Pemrograman Berorientasi Obyek

Methods

anton@ukdw.ac.id

Tanya jawab (1)

The keyword private restricts the access of class to:

A. **const functions**
B. **static functions**
C. **member functions**
D. **clients**

What is the difference between an object and a class?

A.	An object is an extension of the class construct whose default access privilege is public.
B.	The term *object* is just another way of referring to the public data members of a class.
C.	An object is an initialized class variable.
D.	A class is an initialized object variable.

Tanya jawab (2)

```
public class Card {
    public int s;
    ...
}

Card a = new Card();
```

Cara untuk mengakses s adalah :

A. p.s

B. a.s

C. Card.s

D. card.s

```
public class Contoh {
        private int nomor;
        Contoh() {
                nomor = 100;
        }
}
...
Contoh c = new Contoh();
```

Untuk mengubah nomor menjadi 200 pada object c, apa yang harus dilakukan ?

Java Naming conventions

- ***Package names***: start with lowercase letter
  - E.g. java.util, java.net, java.io . . .
- ***Class names***: start with uppercase letter
  - E.g. File, Math . . .
  - avoid name conflicts with packages
  - avoid name conflicts with standard keywords in java system
- ***Variable***, ***field and method names***: start with lowercase letter
  - E.g. x, out, abs . . .
- ***Constant names***: all uppercase letters
  - E.g. PI . . .
- ***Multi-word names***: capitalize the first letter of each word after the first one
  - E.g. HelloWorldApp, getName . . .
- ***Exception class names***: (1) start with uppercase letter (2) end with “Exception” with normal exception and “Error” with fatal exception
  - E.g. OutOfMemoryError, FileNotFoundException

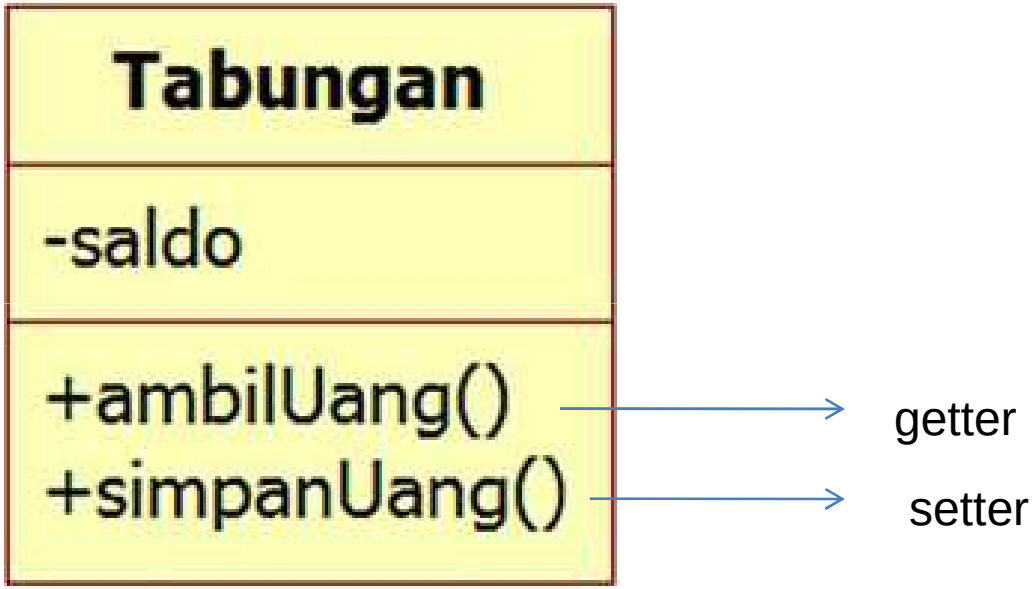
Method

- Disebut juga **behaviour**
- Memiliki hak akses **modifier** seperti pada pertemuan sebelumnya
- Terdapat 2 jenis method:
  - **Asesor**: method “pengambil nilai”, ada return value
    - getter, **get**NamaField
  - **Mutator**: method “pengubah nilai”
    - Setter, **set**NamaField

Tujuan Method

- Untuk implementasi dari enkapsulasi
- Untuk melindungi pengaksesan langsung variabel class dari luar class
- Untuk menciptakan modularitas
- Untuk menciptakan konsep API
- Getter dan Setter tidak harus ada pada semua variabel class
- *Client tidak perlu tahu detail sebuah isi variabel class dan juga isi method!*

Kasus



Contoh

```
public class Tabungan {
        private int saldo;
        ...
        public int getSaldo() {
                return saldo;
        }

        public void setSaldo(int s) {
                if(s >= 0)
                        saldo = s;
        }
}
```

Accessor / getter

Mutator / setter

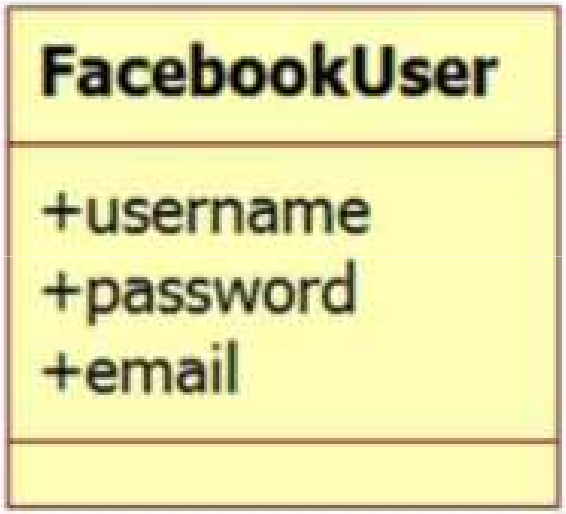
Remember ***this*** ?

```
public class Tabungan {
        private double saldo;

        public void ambilUang(double d) {
                double saldoBaru = saldo – d;
                if(saldoBaru >= 0.0)
                        this.saldo = saldoBaru;
        }
        public void simpanUang(double d) {
                if(d > 0.0) {
                        this.saldo = this.saldo + d;
                }
        }
}
```

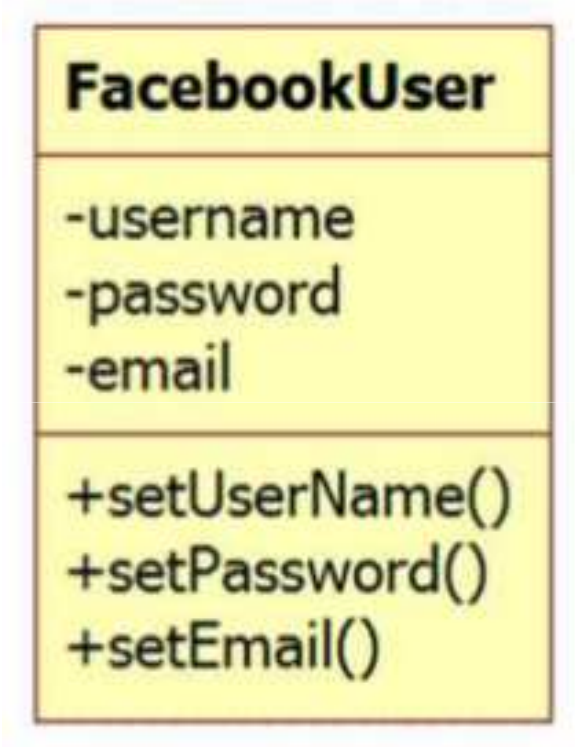

getter

setter

Apa bedanya atribut berikut?



Implementasi Enkapsulasi




```
FacebookUser fu = new FacebookUser()

fu.setUserName("anton");
fu.setPassword("rahasia");
fu.setEmail("anton@gmail.com");
```

Method invocation

- Pemanggilan method menggunakan tanda titik (.)
  - static method:
    - `namaClass.method(arguments)`
  - non-static method:
    - `namaObject.method(arguments)`

Method parameter

- Parameter yang ada pada suatu **method**
- Untuk memanggil method tersebut, kita mungkin perlu menyertakan **parameter**
- Contoh:

```
public void setNama(String nama){
    //isi kode
}
```

Method Paramter

- Ada method yang bisa dipanggil **tanpa** membutuhkan parameter

- Contoh:

```
public String getNama(){
    return this.nama;
}
```

Method Name

- Dalam satu class, bisa didefinisikan dua method atau lebih dengan **nama method yang sama**
- Yang membedakan method satu dengan lainnya adalah **jumlah** dan **jenis** parameternya (signature nya)

Pengiriman parameter

- Pengiriman parameter pada Java bersifat **by value**
- Ada dua value:
  - Tipe data **primitif**
    - Yang dikirimkan adalah nilainya (by value)
    - Perubahan nilai variabel di dalam method tidak berdampak pada nilai variabel di luar method
  - Tipe data **object**
    - Yang dikirimkan adalah ‘nilainya’ (alamat memory object) by value
    - Perubahan object di dalam method akan mempengaruhi object tersebut

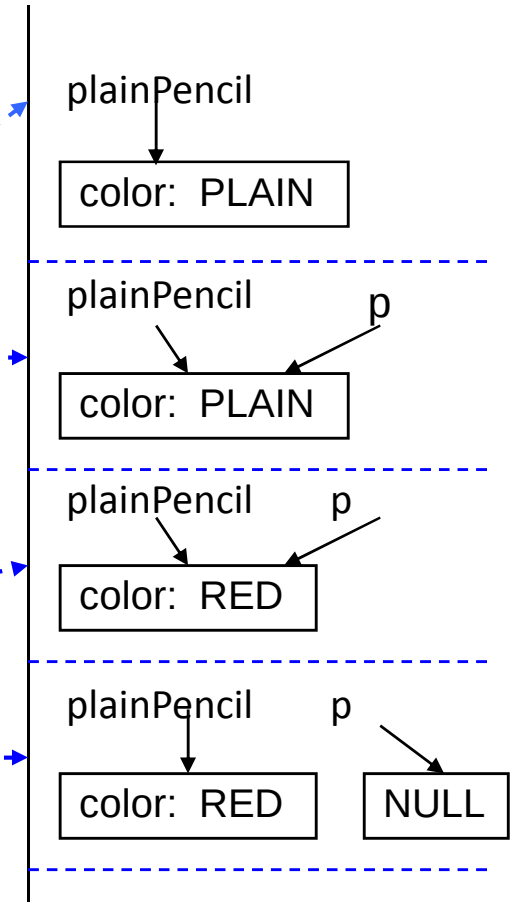
Method parameter

```
public class B {
        private int a;
        B() {
                a = 0;
        }
        public int getA() {
                return a;
        }
        public void setA(int a) {
                this.a = a;
        }
        public void ubahA(B b) {
                b.a = 2000;
        }
}
```

```
public class ContohB {
        public static void main(String args[]){
                //buat object b
                B b = new B();
                b.setA(1000);
                System.out.println("Nilai a = " +
b.getA());
                b.ubahA(b);
                System.out.println("Nilai a = " +
b.getA());
        }
```

Pass By Value

```
class PassRef{
   public static void main(String[] args) {
      Pencil plainPencil = new Pencil("PLAIN");
      System.out.println("original color: " +
   plainPencil.color);

      paintRed(plainPencil);

      System.out.println("new color: " +
   plainPencil.color);
   }

   public static void paintRed(Pencil p) {
      p.color = "RED";
      p = null;
   }
}
```



- You can change which object a parameter refers to inside a method without affecting the original reference which is passed

- What is passed is the object reference, and it's passed in the manner of "PASSING BY VALUE"!

Overloading method

- public void println(String s) { ... }
- public void println (int n) { ... }
- public void println(double d) { ... }

- System.out.println(“Halo semua...”);
- System.out.println(10);
- System.out.println(3.1415);

Method **yang sama**

- public int hitung(int alas, int tinggi) { ... }
- public int hitung(int a, int t) { ... }

- Kedua method tersebut dianggap **sama**, anda akan **gagal** untuk kompilasi

Gagal kompilasi

```
public class Contoh {
    public double hitung(int alas, int tinggi) {
        double luas = 0.5 * alas * tinggi;
        return luas;
    }

    public double hitung(int a, int b) {
        double luas = a * b;
        return luas;
    }
}
```

hitung(int,int) is already defined in Contoh
--
(Alt-Enter shows hints)

```
Contoh.java:17: hitung(int,int) is already defined in Contoh
    public double hitung(int a, int b) {
                  ^
1 error
```

Method **signature**

- public int hitung(int alas, int tinggi) { ... }
- public int hitung(int a, int t) { ... }
- public int hitung(**int, int**) -> method signature

Method signature

- Jika ada dua method bernama sama, method **signature** keduanya harus berbeda.
- Perbedaan dilihat dari jumlah dan urutan parameter

- public void daftar(String nama, int nim)
- public void daftar(int nim, String nama)
- public void daftar (String n, int id)
- public int daftar(String nama, int nim)

Overloading konstruktor

- Satu class bisa memiliki lebih dari 1 constructor
- Constructor harus berbeda (memiliki method signature yang berbeda) satu sama lain

- Tabungan(String nama, double saldo)
- Tabungan()
- Tabungan(String nama)

Contoh

```
public class Tabungan {
        private double saldo;
        Tabungan(double saldo) {
                this.saldo = saldo;
        }

        Tabungan() {
                this.saldo = 0;
        }
        public void simpanUang(double d) {
                if(d > 0.0)
                        this.saldo = this.saldo + d;
        }
        public double getSaldo() { return saldo; }
}
```

Tanya Jawab

Apa perbedaan antara

```
Tabungan t;
```

dan

```
Tabungan t = new Tabungan();
```

dan

```
new Tabungan();
```

Berapa nilai dari b.getSaldo() setelah operasi berikut:

```
...
Tabungan b = new Tabungan(10);
b.simpanUang(5000);
b.ambilUang(b.getSaldo() / 2);
...
```

Tanya jawab

```
public void rahasia(Tabungan that, double jumlah) {
        this.saldo = this.saldo – jumlah;
        that.saldo = that.saldo + jumlah;
}
```

Apa yang dihasilkan pada operasi berikut :

```
Tabungan t1 = new Tabungan(100);
Tabungan t2 = new Tabungan(1000);
t2.rahasia(t1, 200);
System.out.println(t2.getSaldo());
System.out.println(t1.getSaldo());
```

Class **reuse**

- Jika class-class berada dalam satu direktori yang sama, class bisa langsung digunakan
- Jika ingin menggunakan class yang berada dilokasi lain, gunakan keyword **import** dan **package**
- Class-class dikelompokkan dalam **package-package**

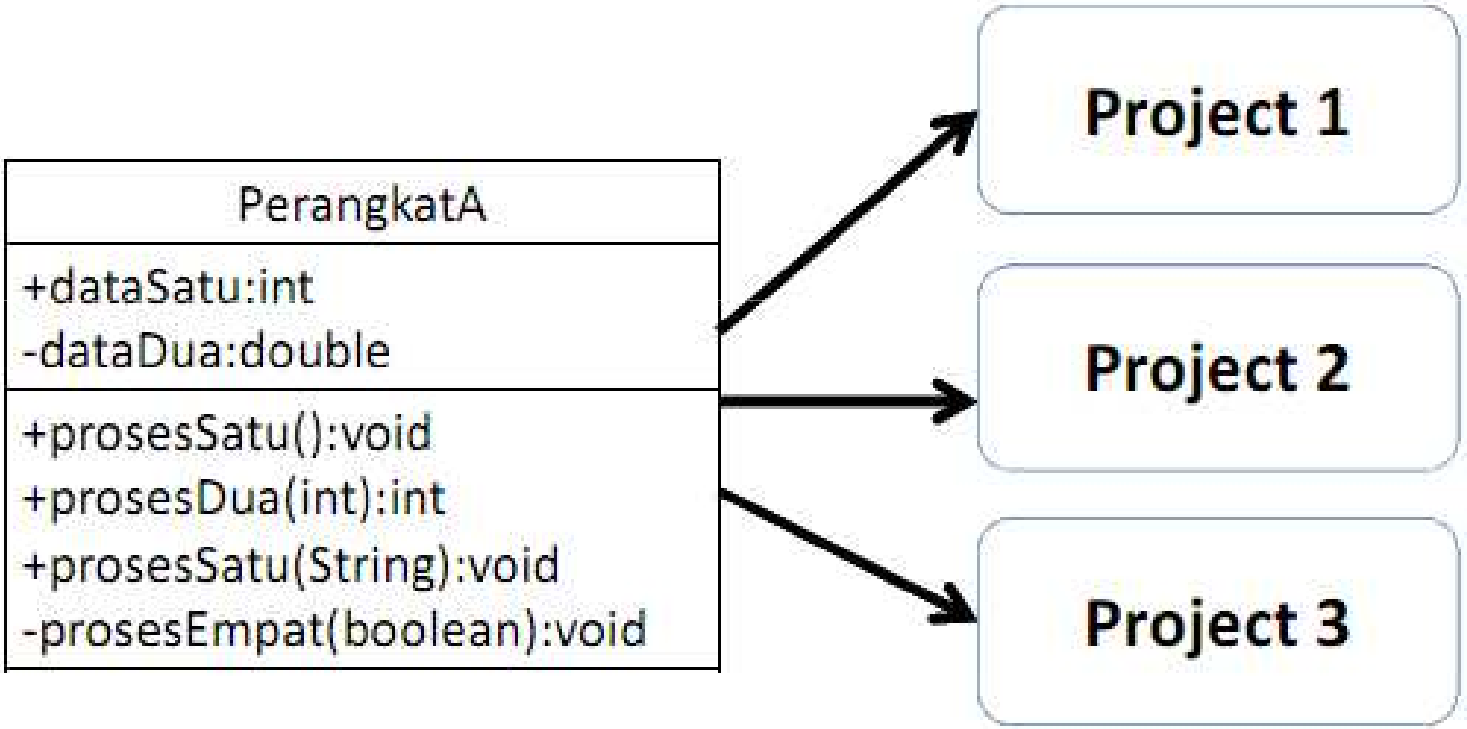
Contoh

```
package tres;

import dos.*;
import uno.Beta;

Class Tester {
    public static void main(String[] args) {
        Delta delta = new Delta();
        Beta beta = new Beta();
        Sigma sigma = new Sigma();
        ...
    }
}
```

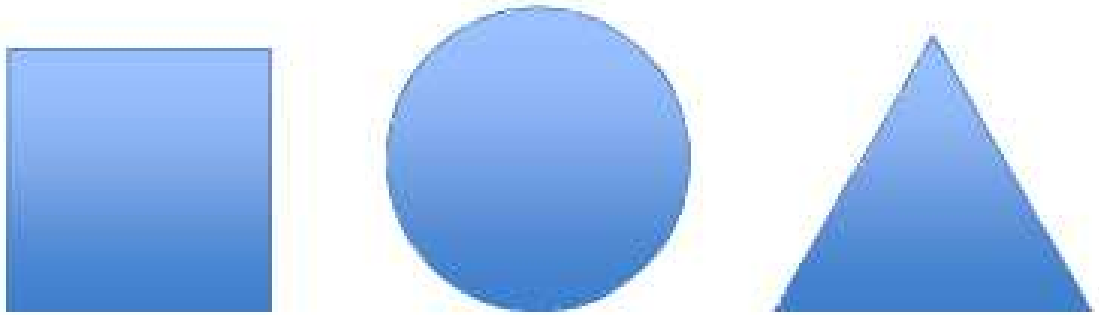
Class reuse

- Class-class yang sudah ada bisa langsung digunakan
- Suatu saat kita butuh mengubah atau menambahkan fungsi baru/atribut baru ke class yang sudah ada
- Ada beberapa pilihan:
  - Membuat class baru
  - Mengubah class lama

Contoh penggunaan



Contoh kasus



Buat aplikasi yang akan menampilkan tiga macam bentuk di layar. Masing-masing Jika diklik dengan mouse akan Berputar 180° dan memainkan file suara berformat aif spesifik untuk setiap bentuk

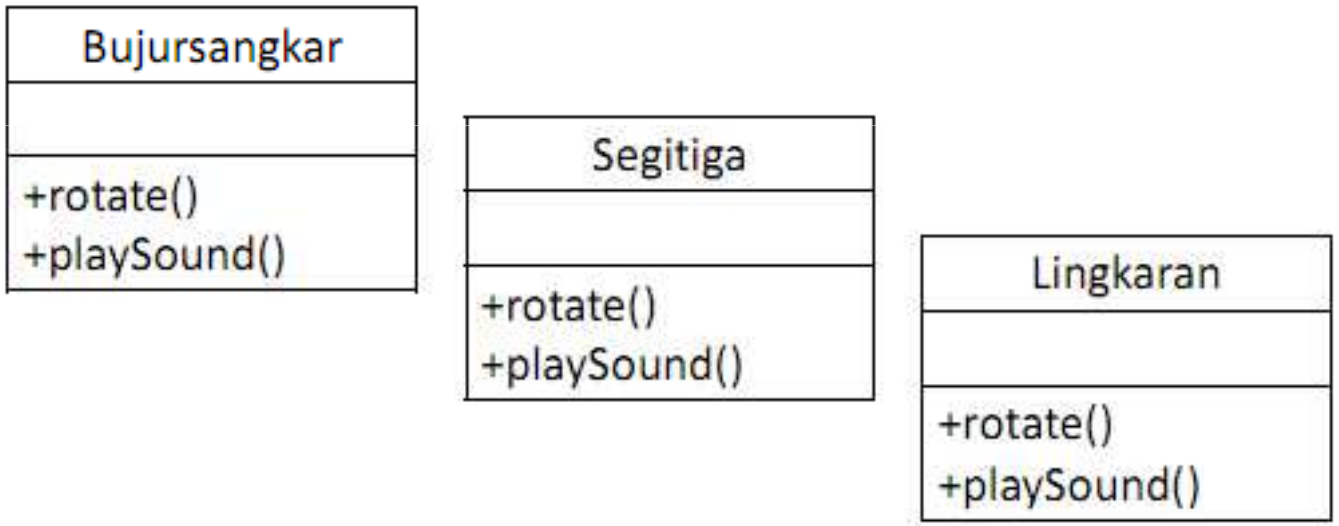
Contoh Kasus

```
rotate(bentuk) {
    //rotasi 180o
}

playsound(bentuk) {
    //cari suara yang akan dimainkan (.aif)
}
```

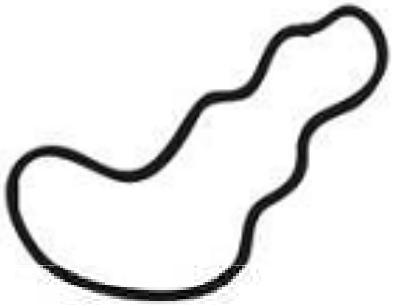
Cara Procedural !

Contoh Kasus

Cara OOP!



Perubahan requirements



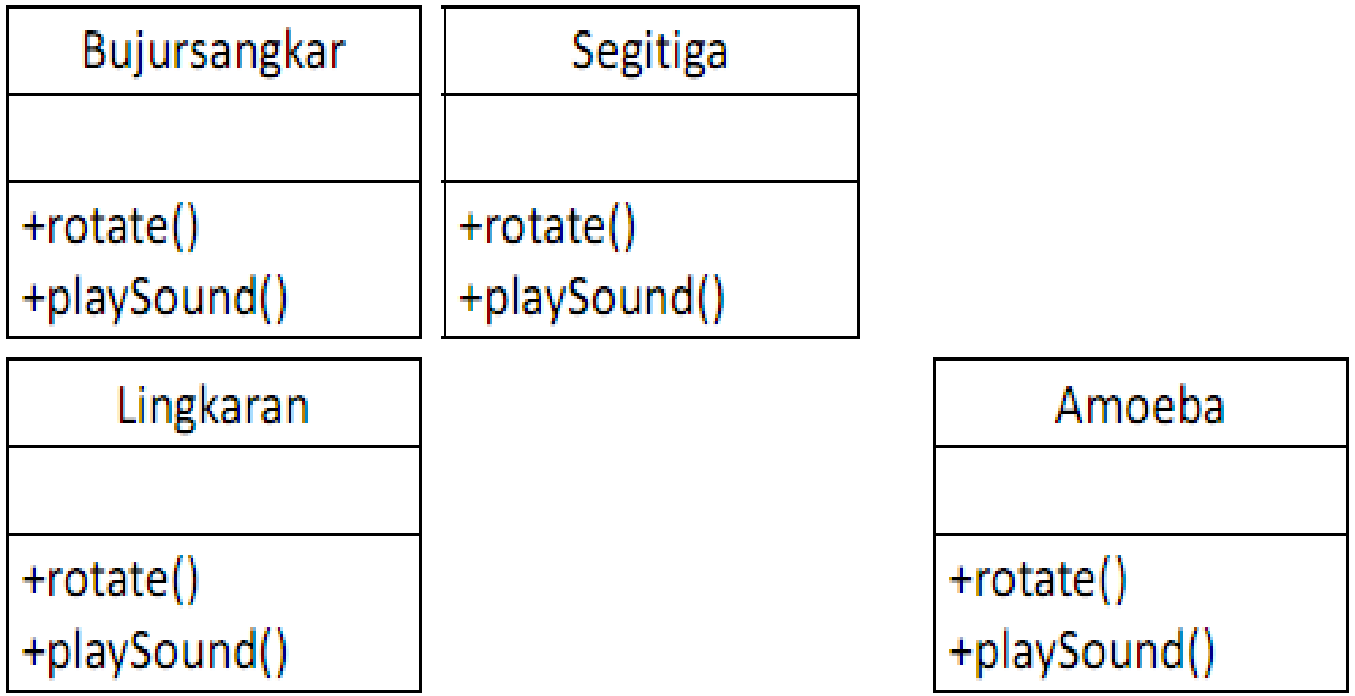
- Tambah satu bentuk lagi, yaitu Amoeba !
- Jika diklik akan berotasi 180° dan memainkan suara dari file .hif

Cara prosedural

```
rotate(bentuk) {
    //rotasi 180o
}

playsound(bentuk) {
    //cari suara yang akan dimainkan
    //jika amoeba, mainkan file .hif
    //jika bukan, mainkan file .aif
}
```

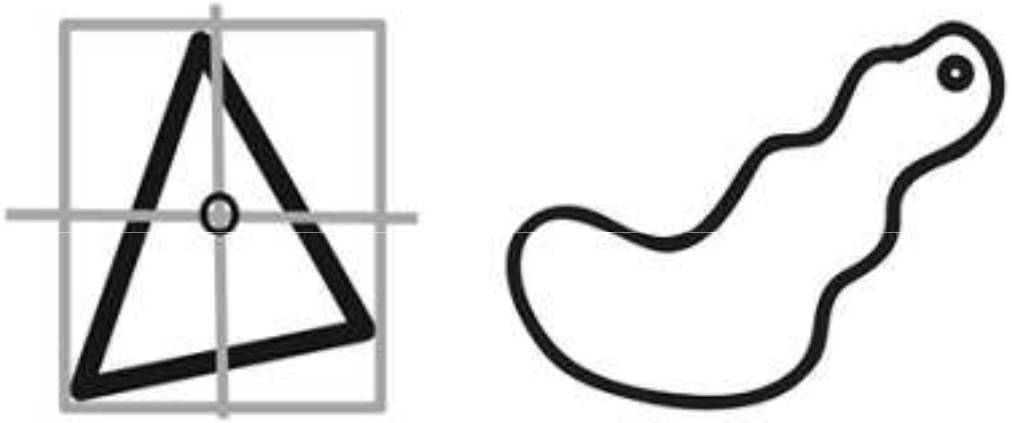
Perubahan requirement akan membuat Anda selalu mengubah code

Cara OOP



Penambahan Amoeba tidak membuat anda harus mengubah code sebelumnya yang sudah berjalan dengan baik !

Ada kesalahan!

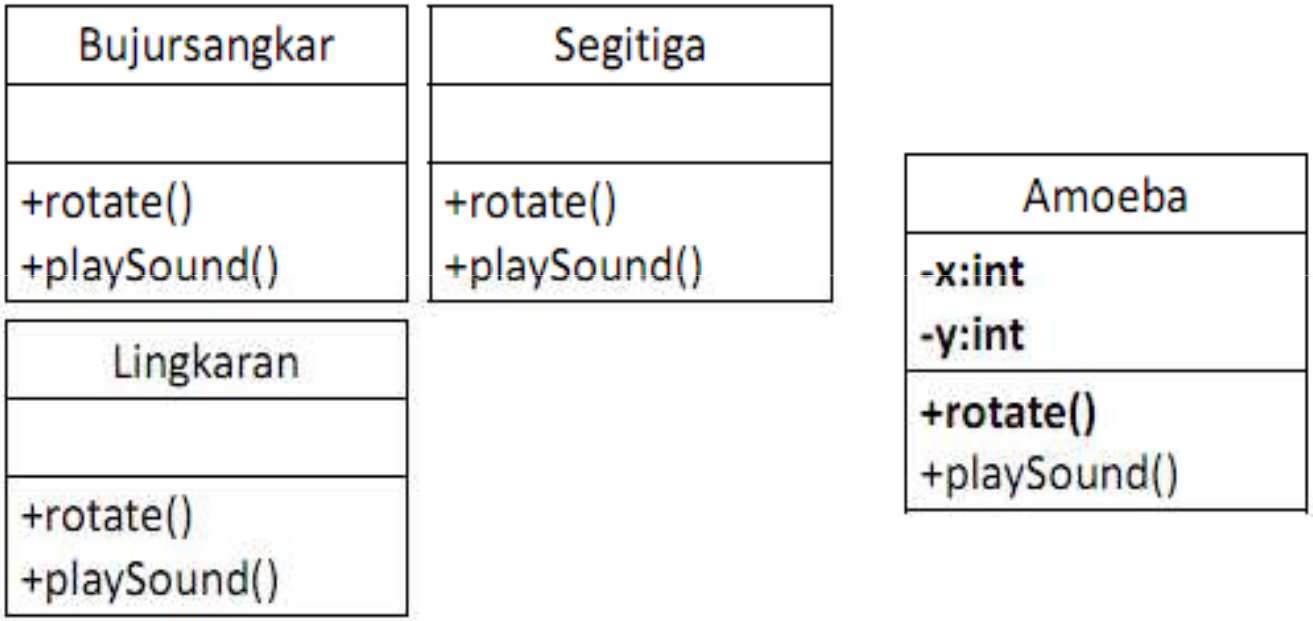


- Ternyata bentuk Amoeba bukan diputar dengan poros titik tengah seperti bentuk lainnya

Cara prosedural

```
rotate(bentuk, x, y) {
    //jika amoeba, rotasi x,y

    //jika bukan amoeba, rotate seperti biasa
}

playsound(bentuk) {
    //cari suara yang akan dimainkan
    //jika amoeba, mainkan file .hif
    //jika bukan, mainkan file .aif
}
```

Cara OOP



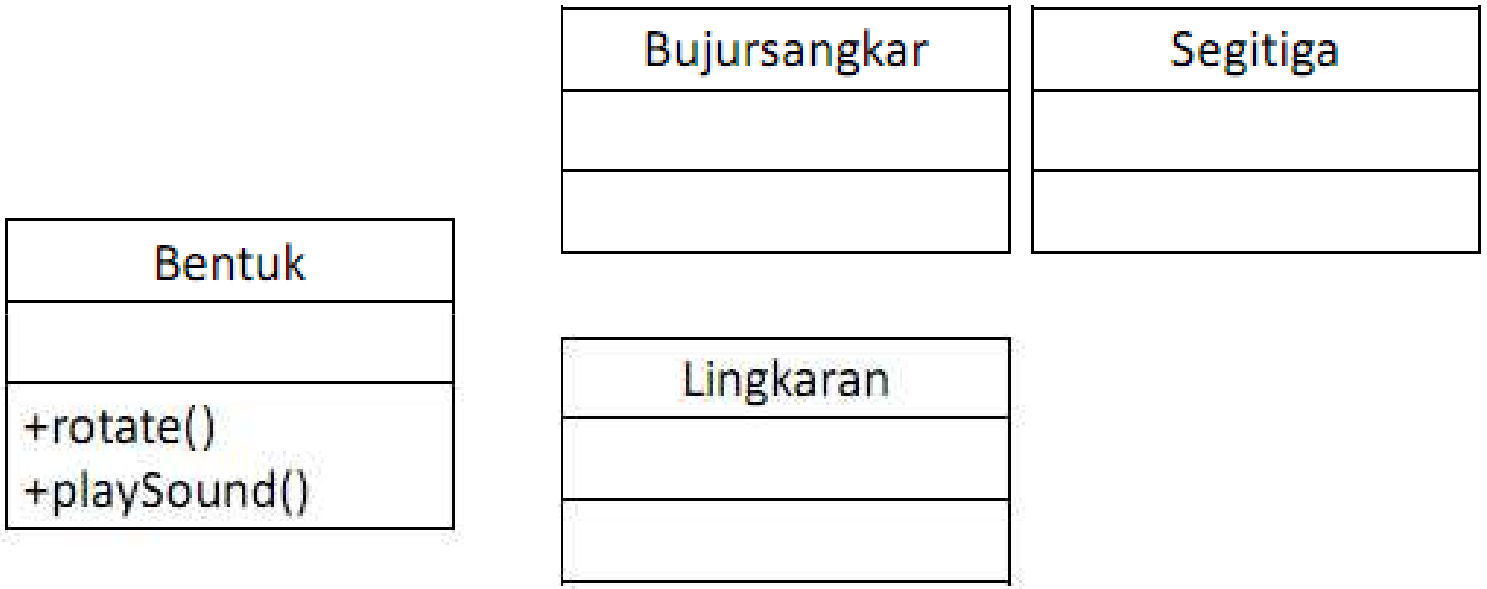
Untuk rotasi Amoeba, tinggal tambahkan field koordinat x dan y, lalu ubah method rotate untuk rotasi x,y

Tapi…

- Dengan cara OOP, kita punya 4 method rotate() dan 4 method playSound()
- Seluruh method tersebut harus di-kelola satu-persatu
- Pada tahap ini, design OO masih belum selesai

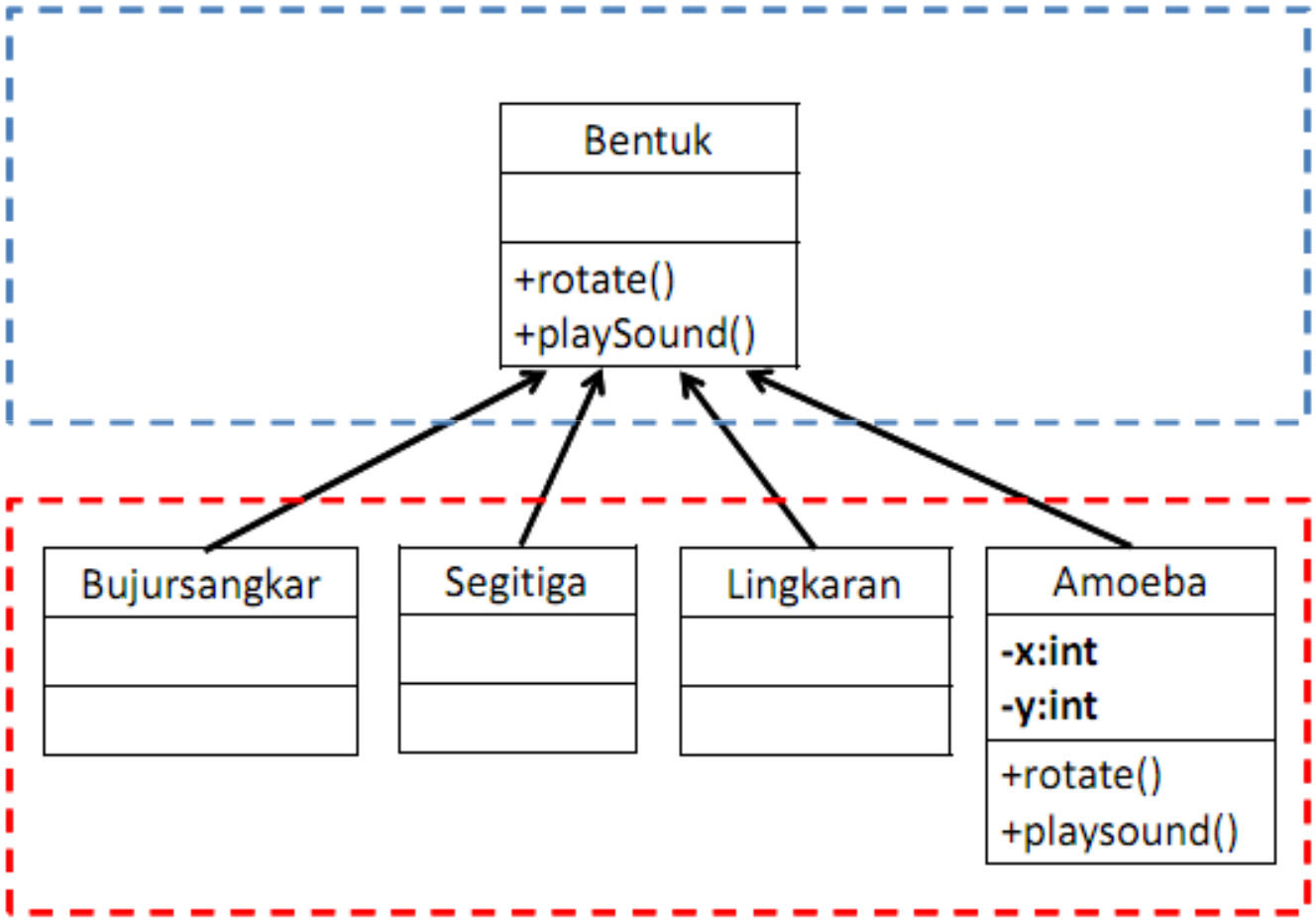
Perhatikan

Seluruh class memiliki method rotate() dan playSound()



Method yang sama dikelompokkan dalam class Bentuk

Akhirnya….



Akan dibahas detail di pertemuan berikutnya.....

Boxing dan Unboxing

- Tipe Data Primitive memiliki padanan tipe data Objectnya (Wrapper)

- Int -> Integer
- Boolean -> Boolean
- dan seterusnya …

Boxing dan Unboxing

- Terjadi secara otomatis
- **Boxing** : Konversi dari primitive type kereference type
- **Unboxing** : Konversi dari reference type ke primitive type

Contoh Boxing

```
public void hitungData(Integer m) {
    ...
}

...
int x = 10;
hitungData(new Integer(10));
hitungData(x);
hitungData(10);
```

Contoh unboxing

```
public void hitungData(int m) {
    ...
}

...
Integer data = new Integer(200);
int x = 100;
hitungData(data);
hitungData(x);
```

NEXT

- Inheritance