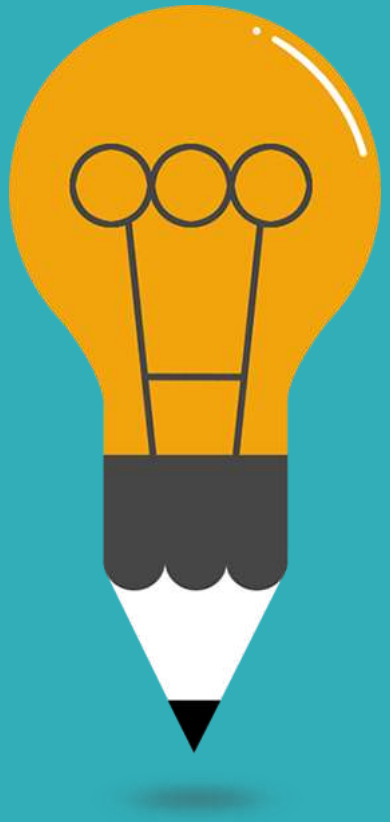


Praktikum PBO 2020

Inheritance and Polymorphism

Pembahasan

Kuis sebelumnya

```
public class Mix4 {
  int counter = 0;
  public static void main(String [] args) {
    int count = 0;
    Mix4 [] m4a  =new Mix4[20];
    int x = 0;
    while (          ) {
      m4a[x] = new Mix4();
      m4a[x].counter = m4a[x].counter + 1;
      count = count + 1;
      count = count + m4a[x].maybeNew(x);
      x = x + 1;
    }
    System.out.println(count + " "
                       + m4a[1].counter);
  }

  public int maybeNew(int index) {
    if (          ) {
      Mix4 m4 = new Mix4();
      m4.counter = m4.counter + 1;
      return 1;
    }
    return 0;
  }
}
```

Soal 1

Pasangkan kandidat kode (bagian kiri) dengan output program yang dihasil kan dari source code yang telah terisi (bagian kanan).

`x < 9` `index < 5`	**14 1**
`x < 20` `index < 5`	**25 1**
`x < 7` `index < 7`	**14 1**
`x < 19` `index < 1`	**20 1**

Soal 2

Tentukan "Siapakah aku/ini" dari masing-masing pernyataan berikut ini:
Pilihan jawaban: *method, instance variable, argument, return, getter, setter, encapsulation, public, private*

- Sebuah class dapat memiliki sejumlah ini
  jawab: [instance variable, getter, setter, method],
- Sebuah method hanya bisa memiliki satu ini
  jawab: [return]
- Aku mengutamakan instance variable-ku private
  jawab: [encapsulation]
- Hanya setter yang harus meng-update ini
  jawab: [instance variable]
- Sebuah method dapat memiliki banyak ini
  jawab: [argument]
- Aku mengembalikan sesuatu menurut definisi
  jawab: [getter]
- Aku tidak boleh digunakan pada instance variable
  jawab: [public]
- Aku bisa memiliki banyak argument
  jawab: [method]
- Menurut definisi, Aku mengambil satu argumen
  jawab: [setter]
- Ini membantu membuat enkapsulasi
  jawab: [getter, setter, public, private]

Soal 3

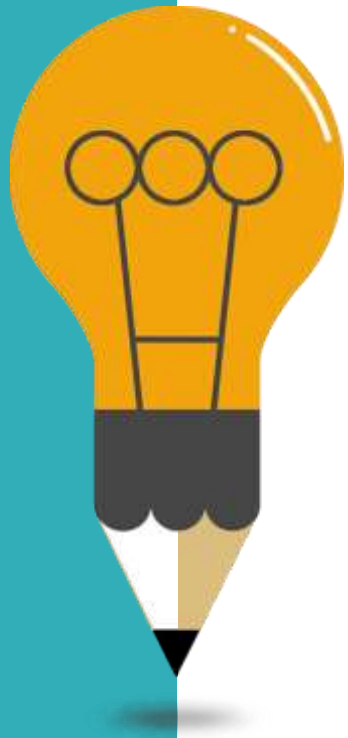
Diketahui terdapat method berikut ini:

```
int calcArea(int height, int width) {
    return height * width;
}
```

Dan dibawah ini terdapat list kode. **Tentukan yes/no** dari list berikut yang menyatakan dapat atau tidak digunakan untuk pemanggilan method tersebut.

- `int a = calcArea(7, 12);` → yes
- `short c = 7;`
  `calcArea(c,15);` → no
- `int d = calcArea(57);` → no
- `calcArea(2,3);` → yes
- `long t = 42;`
  `int f = calcArea(t,17);` → no
- `int g = calcArea();` → no
- `calcArea();` → no
- `byte h = calcArea(4,20);` → no
- `int j = calcArea(2,3,5);` → no

Outline



01 **Inheritance**

Tentang pewarisan, mendesain pohon pewarisan, menguji desain

02 **Access levels dan beberapa aturan pewarisan**

03 **Polymorphism**

Cara kerja polimorfisme, tipe return dan argumen polimorfik

04 **Overriding – Overloading method**

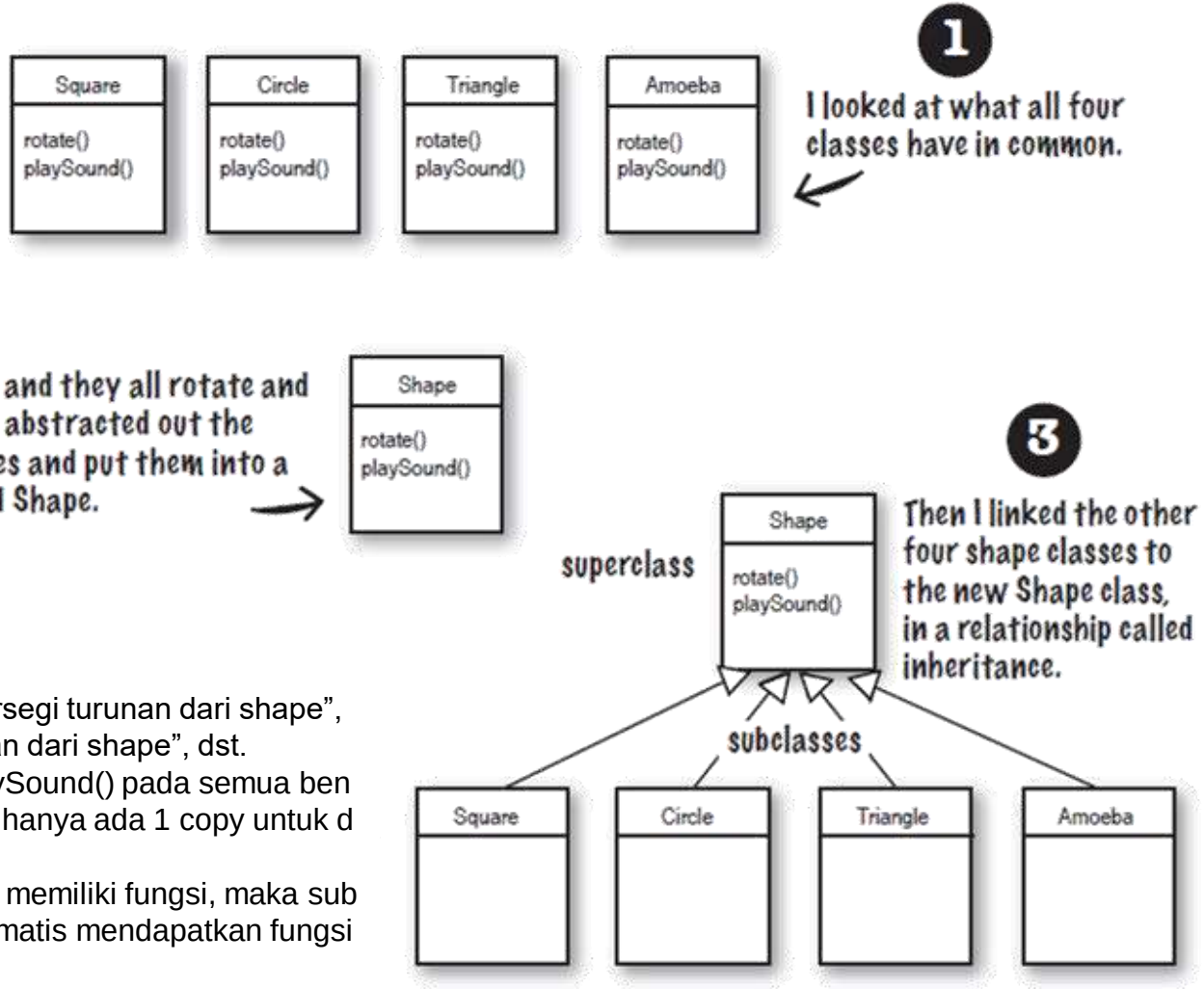
Masih ingat Prosedural VS OO di pembahasan class and object?

Ada duplikasi di code mu! Prosedur rotate ada di semua empat bentuk

Itu bukan prosedur, itu ***method***. Dan keempatnya itu adalah ***class***, bukan bentuk

Whatever! Itu desain yang buruk. Kamu harus me-maintan empat method rotate yang berbeda. Bagaimana itu dikatakan bagus?

Oh... Berarti kamu tidak tahu desain finalnya. Sini aku tunjukkan gimana cara kerja OO inheritance...



- Bisa dibaca “persegi turunan dari shape”, “lingkaran turunan dari shape”, dst.
- rotate() dan playSound() pada semua ben tuk dihapus, jadi hanya ada 1 copy untuk d i-maintain.
- Jika class shape memiliki fungsi, maka sub class secara otomatis mendapatkan fungsi yang sama

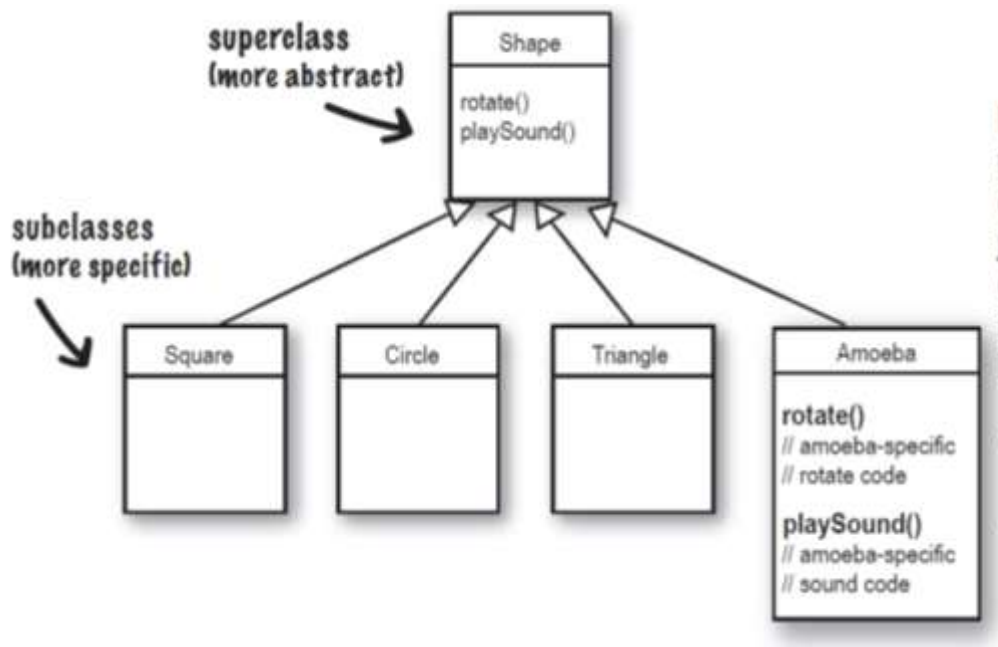
Tapi..

Bukannya masalahnya justru disini? Bentuk amoeba memiliki prosedur rotate dan playSound yang berbeda?!

METHOD

Whatever. Bagaimana bisa amoeba melakukan sesuatu yang berbeda jika dia “inherits” fungsinya dari class Shape?

Itu tahap terakhir. Class amoeba **override** method class shape. Jadi, saat runtime, JVM tahu secara pasti method rotate() mana yang dijalankan ketika seseorang minta amoeba berputar.



4

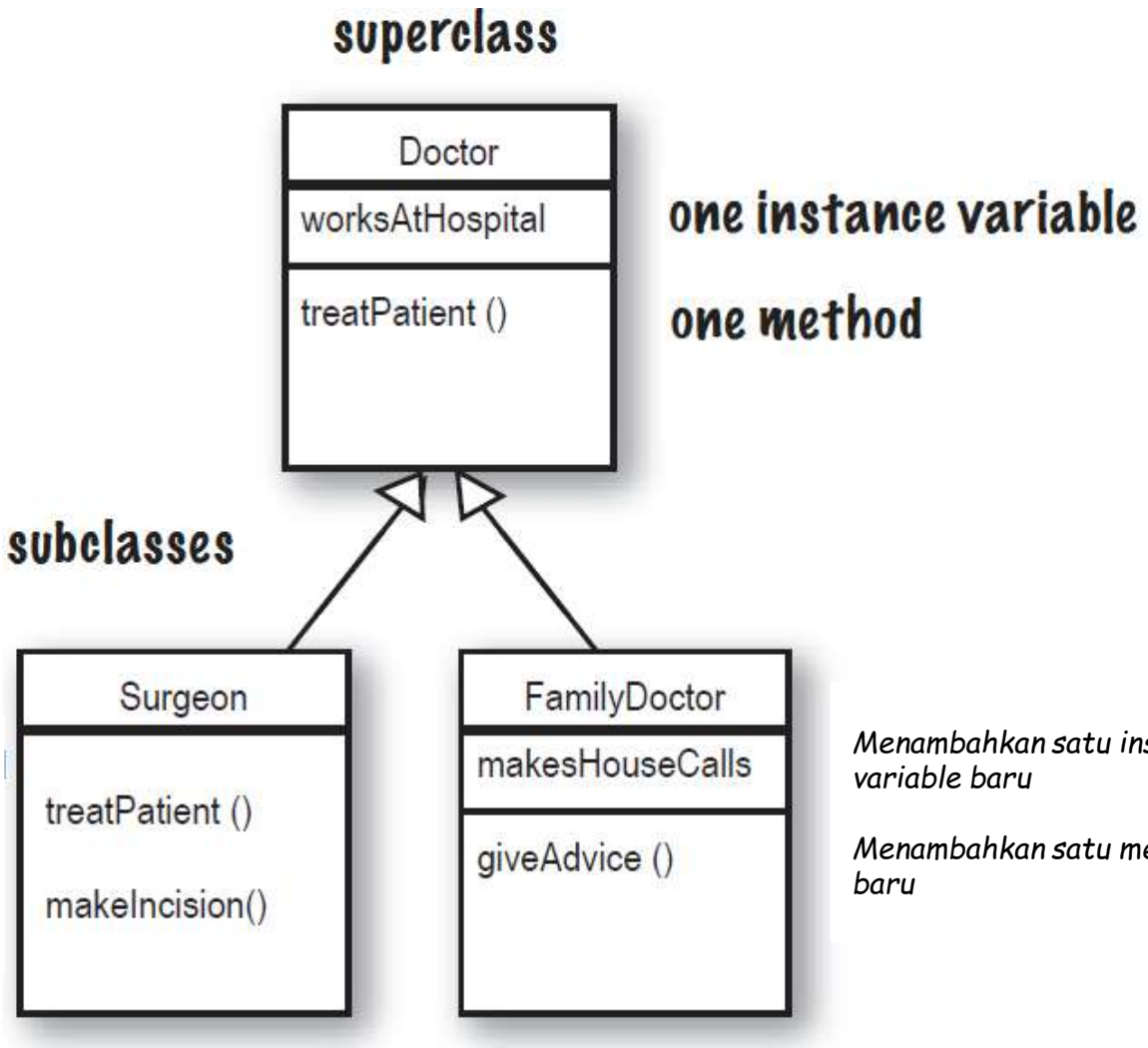
I made the Amoeba class override the rotate() and playSound() methods of the superclass Shape. Overriding just means that a subclass redefines one of its inherited methods when it needs to change or extend the behavior of that method.

Memahami inheritance

inheritance

pewarisan

- Ketika satu class inherit dari class lain, **berarti subclass inherit dari superclass**
- Dalam Java, disebut **subclass extends superclass**
- Hubungan inheritance berarti subclass mewarisi anggota dari superclass, yaitu inst ance variable dan method
- Method memungkinkan di-override, tapi instance variable tidak di-override, karena memang tidak perlu. Instance variable tidak mendefiniskan suatu behavior khusus sehingga subclass dapat memberikan nilai tertentu pada sebuah instance variable yang diwarisi



Override method treatPatien()

Menambahkan satu method baru

Menambahkan satu instance variable baru

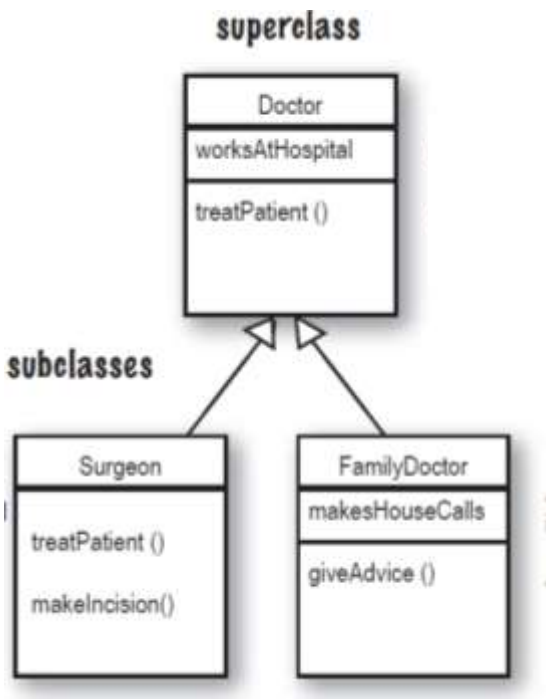
Menambahkan satu method baru

```
14
15      boolean workatHospital;
16
18          //pemeriksaan
19      }
20  }
```

```
12  public class FamilyDoctor extends Doctor {
13
14      boolean makesHouseCalls;
15
16      void giveAdvice() {
17
18      }
19
20  }
```

```
12  public class Surgeon extends Doctor {
13
14      @Override
16      }
17
18      void makeIncision() {
19      }
20  }
```

Apa jawabanmu....



- Berapa banyak instance variabel yang dimiliki Surgeon?
- Berapa banyak instance variabel yang dimiliki FamilyDoctor?
- Berapa banyak method yang dimiliki Doctor?
- Berapa banyak method yang dimiliki Surgeon?
- Berapa banyak method yang dimiliki FamilyDoctor?
- Bisakah FamilyDoctor melakukan treatPatient()?
- Bisakah FamilyDoctor melakukan makeIncision()?

Mendesain inheritance

diskripsi

Bayangkan kamu diminta untuk mendesain sebuah program simulasi dimana user dapat menempatkan banyak ragam hewan pada suatu lingkungan.
Kita telah diberi list beberapa hewan yang akan ada di program. Setiap hewan akan direpresentasikan sebagai objek. Objek tersebut berada pada lingkungan, melakukan apapun yang diprogramkan pada masing-masingnya.
Programmer lain dapat menambahkan satu jenis hewan baru pada program, kapan pun.

Pertama kita harus mencari tahu yang umum, karateristik abstrak yang dimiliki semua hewan dan membangun karateristik-karateristik itu pada sebuah class dimana semua class-class hewan dapat extend padanya.

Mendesain sebuah pohon pewarisan untuk program simulasi hewan

01 Perhatikan objek yang memiliki kesamaan atribut dan behavior

Apa kesamaan enam tipe itu?
Ini akan membantu untuk mengabstraksi behavior

Bagaimana keterkaitan hubungan antar enam tipe itu?
Ini akan membantu mendefiniskan hubungan pohon pewarisan

02 Mendesain sebuah class yang merepresentasikan keadaan (state) dan perilaku (behavior) umum

Ke-enam objek ini adalah hewan, sehingga kita membuat sebuah superclass bernama “Animal”

Tambahkan method dan instance variable yang mungkin dibutuhkan semua hewan

Ingat, penggunaan inheritance adalah untuk mencegah duplikasi code pada subclass

Animal
pitcure food hunger boundaries location
makeNoise() eat() sleep() roam()

State

Terdapat 5 instance variable:

pitcure

nama file yang merepresentasikan gambar JPEG dari hewan

01

food

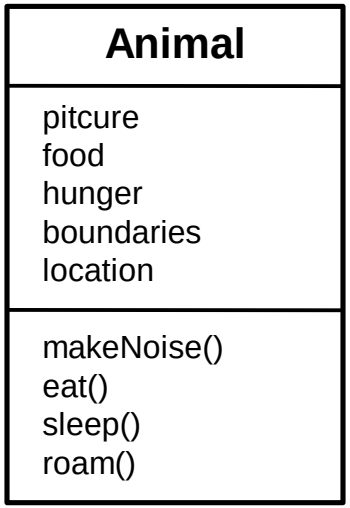
jenis makanan hewan, bisa bernilai meat atau grass

02

hunger

sebuah integer yang merepresentasikan tingkat kelaparan dari hewan, nilainya dapat berubah tergantung kapan dan seberapa banyak hewan makan

03



05

location

koordinat X dan Y dimana hewan berada pada wilayahnya

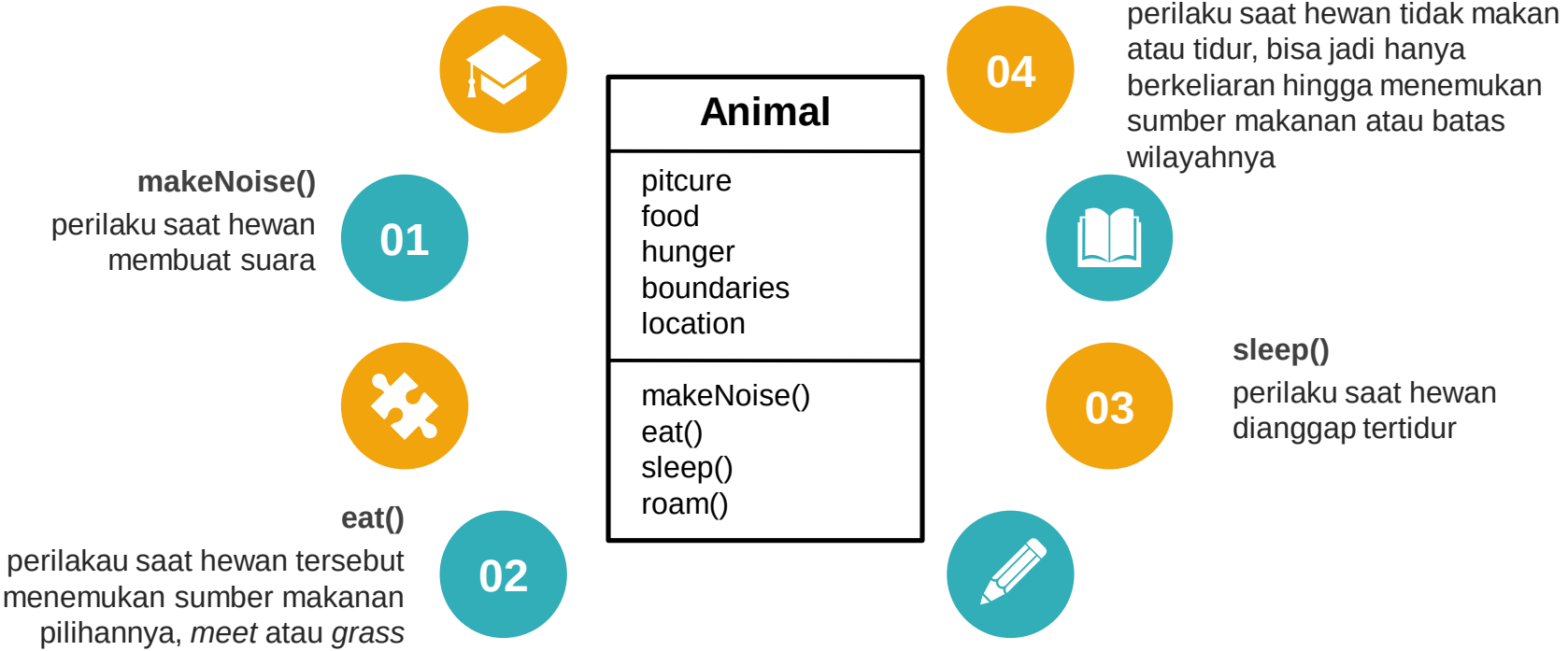
04

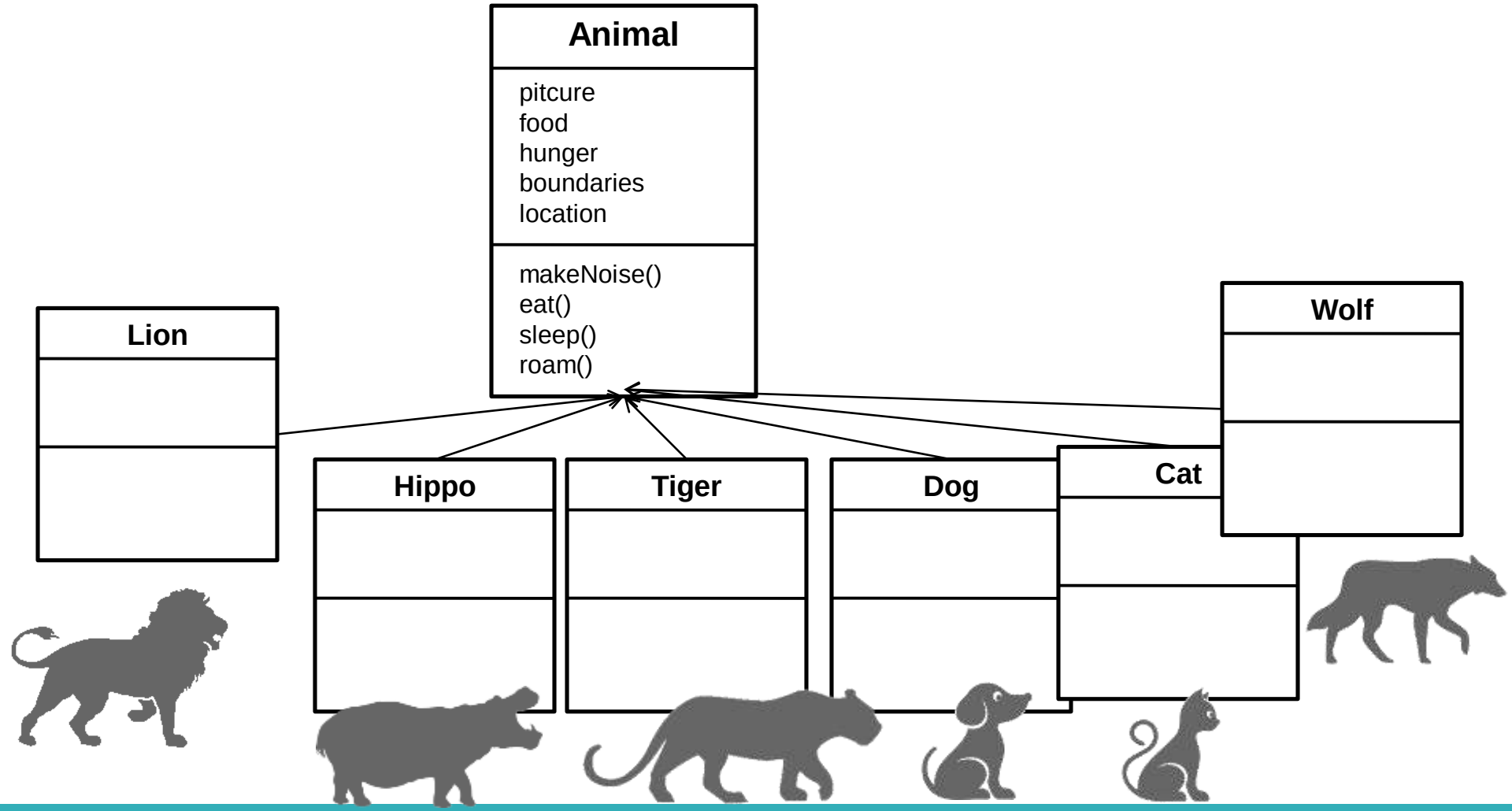
boundaries

nilai yang merepresentasikan panjang dan lebar dari wilayah hewan dapat berkeliaran

Bahavior

Terdapat 4 method:




Animal
pitcure
food
hunger
boundaries
location
makeNoise()
eat()
sleep()
roam()
Lion
Hippo
Tiger
Dog
Cat
Wolf

03 Tentukan apakah sebuah subclass membutuhkan behavior (imlementasi method) yang spesifik tipe subclass tertentu

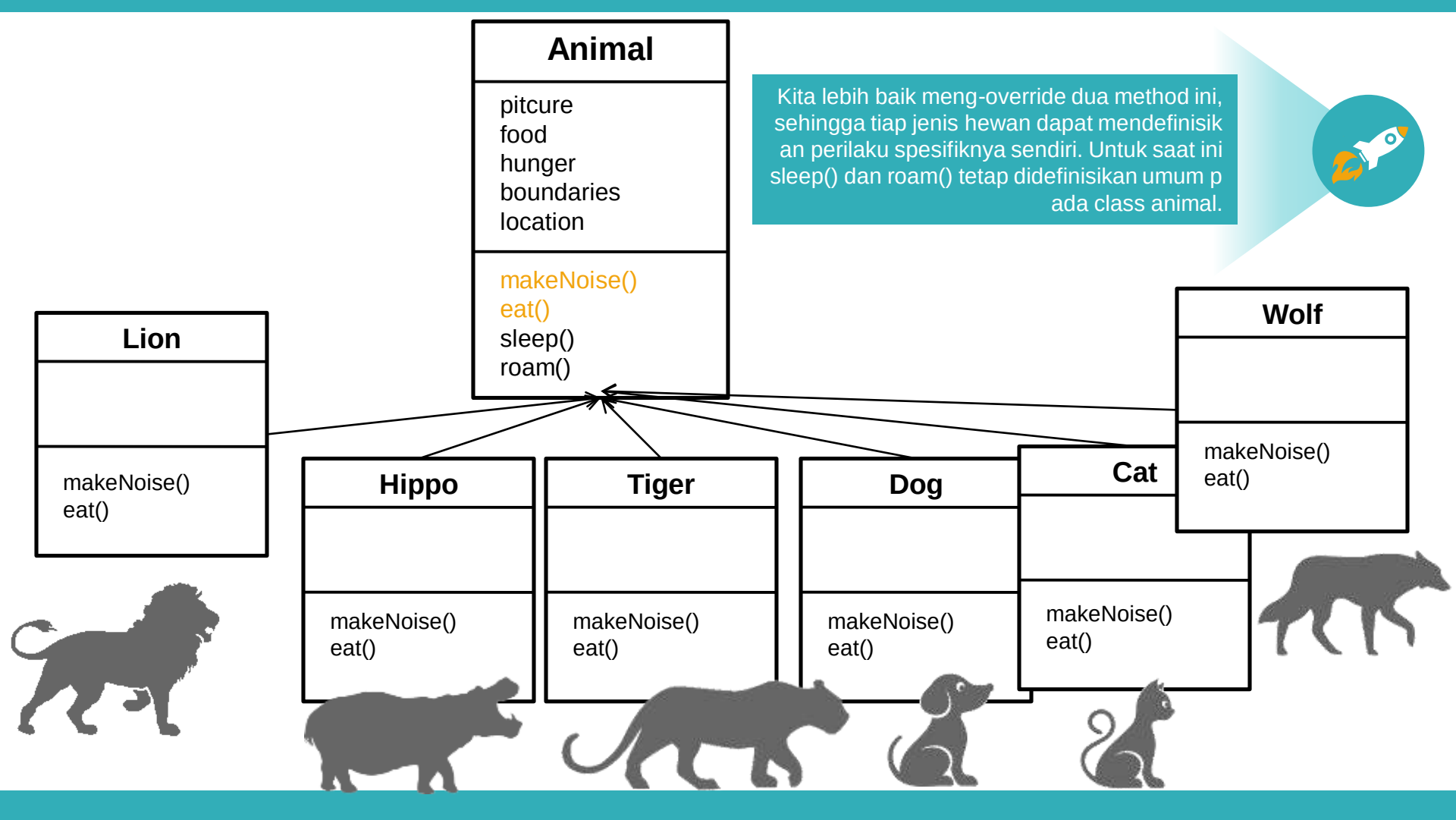
Apakah semua hewan makan dengan cara yang sama?

Kita sepakat bahwa, instance variable akan bekerja pada semua tipe hewan. Seekor singa akan memiliki nilainya sendiri untuk instance variable picture, food, hunger, boundaries, location. Kuda nil akan memiliki nilai yang berbeda, tapi dia tetap memiliki variable yang sama dengan hewan yang lain, anjing, harimau, dst. Lalu bagaimana dengan behavior?

Method mana yang harus kita override?

Apakah singa akan bersuara sebagaimana anjing? Apakah seekor kucing makan seperti kuda nil? Faktanya, makan dan bersuara adalah hal spesifik pada hewan tertentu. Beberapa hewan mungkin membuat suara yang berbeda pada situasi yang berbeda, seperti suara saat makan, saat menabrak musuh, dsb.

Perhatikan class Animal, kita tentukan eat() dan makeNoise() harus di-Override oleh subclass


Animal
pitcure
food
hunger
boundaries
location
makeNoise()
eat()
sleep()
roam()
Kita lebih baik meng-override dua method ini, sehingga tiap jenis hewan dapat mendefinisik an perilaku spesifiknya sendiri. Untuk saat ini sleep() dan roam() tetap didefinisikan umum p ada class animal.
Lion
makeNoise()
eat()
Hippo
makeNoise()
eat()
Tiger
makeNoise()
eat()
Dog
makeNoise()
eat()
Cat
makeNoise()
eat()
Wolf
makeNoise()
eat()

04 Lihat peluang menggunakan abstraksi, dengan mencari dua atau lebih subclass yang dimungkinkan butuh perilaku mirip

Cari peluang dalam pewarisan

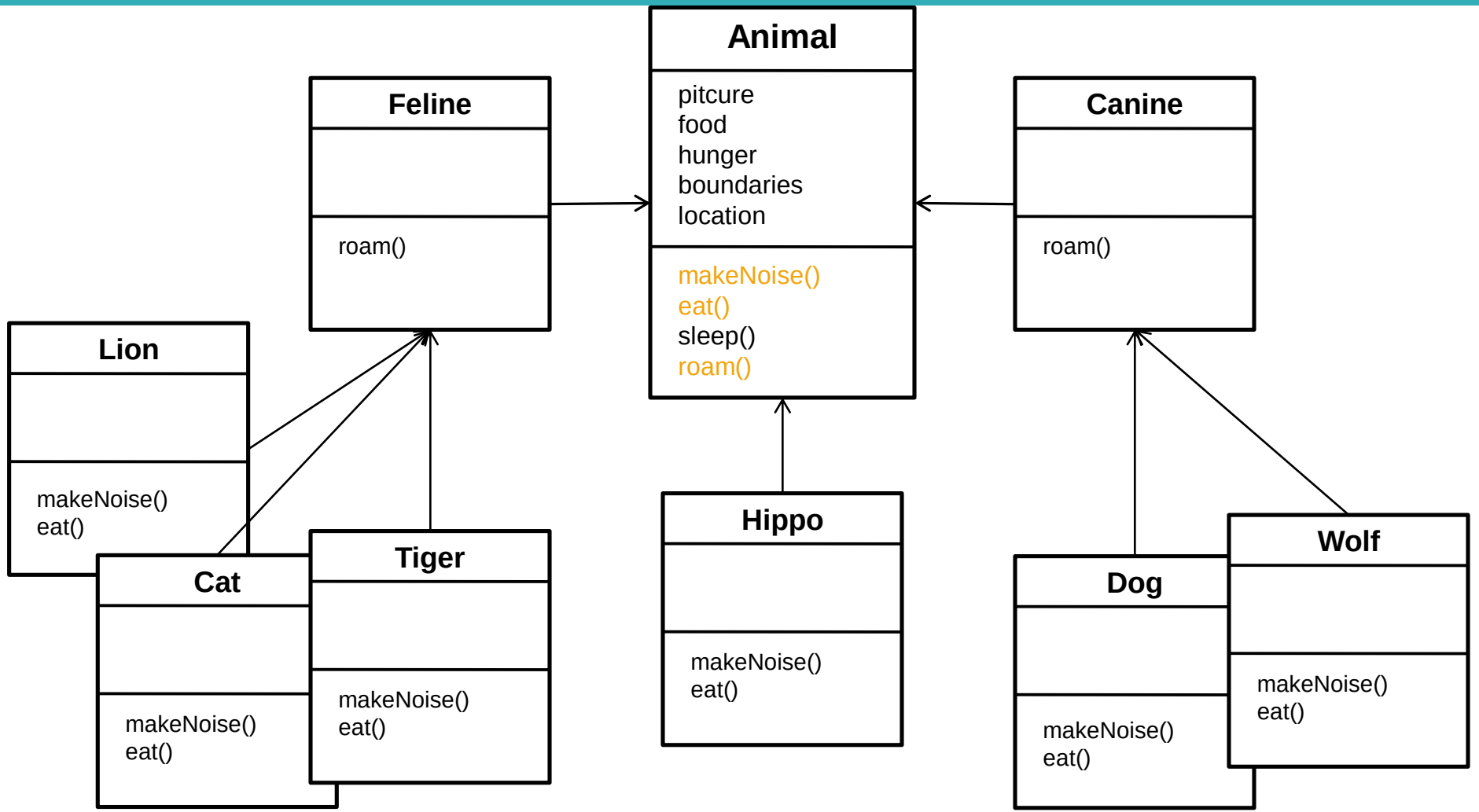
Hirarki class mulai terbentuk. Setiap subclass meng-override method makeNoise() dan eat(). Sehingga jelas gonggongan anjing berbeda dengan meow kucing. Dan seekor kuda nil tidak akan makan seperti singa.

Tapi kita perlu memperhatikan subclass dari Animal . Apakah dua atau lebih subclass dapat digabungkan sedemikian rupa dan diberikan code yang mirip untuk group baru tsb.

Ketika kita melihat class yang ada, tampak bahwa anjing dan serigala dimungkinkan memiliki perilaku yang mirip. Begitu pula dengan singa, harimau, dan kucing.

05 Selesaikan hirarki class

- Karena hewan telah memiliki hirarki sendiri, yaitu kingdom, genus, filum, dst, maka kita dapat menggunakan level yang paling sesuai/masuk akal untuk desain class.
- Disini kita menggunakan “family” secara biologis untuk mengatur hewan. Yaitu dengan menggunakan class Feline (keluarga kucing) dan class Canine (keluarga anjing)
- Canine dapat menggunakan method roam() yang sejenis, karena mereka cenderung bergerak secara berkelompok.
- Kita juga dapati bahwa Feline dapat menggunakan method roam() yang semisal, karena mereka cenderung bergerak menghindari sesama jenisnya.
- Sedangkan hippo, tetap menggunakan method roam() yang diwarkan padanya dari Animal


Animal
pitcure
food
hunger
boundaries
location
makeNoise()
eat()
sleep()
roam()
Feline
roam()
Canine
roam()
Lion
makeNoise()
eat()
Cat
makeNoise()
eat()
Tiger
makeNoise()
eat()
Hippo
makeNoise()
eat()
Dog
makeNoise()
eat()
Wolf
makeNoise()
eat()

tentukan

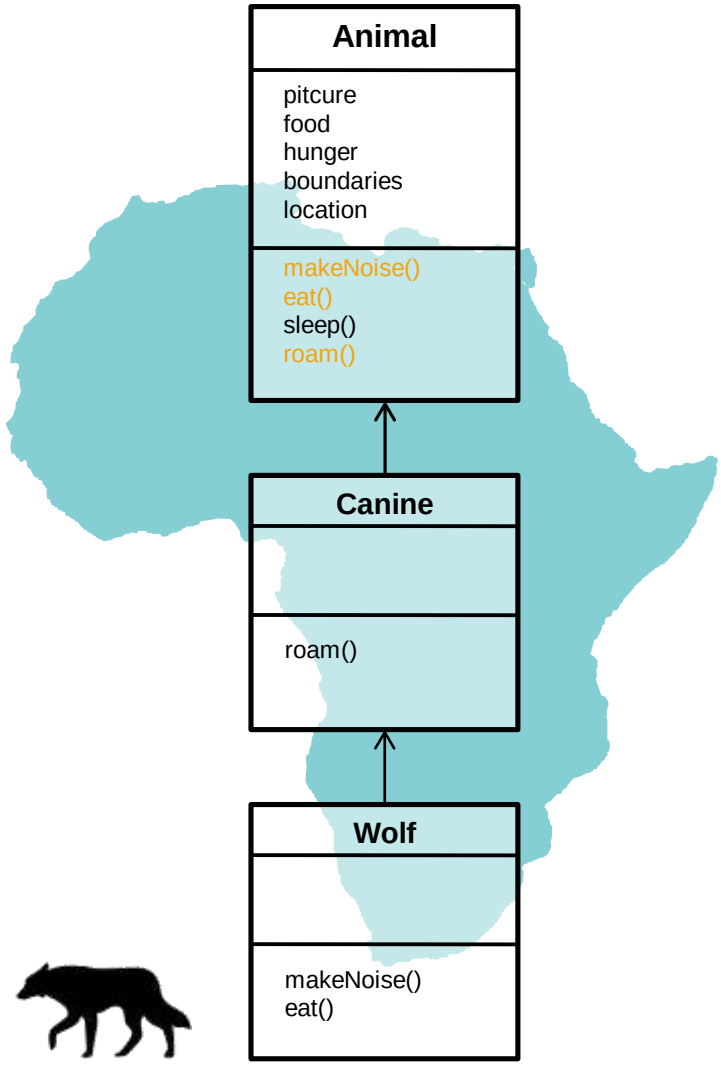
Method

yang dipanggil

Class Wolf memiliki 4 method. Satu method diwarisi dari Animal, satu diwarisi dari Canine (yang juga merupakan method override terhadap class Animal), dan dua method override di class Wolf.

Ketika membuat sebuah objek wolf dan meng-*assign*-nya pada sebuah variable, maka kita dapat menggunakan operator dot pada variable referensi tersebut untuk memanggil ke-empat method tersebut. Tapi, method versi manakah yang terpanggil?

```
Wolf w = new Wolf();

w.makeNoise();

w.roam();

w.eat();

w.sleep();
```

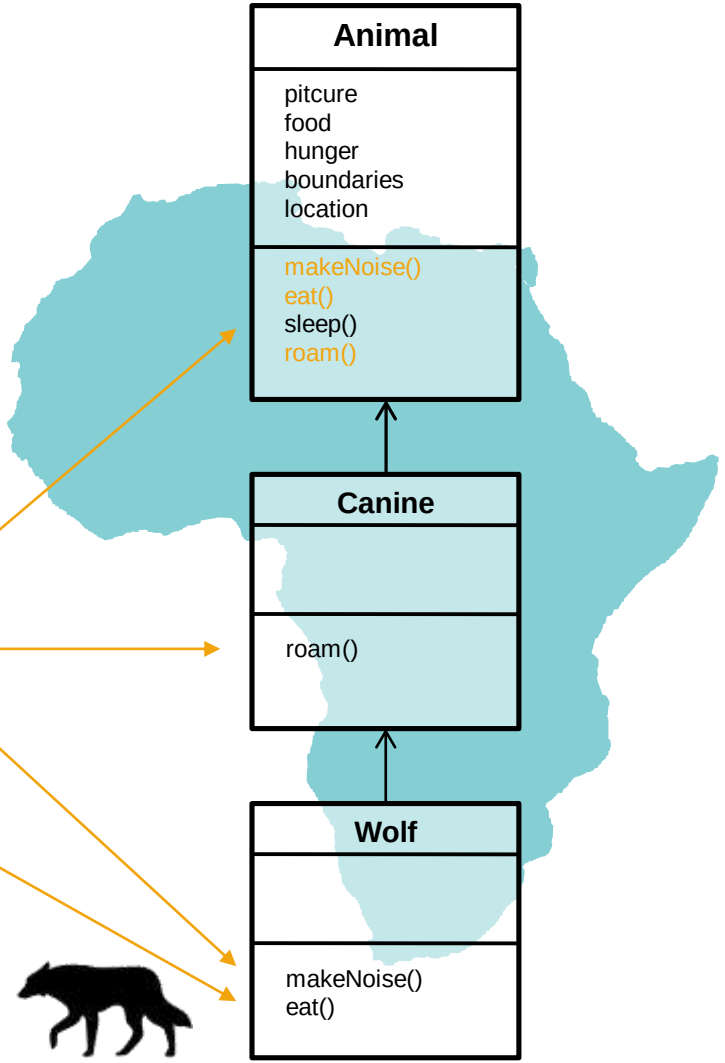


tentukan
Method
yang dipanggil

Ketika memanggil sebuah method pada sebuah referensi objek, itu berarti memanggil versi method yang paling spesifik untuk tipe objek tersebut. Artinya, method paling bawah yang menang. Yaitu, yang “paling bawah” di pohon pewarisan.

Canine dibawah Animal, dan Wolf lebih bawah daripada Canine. Sehingga, memanggil method pada referensi ke sebuah objek Wolf berarti JVM mulai mencari di class Wolf. Jika JVM tidak menemukannya di class Wolf maka ia mencari kembali keatas hirarki pewarisan sampai menemukan kecocokan.

```
Wolf w = new Wolf();

w.makeNoise();

w.roam();

w.eat();

w.sleep();
```

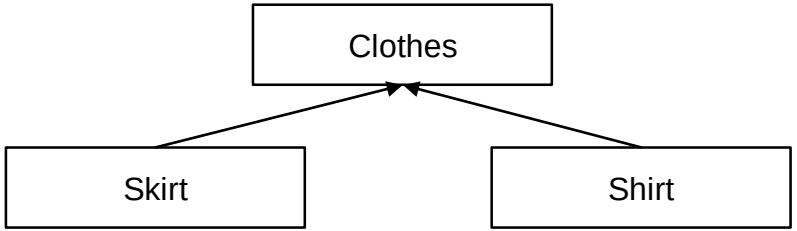


Latihan mendesain (contoh)

Tabel Pewarisan

Class	Superclass	Subclass
Clothes	-	Skirt, Shirt
Skirt	Clothes	
Shirt	Clothes	

Diagram Class Pewarisan



Latihan mendesain

Tabel Pewarisan

Class	Superclass	Subclass
KaryaTulis		
Novel		
Skripsi		
Tesis		
Desertasi		
Artikel		
Cerpen		

Ket: Cari hubungan yang logis dan isi 2 colom disamping class
Anda boleh mengubah atau menambah class yang ada

Diagram Class Pewarisan

?

is-a VS has-a

Hubungan “adalah” atau “memiliki”?

Ingat, saat suatu class mewarisi dari class lain, kita sebut **subclass extends super class**. Untuk mengetahui suatu class benar/perlu extend ke yang lain, maka kita bis a menggunakan “IS-A test”.

- Triangle IS-A shape? → True
- Cat IS-A Feline? → True
- Surgeon IS-A Doctor? → True
- Tub extends Bathroom? → tampaknya OK,
  tapi coba gunakan “IS-A”.
  Tub IS A Bathroom? → False

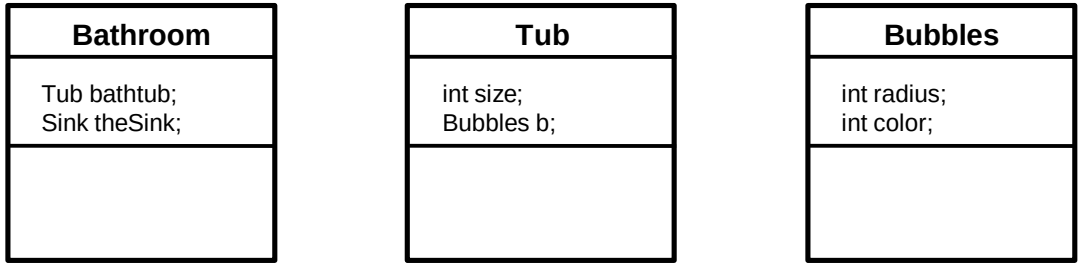
Bagaimana jika dibalik,

- Bathroom extends Tub? → sama saja..tidak cocok
  Bathroom IS-A Tub? → False

Masih tub dan bathroom...

Tub dan bathroom itu berhubungan, tapi bukan pewarisan. Tub dan bathroom dihub ungkan dengan “HAS-A” bukan “IS-A”.

- Bathroom HAS-A Tub? → it can be YES
  jika “iya”, itu berarti bathroom memiliki tub sebagai instance variable, dengan kat a lain, Bathroom memiliki sebuah referensi ke Tub, tetapi bathroom tidak *extend s* ke Tub ataupun sebaliknya.



Bathroom HAS-A Tub dan Tub HAS-A bubbles
Tapi tidak ada yang mewarisi (extends) dari yang lain

IS-A test pada pohon pewarisan

Hingga saat ini, kita dapat menggunakan IS-A test untuk menguji desain pohon pewarisan

> Jika pohon pewarisan didesain dengan bagus, maka hasil IS-A test akan masuk akal, yaitu ketika menanyakan suatu subclass apakah itu adalah sesuatu tipe diatasnya.

Jika class B extend class A, class B IS-A class A. Pada pohon pewarisan, pernyataan tersebut bernilai true. Termasuk pernyataan, jika class C extend class B, class C akan lolos uji IS-A untuk B dan A.

Tapi ingat, hubungan IS-A hanya berlaku searah, tidak kebalikannya.

menguji

desain

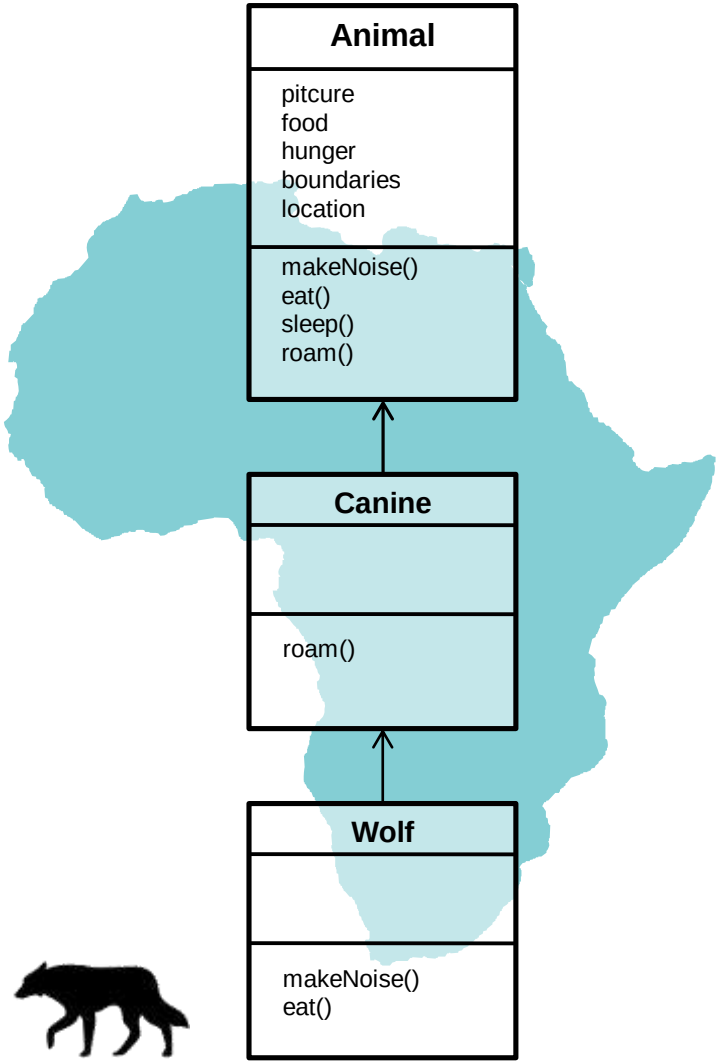
inheritance tree

Pada pohon pewarisan seperti disamping, dapat dikatakan bahwa “wolf extends animal” atau “wolf IS-A animal”. Meskipun animal adalah superclass dari superclass-nya wolf. Selama Animal pada hirarki pewarisan diatas wolf, maka wolf adalah animal akan selalu bernilai true.

Dapat kita katakan bahwa “wolf IS-A canine, maka wolf dapat melakukan apapun yang dilakukan canine. Dan wolf IS-A animal, maka wolf dapat melakukan apapun yang dapat dilakukan oleh animal”. Hal ini meski wolf melakukan override beberapa method pada Animal atau Canine.

Canine extends Animal
Wolf extends Canine
Wolf extends Animal

Canine IS-A Animal
Wolf IS-A Canine
Wolf IS-A Animal



Apa jawabanmu....

Beri tanda (check) pada hubungan yang sesuai dan sertai alasan untuk hubungan yang tidak sesuai

- ❑ Oven extends Dapur
- ❑ Gitar extends Instrumen
- ❑ Orang extends Karyawan
- ❑ Ferrari extends Mesin
- ❑ TelurGoreng extends Makanan
- ❑ AnjingPemburu extends Peliharaan
- ❑ Wadah extends Toples
- ❑ Logam extends Titanium
- ❑ BerambutPirang extends Cerdas

Access levels

Mengontrol “siapa melihat apa”

Access levels

Terdapat 4 tingkat akses

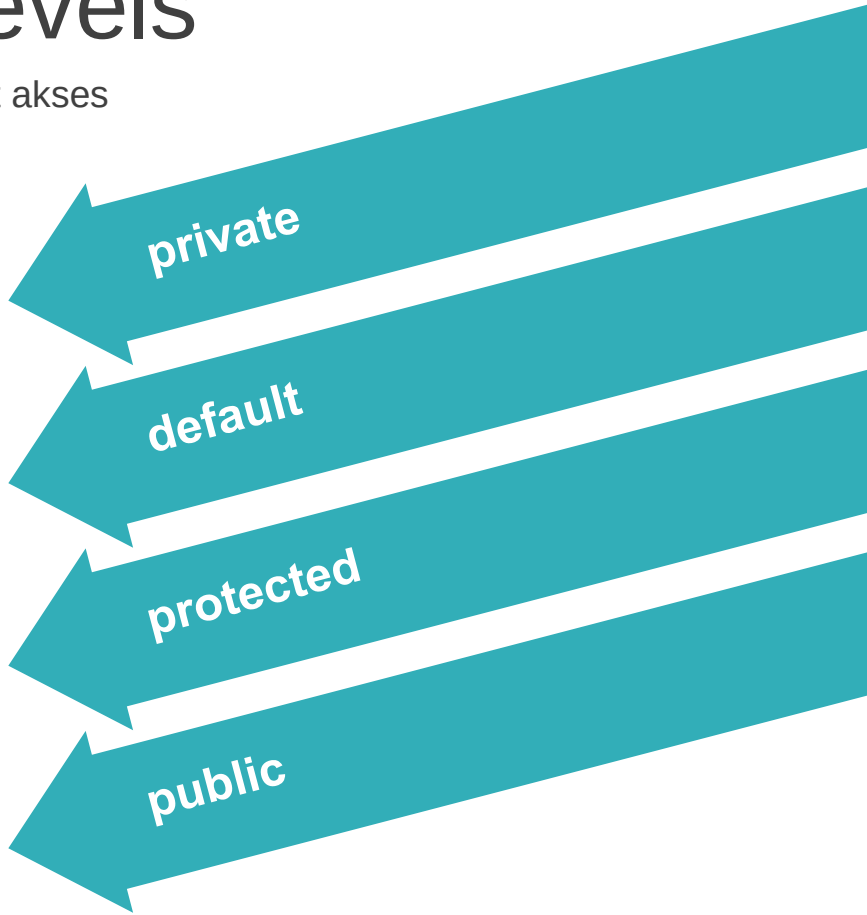
Bagaimana mengetahui apa yang dapat subclass warisi dari superclass-nya?

Sebuah subclass mewarisi anggota dari superclass, baik instance variable maupun method. Superclass dapat memilih apakah ia ingin subclass mewarisi a nggota tertentu berdasarkan tingkat akses yang dib erikan anggota tertentu.

Tingkat akses mengontrol “siapa melihat apa”, dan merupakan hal penting untuk mendapatkan kode java dengan desain yang baik dan kuat.

Kali ini fokus pembasahan pada private dan public. Aturan dari keduanya adalah:

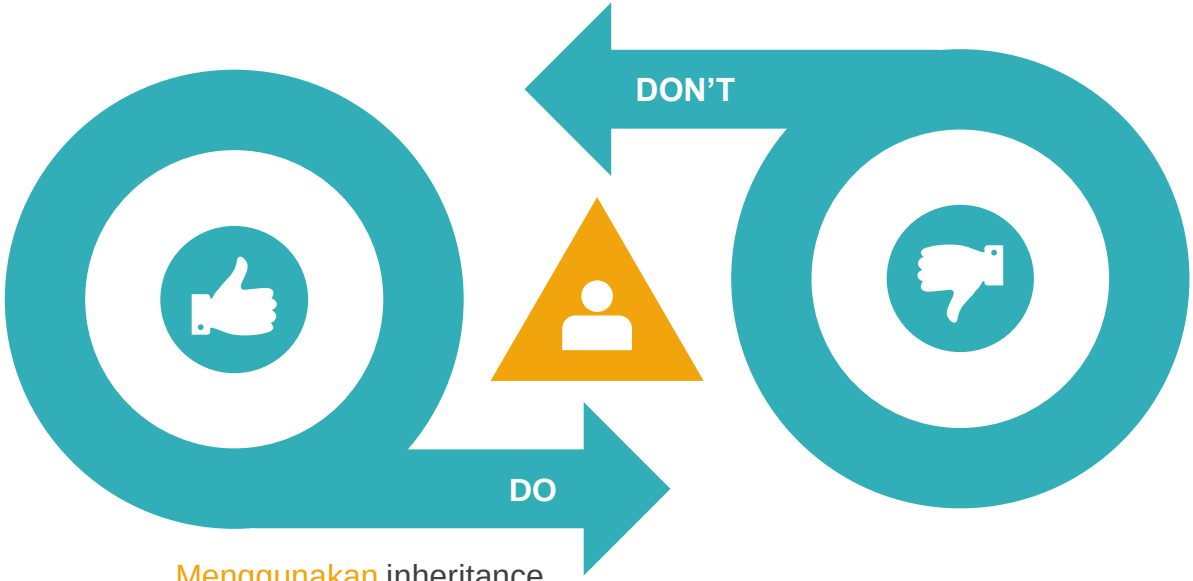
Anggota bersifat public akan diwariskan
Anggota bersifat private tidak diwariskan



Beberapa Aturan

Untuk membangun desain pewarisan yang baik

Bukan menyalahgunakan inheritance



Menggunakan inheritance

Beberapa Aturan

Untuk membangun desain pewarisan yang baik

Menggunakan inheritance

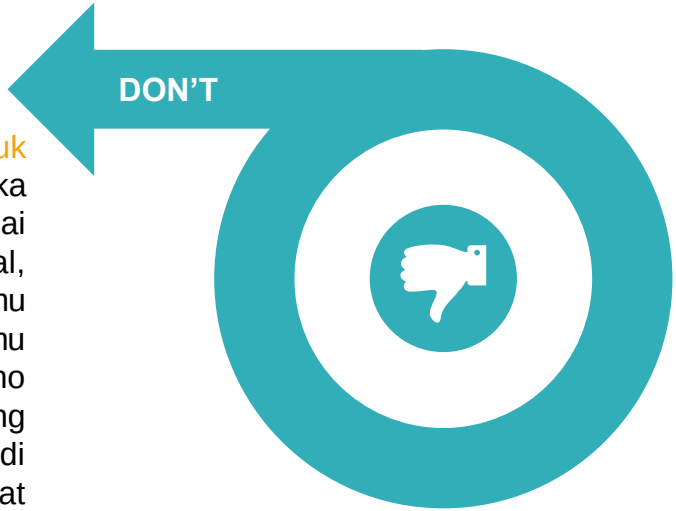
Gunakan inheritance ketika satu class adalah tipe yang lebih spesifik dari superclass. Misal, kucing adalah sesuatu yang lebih spesifik dari hewan, maka logis jika cat extends animal

Inheritance dapat dipertimbangkan untuk digunakan ketika terdapat behavior yang perlu dibagi diantara beberapa class setipe. Misal, baik pada Square, Circle, dan Triangle semua memerlukan rotate dan playSound, maka dapat diterima jika meletakkannya di superclass Shape sehingga memudahkan dalam maintenace dan ekstensibilitas.

Beberapa Aturan

Untuk membangun desain pewarisan yang baik

Bukan menyalahgunakan inheritance

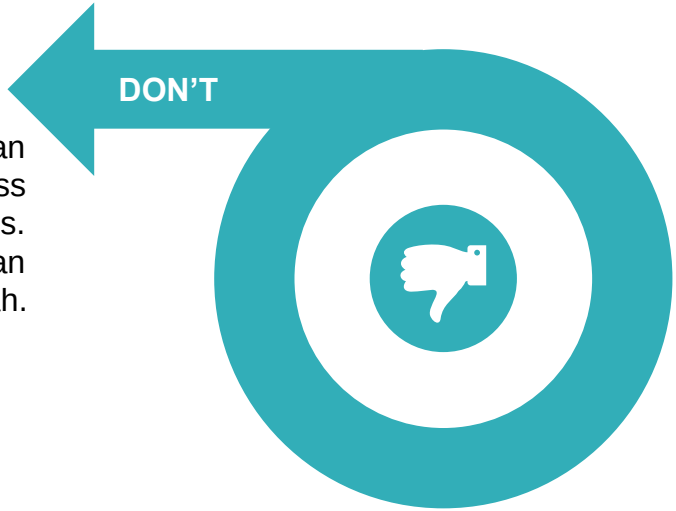


Jangan menggunakan inheritance hanya untuk menggunakan kembali code dari class lain, jika hubungan antara superclass dan subclass tidak sesuai dengan salah satu dari dua aturan sebelumnya. Misal, kamu menulis code cetak pada class Alarm dan kamu perlu code cetak pada class Piano, sehingga kamu membuat Piano extend Alarm yang artinya Piano mewarisi code cetak. Itu salah! Piano bukan tipe yang lebih spesifik dari Alarm. Seharusnya code cetak ada di class Printer (misalkan), dan semua objek yang dapat dicetak dapat memanfaatkannya dengan hubungan HAS-A

Beberapa Aturan

Untuk membangun desain pewarisan yang baik

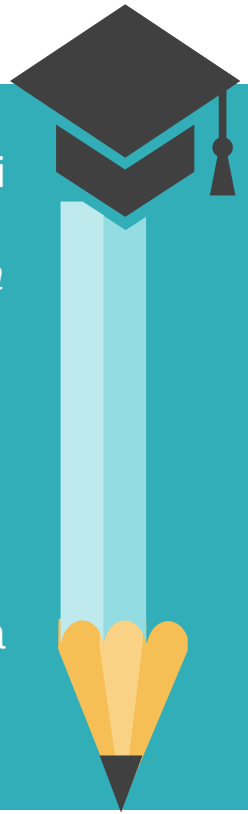
Bukan menyalahgunakan inheritance



Jangan menggunakan inheritance jika subclass dan superclass tidak lolos uji IS-A. Pastikan apakah subclass adalah tipe yang yang lebih spesifik dari superclass. Misal, Teh adalah Minuman, itu benar. Sedangkan Minuman adalah Teh, itu salah.

catatan

Poin-poin pembahasan inheritance



- subclass *extends* superclass
- Subclass mewarisi semua *public* instance variabel dan method, tetapi tidak mewarisi *private* instance variable dan method dari superclass
- Method yang diwariskan *dapat* di override, sedangkan instance variable *tidak dapat* di override (meskipun dapat didefinisikan ulang pada subclass)
- Gunakan *IS-A test* untuk memastikan kevalidan hirarki pewarisan. Jika X extends Y, maka X is-A Y harus bernilai true
- Hubungan IS-A hanya bekerja *satu arah*. Misal, kuda nil adalah hewan, tapi tidak semua hewan adalah kuda nil.
- Ketika sebuah method di override pada subclass dan method tersebut dipanggil oleh instance dari subclass, maka method yang di-override yang dipanggil (paling bawah yang menang)
- Jika class B extends A, dan C extends B, maka class B IS-A class A, class C IS-A class B, begitu juga C IS-A class A

Jadi...

Keuntungan menggunakan inheritance adalah

Mencegah duplikasi code

Letakkan code pada satu tempat dan subclass akan mewarisi code tersebut dari superclass.
Dengan merubah superclass maka semua class yang extend padanya secara otomatis akan menggunakan versi terbaru.

Dapat menentukan protokol umum untuk sekelompok class

Inheritance menjamin bahwa semua class yang dikelompokkan dalam supertipe tertentu memiliki semua method yang dimiliki supertipe.
Ketika mendefinisikan suatu supertipe untuk sekelompok class, maka subclass dari supertipe tersebut dapat disubtitusi dengan supertipe.
Dengan kata lain, anda dapat merujuk ke objek subclass menggunakan referensi yang dideklarasikan sebagai supertipe.

polimorfisme

Polymorphism

Bagaimana cara kerja polimorfisme?

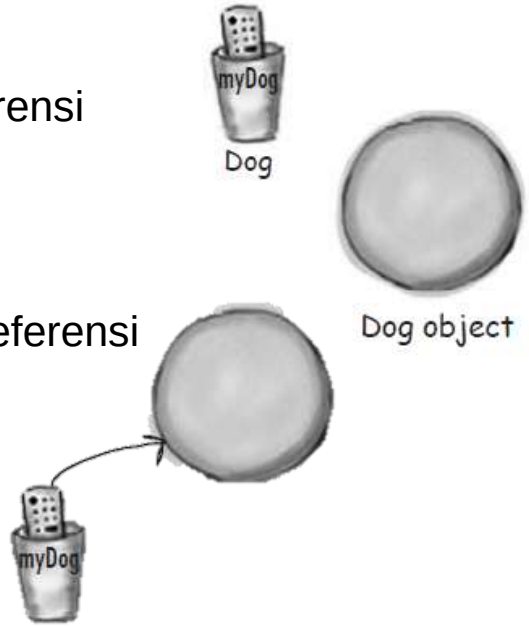
Ingat, bagaimana mendeklarasikan sebuah referensi dan membuat sebuah objek.

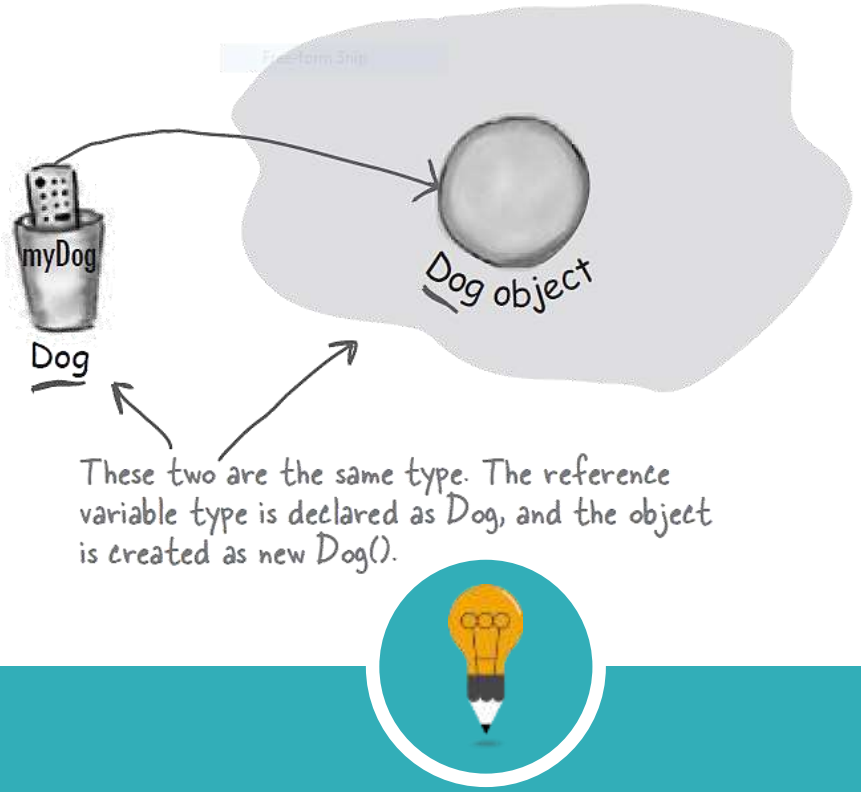
Tiga tahap deklarasi objek, membuat, dan *assigment*

```
Dog myDog = new Dog();
```

1 = Dog myDog
2 = new Dog()
3 = =

1. Mendeklarasikan sebuah variabel referensi
2. Membuat sebuah objek
3. Menghubungkan (link) objek dengan referensi



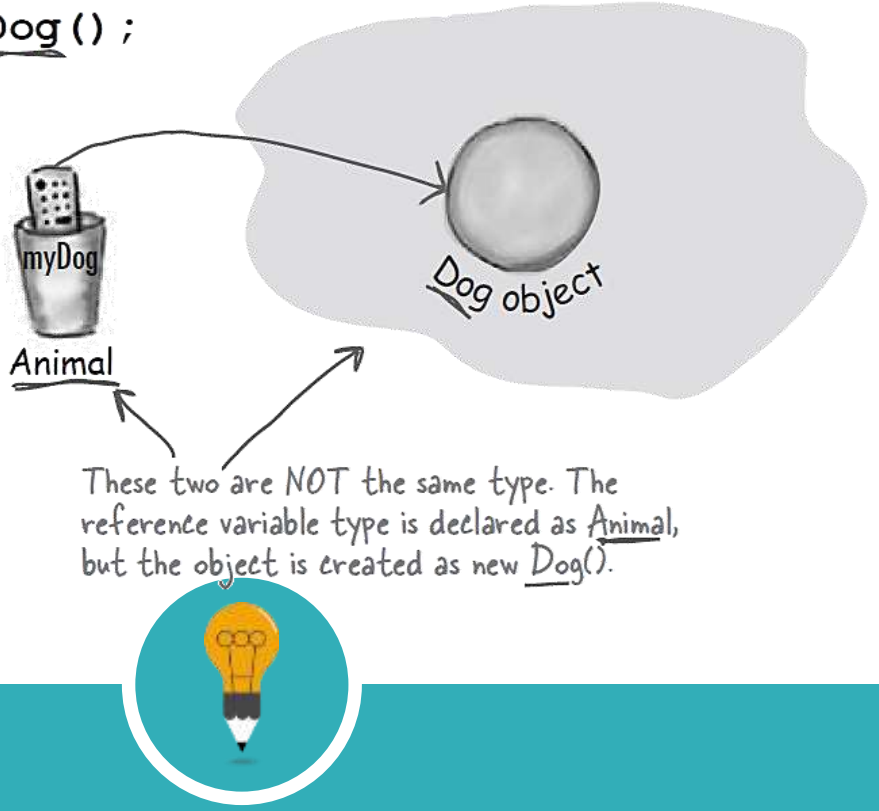


“

tipe referensi dan tipe objek sama

”

```
Animal myDog = new Dog();
```



Tapi, dengan polimorfisme referensi dan objek bisa jadi **berbeda**

Pada polimorfisme

Tipe referensi bisa jadi adalah superclass dari tipe objek yang sebenarnya

apapun yang extends pada tipe variable referensi yang dideklarasikan dapat di-assign pada variabel referensi.

Dengan begitu maka dimungkinkan membuat semacam array polimorfik.

Mendeklarasikan sebuah array bertipe Animal. Yaitu sebuah array yang akan m enangani objek bertipe Animal

```
Animal[] animal = new Animal[5]
animals[0] = new Dog();
animals[1] = new Cat();
animals[2] = new Wolf();
animals[3] = new Hippo();
animals[4] = new Lion();
```

Tapi lihat, kamu dapat menuliskan subclass Animal yang mana saja pada array Animal

```
for (int i=0; i<animals.lenght; i++){

  animals[i].eat();

  animals[i].roam();

}
```

Dan pada perulangan ini, kamu dapat memanggil method class Animal

Saat "i" bernilai 0, yaitu Dog pada indeks 0 array, maka method yang dijalankan adalah method eat() milik Dog. Sedangkan saat "i" bernilai 1, maka yang dijalankan adalah method eat() milik Cat

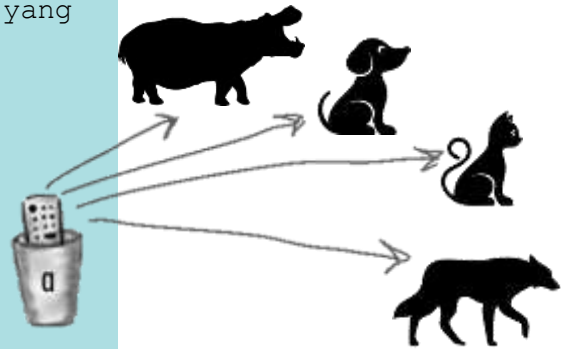
...dan

Tipe return dan argument polimorfik

Kita dapat mendeklarasikan sebuah variabel referensi dari tipe super, misal Animal, dan menugaskan (*assign*) sebuah objek subclass padanya, misal Dog. Lalu bagaimana jika referensi adalah sebuah argumen ke method...

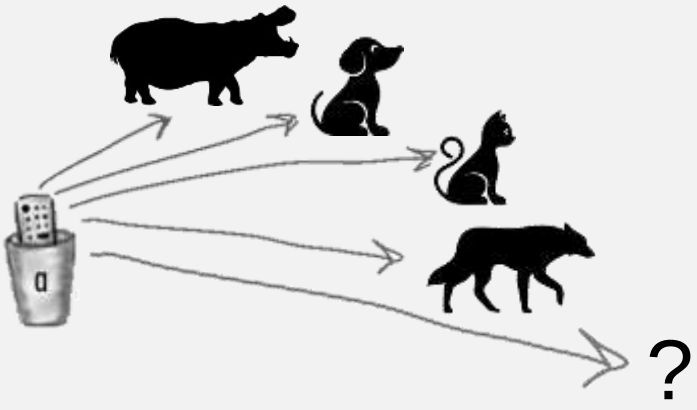
Parameter Animal dapat mengambil tipe animal apapun sebagai argumen.
Ketika Vet (dokter hewan) selesai melakukan suntikan, ia menyampaikan pada Animal untuk makeNoise(). Dan Animal apapun yang berada pada heap, itulah yang method makeNoise()-nya akan dijalankan

```
class Vet {
  public void giveShot(Animal a) {
   // lakukan sesuatu pada Animal yang
   // ditunjuk oleh parameter 'a'
   a.makeNoise();
  }
}


class PetOwner {
  public void start() {
   Vet v = new Vet();
   Dog d = new Dog();
   Hippo h = new Hippo();

   v.giveShot(d);

   v.giveShot(h);
  }
}
```



Method makeNoise() milik Dog yang dijalankan

Method makeNoise() milik Hippo yang dijalankan



Dengan polimorfisme

Kamu dapat menulis kode yang tidak berubah ketika kamu memberikan tipe subclass baru ke program

Contoh pada class Vet, ketika menulis class Vet menggunakan argumen yang dideklarasikan sebagai tipe Animal, maka code tersebut dapat menangani subclass Animal yang manapun. Artinya, jika pihak lain ingin memanfaatkan class Vet, yang perlu dilakukan adalah memastikan bahwa tipe Animal yang baru extends pada class Animal. Method Vet akan tetap bekerja. Meskipun class Vet ditulis tanpa mengetahui subtipe Animal yang baru, Vet tetap dapat berfungsi untuk subtipe tersebut.

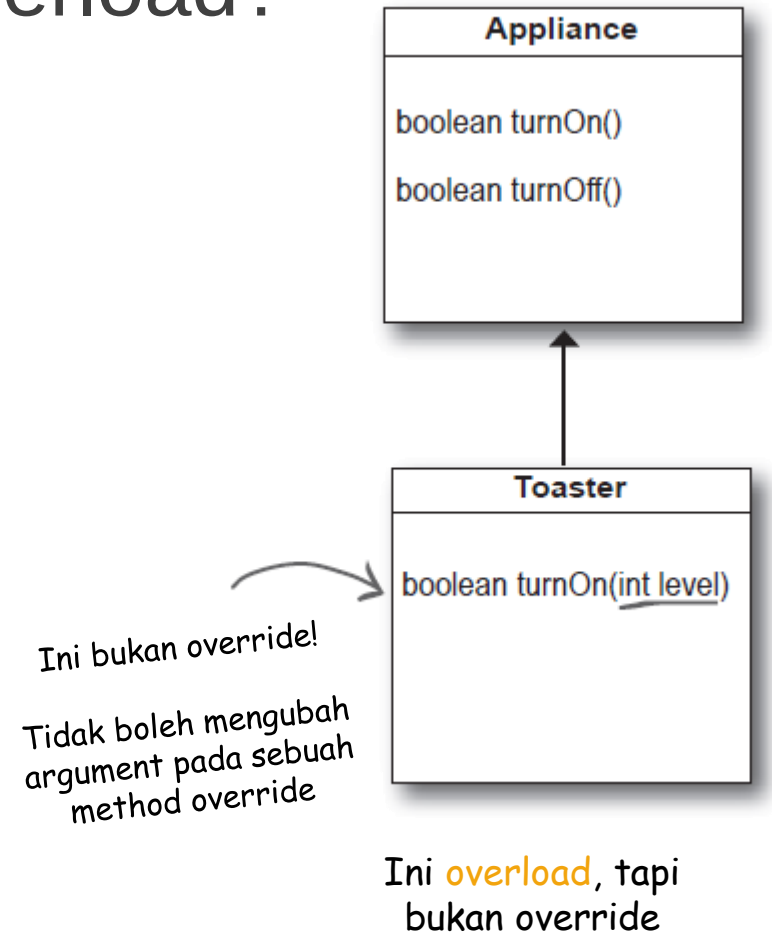
Keuntungan menggunakan polimorfisme

Overriding – overloading method

override atau overload?

Ketentuan yang harus dipenuhi saat melakukan override sebuah method dari suatu superclass adalah: argument dan tipe return dari method yang meng-override harus tampak sama persis dengan method yang di override pada superclass. Dengan kata lain method override harus memiliki argumen dan tipe return yang sama.

Ingat, pada polimorfisme, method yang dipanggil adalah “method yang paling bawah”. Tapi untuk contoh disamping, method turnOn() tanpa argument di class Appliance yang akan dijalankan. Meskipun ada method turnOn(int level) di class Toaster. Ini karena method turnOn(int level) bukan method override.

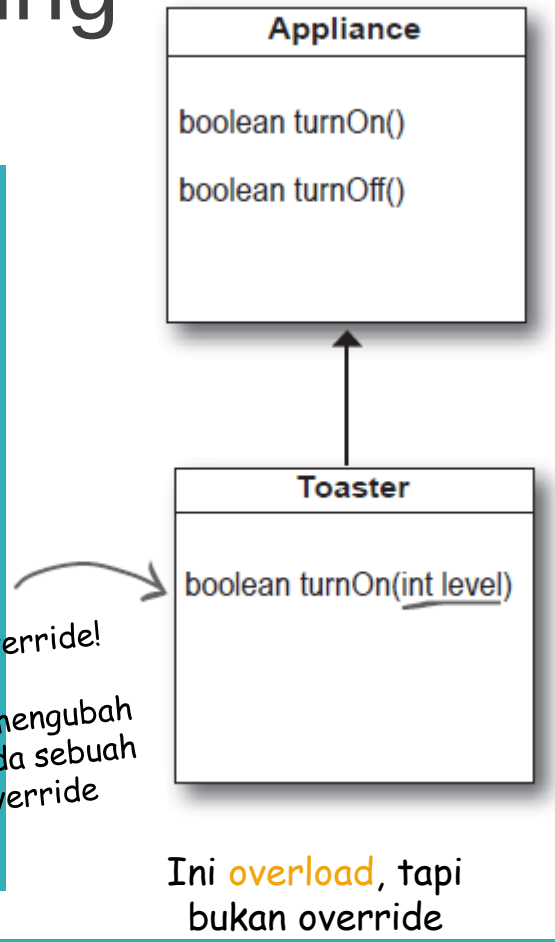


Aturan untuk overriding

01

Argumen harus sama dan tipe return harus sesuai

Superclass mendefinisikan bagaimana code lain menggunakan method. Apakah superclass menggunakan argumen, sehingga subclass yang meng-override method harus menggunakan argumen yang sama. Dan apakah superclass dideklarasikan sebagai sebuah tipe return, sehingga method yang meng-override harus dideklasikan baik dengan tipe sama atau sebuah tipe subclass. Ingat, sebuah objek subclass dipastikan mampu melakukan apapun yang dideklarasikan oleh superclass, jadi dibolehkan untuk mengembalikan tipe subclass.



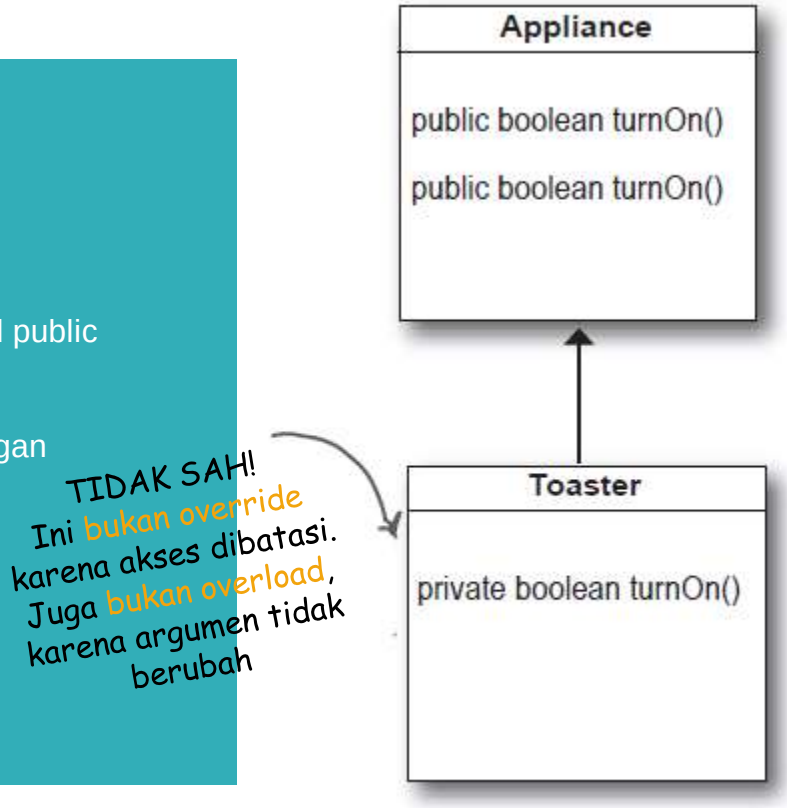
Aturan untuk overriding

02

Level akses method tidak boleh kurang

Artinya, level akses harus sama atau atau “lebih longgar”. Sehingga tidak dibolehkan misalnya meng-override method public dengan method private.

Ada juga aturan lain tentang overriding yang berkaitan dengan exception handling (*nanti kita bahas ya... semoga*)



tentang verloading a method

Method overloading yaitu ketika dua method (atau lebih) memiliki nama sama tetapi berbeda list argumen. Tidak ada hubungan dengan pembahasan inheritance dan polimorfisme. Overloading tidak sama dengan overriding.

Overloading memberikan kesempatan untuk membuat banyak versi dari sebuah method, dengan list argument yang berbeda. Ini untuk menfasilitasi pemanggil. Contoh, jika mempunyai method yang hanya menangani tipe int, maka pemanggil harus mengconvert, misal double ke int sebelum memanggil method tsb. Tapi jika method tsb di-overload dengan versi lain yang menangani tipe double, maka hal ini akan memberi kemudahan bagi pemanggil

Karena method overload tidak ada kaitannya dengan aturan polimorfisme yang didefinisikan oleh superclass, maka aturan method overload lebih fleksibel.

aturan-aturan

verloading a method

01

Tipe return boleh berbeda

Anda bebas merubah tipe return pada method overload selama list argument berbeda

02

Jangan hanya merubah tipe return

Jika hanya tipe return yang berbeda maka itu buka overload yang valid. Compiler akan mengira Anda melakukan override method. Untuk melakukan overload sebuah method, Anda juga harus merubah list argumen

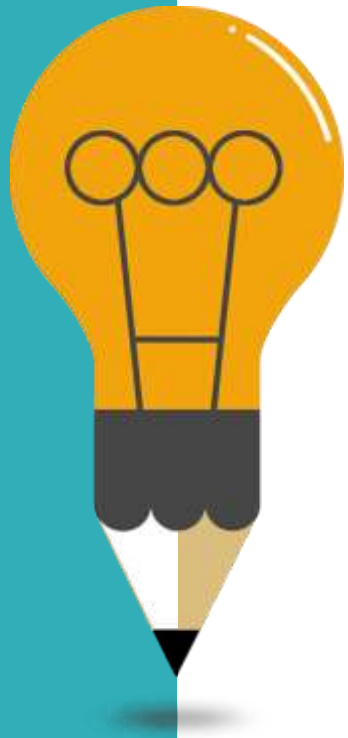
03

Level akses dapat berbeda

Anda bebas meng-overload sebuah method dengan method yang aksesnya lebih terbatas

```
public class Overloads {
  String uniqueID;
  public int addNums(int a, int b) {
    return a + b;
  }
  public double addNums(double a, double b) {
    return a + b;
  }
  public void setUniqueID(String theID) {
    // lots of validation code, and then:
    uniqueID = theID;
  }
  public void setUniqueID(int ssNumber) {
    String numString = "" + ssNumber;
    setUniqueID(numString);
  }
}
```

...akhirnya



01 **Inheritance**

Tentang pewarisan, mendesain pohon pewarisan, menguji desain

02 **Access levels dan beberapa aturan pewarisan**

03 **Polymorphism**

Cara kerja polimorfisme, tipe return dan argumen polimorfik

04 **Overriding – Overloading method**

Tugas

01 coding

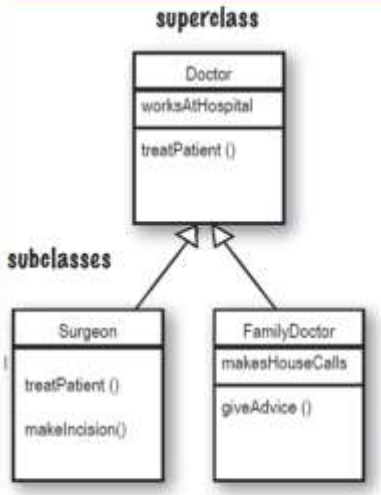
Tulis penggalan source-code Animal pada materi ini. Lengkapi sesuai desain yang telah dibuat. Jalankan program dan pahami.

Anda boleh manambahkan dan memodifikasi dari yang ada

Tugas

02

Apa jawabanmu....



- Berapa banyak instance variabel yang dimiliki Surgeon?
- Berapa banyak instance variabel yang dimiliki FamilyDoctor?
- Berapa banyak method yang dimiliki Doctor?
- Berapa banyak method yang dimiliki Surgeon?
- Berapa banyak method yang dimiliki FamilyDoctor?
- Bisakah FamilyDoctor melakukan treatPatient()?
- Bisakah FamilyDoctor melakukan makeIncision()?

Tugas

03

Latihan mendesain

Tabel Pewarisan

Class	Superclass	Subclass
KaryaTulis		
Novel		
Skripsi		
Tesis		
Desertasi		
Artikel		
Cerpen		

Ket: Cari hubungan yang logis dan isi 2 colom disamping class
Anda boleh mengubah atau menambah class yang ada

Diagram Class Pewarisan

Tugas

04

Apa jawabanmu....

Beri tanda (check) pada hubungan yang sesuai dan sertai alasan untuk hubungan yang tidak sesuai

- ❑ Oven extends Dapur
- ❑ Gitar extends Instrumen
- ❑ Orang extends Karyawan
- ❑ Ferrari extends Mesin
- ❑ TelurGoreng extends Makanan
- ❑ AnjingPemburu extends Peliharaan
- ❑ Wadah extends Toples
- ❑ Logam extends Titanium
- ❑ BerambutPirang extends Cerdas

Alhamdulillah

@KhadijahHolle