaultInt = 30;  
  
    }  
}
```

kesalahan: privateInt  
memiliki akses pribadi



# Contoh

```
kelas publik Parent{  
    publik int publikInt; privat int privatInt;  
    int defaultInt;  
  
    publik void setPrivateInt(int privateInt){ ini.privateInt = privateInt;  
  
    }  
}
```

```
kelas publik Anak memperluas Induk { publik void  
    methodChild() { publicInt = 10; setPrivateInt  
        (20); defaultInt = 30;  
  
    }  
}
```

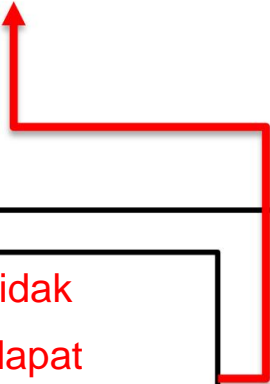
# Contoh

paket uji; kelas publik

```
Parent{  
    publik int publikInt; privat int privatInt;  
    int defaultInt;  
  
    publik void setPrivateInt(int privateInt){ ini.privateInt = privateInt;  
  
}
```

import test.Parent; kelas publik

```
Anak memperluas Parent{ publik void methodChild()  
    { publicInt = 10; setPrivateInt(20); defaultInt  
      = 30;  
  
    }  
}
```



kesalahan: defaultInt tidak  
publik di Induk; tidak dapat  
diakses dari luar paket

## Akses Pengubah Dilindungi

- Semua kelas dalam paket yang sama
- Semua subkelas (kelas anak) bahkan dalam paket yang berbeda



# Contoh

paket uji; kelas publik


```
Parent{  
    publik int publikInt; privat int privatInt;  
    int defaultInt; dilindungi int dilindungiInt;  
  
    publik void setPrivateInt(int privateInt){ ini.privateInt = privateInt;  
  
    }  
}
```

import test.Parent; kelas publik

```
Anak memperluas Parent{ publik void methodChild()  
    { publicInt = 10; setPrivateInt(20); protectedInt  
      = 30;  
  
    }  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak(); c.metodeAnak();  
  
    }  
}
```

Modifikasi ke  
protectedInt dilakukan oleh  
Kelas anak



# Contoh

paket uji; kelas publik

```
Parent{  
    publik int publikInt; privat int privatInt;  
    int defaultInt; dilindungi int dilindungiInt;  
  
    publik void setPrivateInt(int privateInt){ ini.privateInt = privateInt;  
  
    }  
}
```


import test.Parent; kelas publik

```
Anak memperluas Parent{  
  
    }
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak(); c.publicInt =  
        10; c.setPrivateInt(20);  
        c.protectedInt = 40;  
  
    }  
}
```

error: protectedInt memiliki  
akses terlindungi di induk

(protectedInt dimiliki oleh  
Orang tua dalam paket  
berbeda )





# Utama

- Mendefinisikan ulang bidang atau metode yang diwarisi oleh kelas Induk di Kelas anak
- subkelas dapat mengimplementasikan metode kelas induk secara lebih spesifik berdasarkan kebutuhannya.
- Saat mengganti bidang atau metode, kelas Anak akan memiliki akses ke bidang/metode induk asli dan bidang/metode anak baru yang didefinisikan ulang
- Panggil bidang/metode induk menggunakan kata kunci "super"
  - Kata kunci "super" hanya tersedia di dalam kelas anak

# Contoh

```
kelas publik Parent{ publik String  
    toString(){ kembalikan "ini adalah metode  
        Parent";  
    }  
}
```

```
kelas publik Anak memperluas Induk{  
  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak();  
        Induk p = new Induk();  
  
        Sistem.keluar.println(p.keString());  
        Sistem.keluar.println(c.keString());  
    }  
}
```

> ini adalah metode Induk  
> ini adalah metode Induk

# Contoh

```
kelas publik Parent{ publik String  
    toString(){ kembalikan "ini adalah metode  
        Parent";  
    }  
}
```

```
kelas publik Anak memperluas Induk { publik String  
    toString() {  
        kembalikan "metode ini ditimpa"  
            oleh Anak";  
    }  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak();  
        Induk p = new Induk();  
  
        Sistem.keluar.println(p.keString());  
        Sistem.keluar.println(c.keString());  
    }  
}
```

> ini adalah metode Induk  
> metode ini ditimpa oleh Anak

# Contoh

```

kelas publik Parent{ publik String
    toString(){ kembalikan "ini adalah metode
        Parent";
    }
}

```

```

kelas publik Anak memperluas Induk { publik String
    toString() {
        kembalikan "metode ini ditimpa"
            oleh Anak";

    } publik String toStringParent(){ kembalikan super.toString();

    }
}

```

```

kelas publik Driver{
    publik statis void utama(String args[]){
        Anak c = new Anak();
        Induk p = new Induk();

        Sistem.keluar.println(p.toString()); Sistem.keluar.println(c.toString());
        Sistem.keluar.println(c.toStringParent());

    }
}

```

> ini adalah metode Induk  
 > metode ini ditimpa oleh Anak > ini adalah metode Induk

# Contoh

```
kelas publik Parent{  
    nomor int yang dilindungi ;  
  
}
```

```
kelas publik Anak memperluas Induk { privat int angka;
```

```
    publik void metodeAnak(){  
        angka = 10;  
        super.angka = 20;  
  
        System.out.println(angka);  
        System.out.println(super.angka);  
    }  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak();  
        c.metodeAnak();  
    }  
}
```

```
> 10  
> 20
```


# Contoh

```
kelas publik Parent{  
    nomor int yang dilindungi ;  
  
    publik void setNumber(int nomor){  
        ini.angka = angka;  
    }  
  
    publik int getNumber(){ kembalikan  
        nomor;  
    }  
}
```

```
kelas publik Anak memperluas Induk { privat int angka;  
  
    publik void metodeAnak(){  
        angka = 10;  
        super.angka = 20;  
    }  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak();  
        c.metodeAnak();  
  
        System.out.println(c.dapatkanNomor());  
    }  
}
```

> 20



setNumber() dan  
getNumber() dimiliki oleh  
induk, mengakses nilai angka  
dari induk (super)

# Contoh

```
kelas publik Parent{  
    nomor int yang dilindungi ;  
  
    publik void setNumber(int nomor){  
        ini.angka = angka;  
    }  
  
    publik int getNumber(){ kembalikan  
        nomor;  
    }  
}
```

```
kelas publik Anak memperluas Induk { privat int angka;  
  
    publik void metodeAnak(){  
        angka = 10;  
        super.angka = 20;  
    }  
  
    } publik int getNumber(){ kembalikan  
        nomor;  
    }  
}
```

```
kelas publik Driver{  
    publik statis void utama(String args[]){  
        Anak c = new Anak(); c.metodeAnak();  
  
        System.out.println(c.dapatkanNomor());  
    }  
}
```

> 10

Jika ditimpa, metode  
akan mengakses  
bidang anak terlebih  
dahulu



## Pengubah Non-Akses Final

- Untuk membatasi akses untuk mengubah kelas, bidang, atau metode
- Pengubah akhir dalam metode
  - Metode tidak dapat ditimpa oleh kelas anak, hanya untuk penggunaan – `publik final void finMethod(){ ... }`

```
kelas publik Parent{ publik void  
    publicMethod(){  
  
    } publik akhir void finMethod(){  
  
    }  
}
```

```
kelas publik Anak memperluas Induk { publik void  
    publicMethod(){ //ok  
  
    } publik void finMethod(){ // dilarang  
  
    }  
}
```

## Pengubah Non-Akses Final

- ▶ Pengubah akhir di kelas
  - Kelas tidak dapat diperluas (disubkelaskan) oleh kelas lain
  - `public final class FinClass{ ... }`

```
Kelas publik final FinalParent{  
  
}
```

```
kelas publik Child extends FinalParent{ // dilarang  
  
}
```

## Pengubah Non-Akses Final

### ► Pengubah akhir di bidang/variabel

- Bidang/variabel adalah konstanta, nilai ditetapkan saat dideklarasikan atau dimulai dalam konstruktor, dan tidak dapat diubah setelahnya – private final double

finDouble

```
kelas publik Driver{  
    publik statis void main(String args[]){ int akhir a = 5; int akhir b; b =  
        8;  
  
        bilangan          // dilarang //  
        prima : 15 ;      dilarang  
    }  
}
```

- dalam diagram kelas, konstanta final direpresentasikan dengan huruf kapital semua dan menggunakan garis bawah untuk memisahkan kata-kata

## Pengubah Non-Akses Final

- ▶ Pengubah Final dalam Variabel Referensi
  - Setelah variabel merujuk ke suatu objek, variabel tersebut tidak dapat diikat kembali untuk merujuk ke objek lain.
  - Tetapi objek yang dirujuknya masih bisa berubah, jika memang awalnya bisa berubah.
  - Contoh:  
    `private final NamaKelas finObject;`

## Pengubah Non-Akses Final

```
kelas publik Parent{ int nomor;
```

```
    publik void setNumber(int nomor){  
        ini.angka = angka;  
    }
```

```
    publik int getNumber(){ kembalikan  
        nomor;  
    }  
}
```

```
kelas publik Driver{
```

```
    publik statis void utama(String args[]){  
        Induk p = Induk baru (); Induk akhir  
        fp1 = Induk baru (); Induk akhir fp2; fp2 = p;
```

```
        fp1 = p; fp2  
        = new Parent();
```

**// dilarang //**  
**dilarang**

```
        fp2.setNomor(5);  
        fp1.setNomor(20);
```

```
    }  
}
```

# Konstruktor **tidak** diwariskan

- Jika hanya ada konstruktor non-default di kelas induk
  - Harus ada setidaknya satu konstruktor anak yang memanggil satu konstruktor induk
  - Setiap konstruktor di kelas anak harus memanggil satu induk konstruktor
  - Panggil konstruktor induk menggunakan metode super(parameter)
- Jika ada konstruktor default di kelas induk
  - kelas anak tidak harus memanggil konstruktor induk

# Contoh

```
kelas publik Parent{  
    dilindungi int vP1;  
    dilindungi ganda vP2;  
  
    publik Induk(){  
    }  
  
    publik void metodeInduk(){  
    }  
}
```

```
kelas publik Anak memperluas Induk{  
    pribadi int vC;  
  
    publik Anak(int vC){  
        ini.vC = vC;  
    }  
  
    publik void metodeAnak(){  
    }  
}
```

Jika terdapat konstruktor default atau tidak ada konstruktor sama sekali di kelas induk, maka kelas anak tidak perlu memanggil konstruktor kelas induk.

kelas anak mungkin atau mungkin tidak memiliki konstruktornya sendiri



# Contoh

```

kelas publik Parent{
    dilindungi int vP1;
    dilindungi ganda vP2;

    publik Induk(int vP1){
        ini.vP1 = vP1;
    }

    publik Induk(int vP1, double vP2 ){
        ini.vP1 = vP1;
        ini.vP2 = vP2;
    }

    publik Induk(ganda vP2 ){
        ini.vP2 = vP2;
    }

    publik void metodeInduk(){
    }
}

```

```

kelas publik Anak memperluas Induk{
    pribadi int vC;

    publik Anak(int vC, double vP){
        ini.vC = vC;
        super(vP);
    }

    publik void metodeAnak(){
    }

}

```

Jika ada konstruktor non-default di kelas induk,  
Harus ada SEKURANG-KURANGNYA satu konstruktor anak yang memanggil satu konstruktor induk

Gunakan kata kunci **super()** untuk memanggil konstruktor induk

# Contoh

```

kelas publik Parent{
    dilindungi int vP1;
    dilindungi ganda vP2;

    publik Induk(int vP1){ ini.vP1 = vP1;

}

    publik Induk(int vP1,double vP2 ){ ini.vP1 = vP1; ini.vP2 =
        vP2;

}

    publik Induk(ganda vP2 ){ ini.vP2 = vP2;

}

    publik void metodeInduk(){ }

}
  
```

```

kelas publik Anak memperluas Induk { privat int vC;

    publik Anak(int vC, double vP){ this.vC = vC; super(vP);

}

    publik Anak(ganda vP){ super(5, vP);

}

    publik Anak(){ super(2.5);

}

    publik void metodeAnak(){ }
  
```

Semua konstruktor anak harus memanggil konstruktor induk

# Contoh

```

kelas publik Parent{
    dilindungi int vP1;
    dilindungi ganda vP2;

    publik Induk(){
        Nilai P1 = 50;
    }

    publik Induk(int vP1,double vP2 ){
        ini.vP1 = vP1;
        ini.vP2 = vP2;
    }

    publik Induk(ganda vP2 ){
        ini.vP2 = vP2;
    }

    void methodParent(){
}
  
```

Jika kelas induk memiliki konstruktor default, publik maka kelas anak mungkin } atau mungkin tidak memanggil konstruktor induk sama sekali

```

kelas publik Anak memperluas Induk{
    pribadi int vC;

    publik Anak(int vC, double vP){
        super(vC, vP);
    }

    publik Anak(int vC){
        ini.vC = vC;
    }

    publik Anak(){
        ini.vC = 25;
    }

    methodChild(){
}
  
```

Konstruktor anak yang public void tidak memanggil konstruktor induk mana pun akan secara otomatis memanggil konstruktor induk default

## Kapan Harus Menggunakan

### ▶ Spesialisasi

- Tampilan atas ke bawah

### ▶ Generalisasi

- Tampilan bawah ke atas

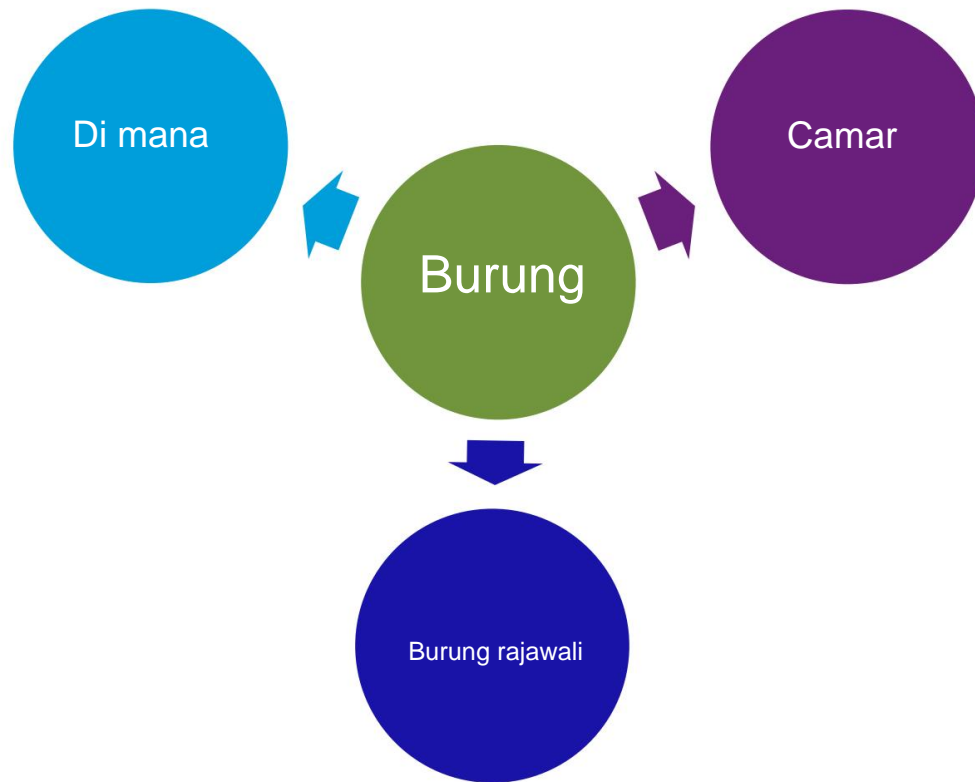


# Spesialisasi

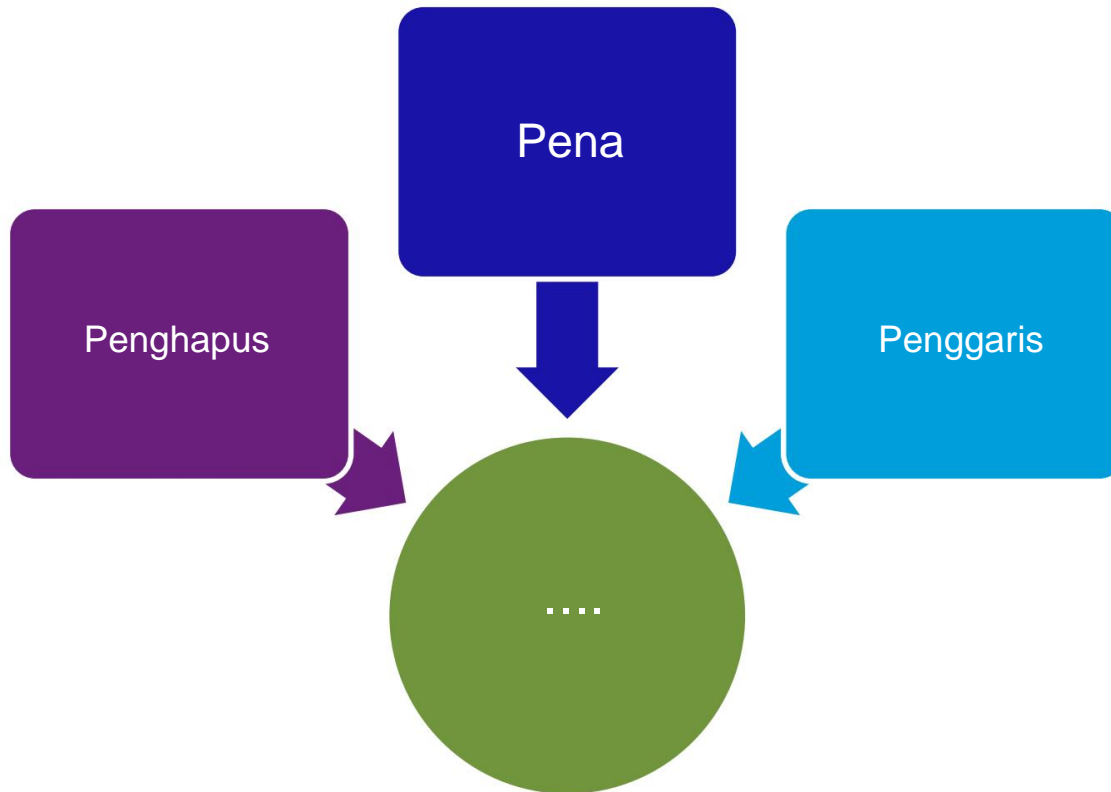


Burung

# Spesialisasi

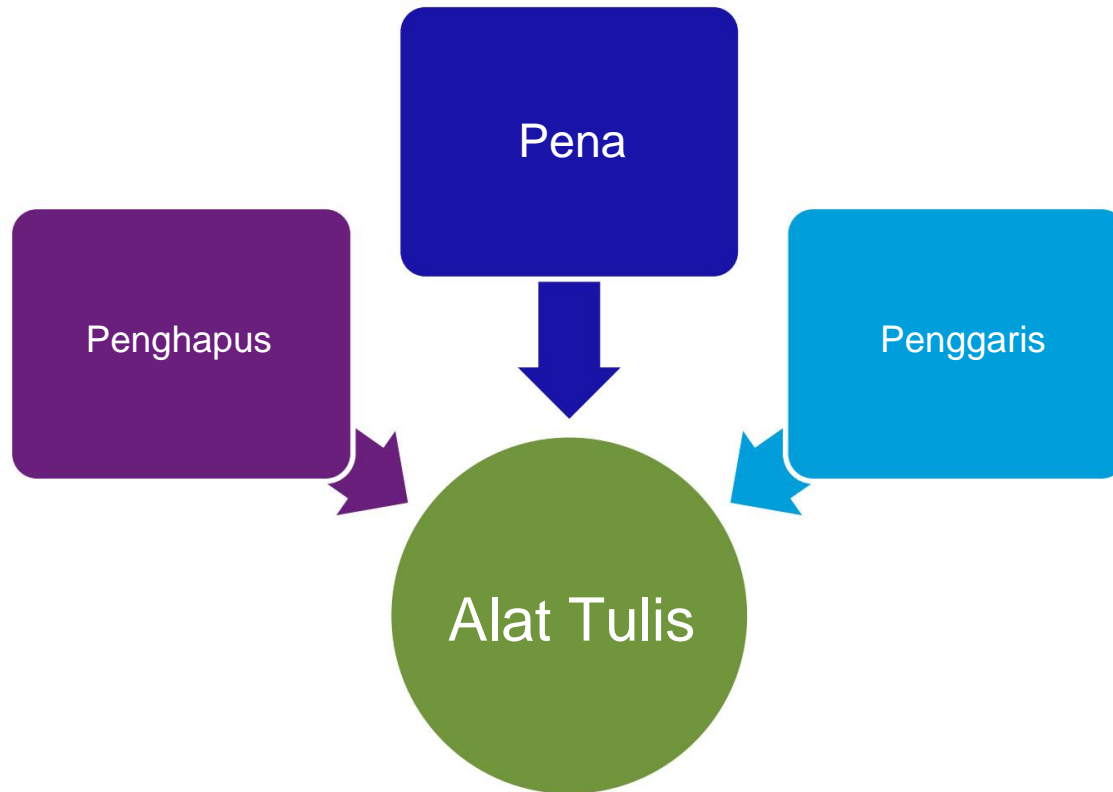


# Generalisasi

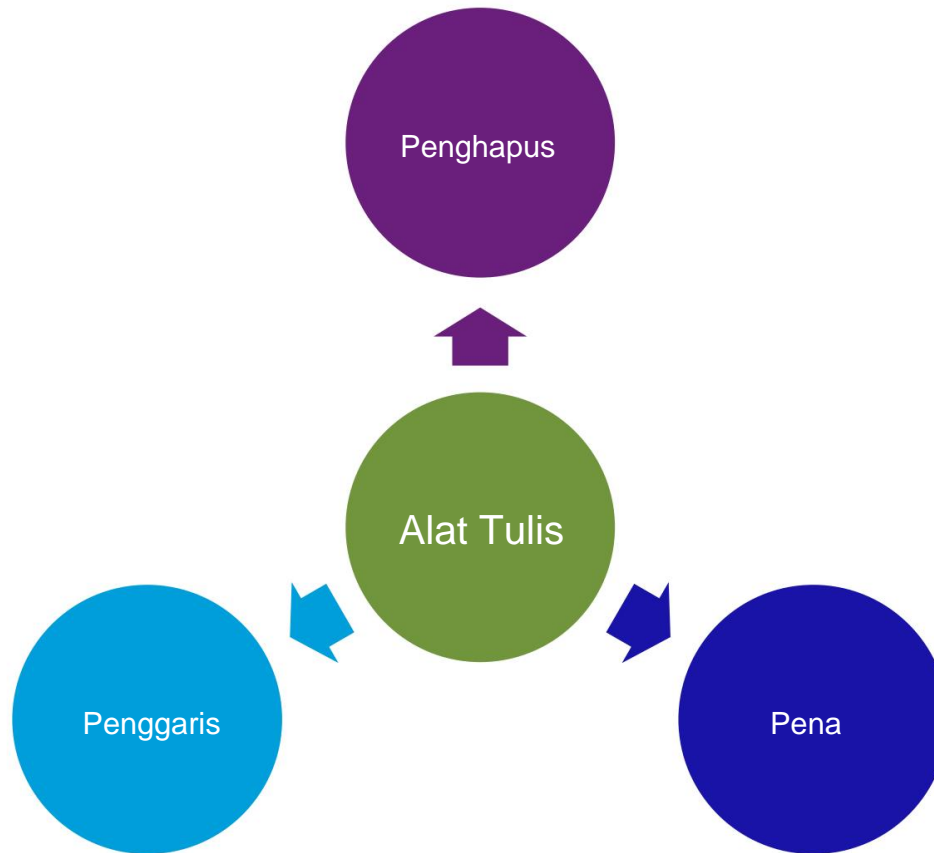




# Generalisasi

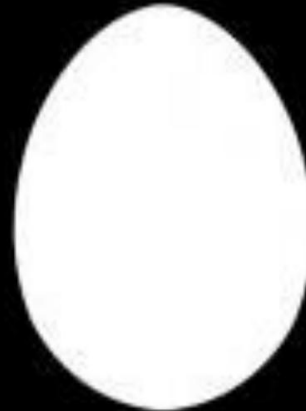
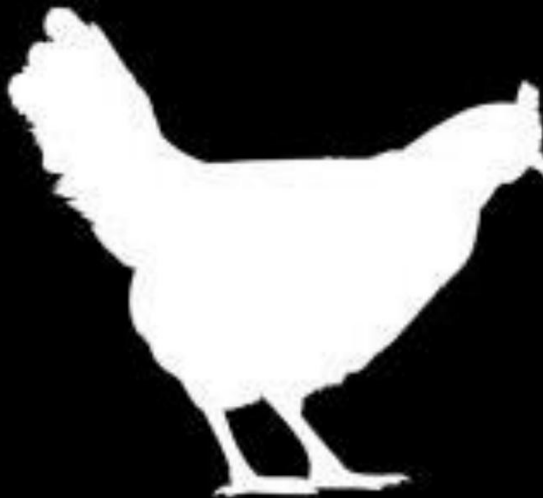


# Spesialisasi atau Generalisasi?





# Generalisasi & Spesialisasi



## Kapan harus digunakan

### ▶ Tampilan Atas-Bawah

- untuk membuat kelas baru dengan beberapa perilaku yang sudah ada di kelas lain (atau mirip dengan kelas lain)
- untuk membuat kelas yang lebih spesifik atau modifikasi dari basis kelas
- untuk memecah kelas besar menjadi kelas-kelas yang lebih spesifik
- untuk menciptakan bentuk hierarki kelas

### ▶ Karena alasan tersebut kita dapat membuat subkelas yang mewarisi kelas dasar

## Kapan harus digunakan

### ▶ Tampilan Bawah-Atas

- Ketika ada beberapa kelas yang secara umum sama (banyak metode dan bidang yang sama atau serupa) tetapi tidak perlu bahwa salah satu kelas adalah anak dari yang lain
- Untuk membuat modul untuk berinteraksi antar kelas

### ▶ Karena alasan tersebut kita dapat membuat kelas induk yang mengumpulkan kelas-kelas lain sebagai kelas anaknya

# Manfaat Warisan

- ▶ Dapat digunakan kembali
  - Gunakan kembali metode dan data kelas yang ada
- ▶ Memperpanjang
  - Menambahkan data baru dan metode baru ke kelas yang ada
- ▶ Memodifikasi
  - Ubah kelas yang ada dengan mengganti metodenya dengan implementasi Anda sendiri
- ▶ Pengurangan (Pengelompokan dan Pengorganisasian)

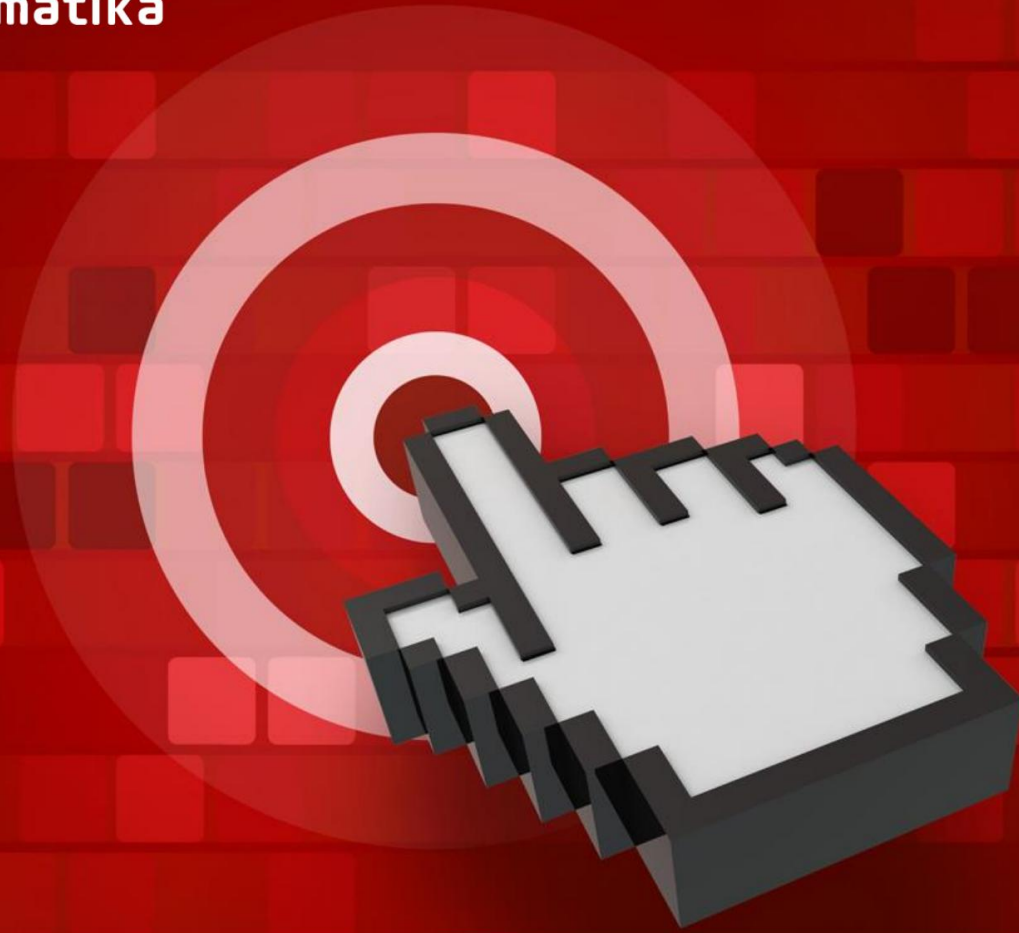
## 5 hal yang mungkin Anda temukan di Hirarki *Warisan*:

- superclass terlalu umum untuk mendeklarasikan semua perilaku, jadi setiap subclass menambahkan perilakunya sendiri
- superclass mengatur perilaku abstrak dan karenanya mendelegasikan implementasi ke subclass-nya.
- superclass menentukan perilaku, subclass mewarisi perilaku.
- superclass menentukan perilaku, subclass dapat memilih untuk mengesampingkan perilaku sepenuhnya
  - hanya karena sebuah subkelas mewarisi sebuah metode tidak berarti bahwa ia harus bertindak dengan cara yang sama seperti superkelasnya
  - subkelas dapat memilih untuk menolak implementasi superkelasnya metode dan superclass
- "lakukan dengan cara saya" menentukan perilaku, subclass dapat memilih untuk mengesampingkan perilaku sebagian
  - disebut pengabaian sebagian

# Pertanyaan?







# TERIMA KASIH