Getter & Setter

Encapsulation

- Encapsulation artinya memastikan data sensitif sebuah object tersembunyi dari akses luar
- Hal ini bertujuan agar kita bisa menjaga agar data sebuah object tetap baik dan valid
- Untuk mencapai ini, biasanya kita akan membuat semua field menggunakan access modifier private, sehingga tidak bisa diakses atau diubah dari luar
- Agar bisa diubah, kita akan menyediakan method untuk mengubah dan mendapatkan field tersebut

Getter dan Setter

- Di Java, proses encapsulation sudah dibuat standarisasinya, dimana kita bisa menggunakan Getter dan Setter method.
- Getter adalah function yang dibuat untuk mengambil data field
- Setter ada function untuk mengubah data field

Getter & Setter Method

Tipe Data	Getter Method	Setter Method
boolean	isXxx()	setXxx(boolean value)
primitif	getXxx()	setXxx(primitif value)
Object	getXxx()	setXxx(Object value)

Kode: Getter & Setter

```
class Data{
    4 usages
    public int intPublic;
    3 usages
    private int intPrivate;
    2 usages
    private double doublePrivate;

    1 usage
    public Data(){
        this.intPublic = 0;
        this.intPrivate = 10;
    }
    // getter
    1 usage
    public int getIntPrivate(){
        return this.intPrivate;
    }
    // setter
    1 usage
    public void setDoublePrivate(double value){
        this.doublePrivate = value;
    }
```

Kode: Getter & Setter

```
void display(){
    System.out.println(this.intPublic);
    System.out.println(this.intPrivate);
    System.out.println(this.doublePrivate);
}
```

Kode: Getter & Setter

```
class Lingkaran{
    5 usages
    private double diameter;

    1 usage
    Lingkaran(double diameter){
        this.diameter = diameter;
    }
    // setter
    1 usage
    public void setJari2(double jari2){
        this.diameter = jari2*2;
    }
    // getter
    2 usages
    public double getJari2(){
        return this.diameter/2;
    }
    // getter
    1 usage
    public double getLuas(){
        return 3.14*diameter*diameter/4;
    }
}
```

Kode: Getter & Setter

```java
public class Main{
    public static void main(String[] args) {

        Data object = new Data();

        // read and write dengan menggunakan public
        object.intPublic = 5; // Write
        System.out.println("public : " + object.intPublic); // read

        // read only (kita bisa menggunakan GETTER)
        int angka = object.getIntPrivate();
        System.out.println("getter : " + angka);

        // write only (kita hanya bisa menulis SETTER)
        object.setDoublePrivate(0.05);
        object.display();

        // gabungkan read dan write dengan setter dan getter
        Lingkaran object2 = new Lingkaran( diameter: 5);
        System.out.println("jari-jari : " + object2.getJari2());
        object2.setJari2(14);
        System.out.println("jari-jari : " + object2.getJari2());
        System.out.println("Luas : " + object2.getLuas());
    }
}
```

Static Keyword

Static Keyword

- Sebelumnya kita sudah sering melihat kata kunci static, namun belum pernah kita bahas
- Dengan menggunakan kata kunci static, kita bisa membuat field, method atau class bisa diakses langsung tanpa melalui object nya
- Perlu diingat, static hanya bisa mengakses static lainnya

Static Dapat Digunakan di

- Field, atau disebut class variable, artinya field tersebut bisa diakses langsung tanpa membuat object terlebih dahulu
- Method, atau disebut class method, artinya method tersebut bisa diakses langsung tanpa membuat object terlebih dahulu
- Block, static block akan otomatis dieksekusi ketika sebuah class di load
- Inner Class, artinya inner class tersebut bisa diakses secara langsung tanpa harus membuat object outer class terlebih dahulu. Static pada inner class menyebabkan kita tidak bisa mengakses lagi object outer class nya

Kode : Static Field Display

```
class Display{
    11 usages
    static String type = "Display";
    2 usages
    private String name;

    2 usages
    Display(String name){
        this.name = name;
    }
    1 usage
    void setType(String typeInput){
        // type = typeInput; // alternatif 1
        // this.type = typeInput; // alternatif 2
        Display.type = typeInput; // alternatif 3
    }
    2 usages
    void show(){
        System.out.println("Display name = " + this.name);
    }
}
```

Kode : Static Field Main(1)

```java
public static void main(String[] args) {
    Display display1 = new Display( name: "Monitor");
    display1.show();
    Display display2 = new Display( name: "Smartphone");
    display2.show();

    // tampilkan static variable atau class variable
    System.out.println("\nMenampilkan static variable");
    System.out.println(display1.type);
    System.out.println(display2.type);
    System.out.println(Display.type);

    // kita coba mengganti variable staticnya
    Display.type = "Tampilan";
    // display1.type = "Tampilan";
    // display2.type = "Tampilan";

    System.out.println("\nMenampilkan static variable");
    System.out.println(display1.type);
    System.out.println(display2.type);
    System.out.println(Display.type);
```

Kode : Static Field Main (2)

```
// kita coba mengganti variable staticnya
// display2.setType("Monitor");
display1.setType("Monitor");

System.out.println("\nMenampilkan static variable");
System.out.println(display1.type);
System.out.println(display2.type);
System.out.println(Display.type);
```

Kode : Static Method Player

```
import java.util.ArrayList;
14 usages
class Player{
    2 usages
    private static int numberOfPlayer;
    2 usages
    private static ArrayList<String> nameList = new ArrayList<String>();
    3 usages
    private String name;
    4 usages
    Player(String name){
        this.name = name;
        Player.numberOfPlayer++;
        Player.nameList.add(this.name);
    }
    no usages
    void show(){
        System.out.println("Player name = " + this.name);
    }
    // static method
    1 usage
    static void showNumberOfPlayer(){
        System.out.println("Number of Player = " + Player.numberOfPlayer);
    }
    1 usage
    static ArrayList<String> getNames(){
        return Player.nameList;
    }
```

Kode : Static Method Main

```
class StaticMethod{
    public static void main(String[] args) {
        Player player1 = new Player( name: "Saitama");
        Player player2 = new Player( name: "All Mighty");
        Player player3 = new Player( name: "Midnight");
        Player player4 = new Player( name: "Mt Lady");

        Player.showNumberOfPlayer();

        System.out.println("Names = " + Player.getNames());
        // System.out.println("Names = " + player1.getNames()); // ini bisa dilakukan

    }
}
```

Kode 2: Static Keyword Constant

```
public class Constant {

  public static final String APPLICATION = "Belajar Java OOP";
  public static final Integer VERSION = 1;

}
```

Kode 2: Static Keyword Application

```java
public class Application {

  public static final int PROCESSOR;

  static {
    PROCESSOR = Runtime.getRuntime().availableProcessors();
  }

}
```

Kode 2: Static Keyword MathUtil

```java
public class MathUtil {

  public static int sum(int... values) {
    int total = 0;
    for (var value : values) {
      total += value;
    }
    return total;
  }

}
```

Kode 2: Static Keyword InnerClass

```
class Country {
    2 usages
    private String name;
    no usages
    public String getName() {
        return name;
    }
    no usages
    public void setName(String name) {
        this.name = name;
    }
    2 usages
    public static class City {
        2 usages
        private String name;
        1 usage
        public String getName() {
            return name;
        }
        1 usage
        public void setName(String name) {
            this.name = name;
        }
    }
}
```

Kode 2: Static Main Class

```
public class StaticKeyword {
    public static void main(String[] args) {

        System.out.println(Constant.APPLICATION);
        System.out.println(Constant.VERSION);

        System.out.println(
                MathUtil.sum( ...values: 1,1,1,1,1)
        );

        Country.City city = new Country.City();
        city.setName("Subang");

        System.out.println(city.getName());

        System.out.println(Application.PROCESSORS);
    }
}
```

Latihan

Latihan 1 Sistem Pengelolaan Gaji Karyawan

Buatlah sebuah program untuk mengelola data gaji karyawan menggunakan konsep enkapsulasi dalam Java. Program tersebut harus memenuhi spesifikasi berikut:

Kelas Karyawan:

Atribut:

- nama (String): Nama karyawan.
- idKaryawan (String): ID unik untuk karyawan.
- gajiPokok (double): Gaji pokok karyawan.
- bonus (double): Bonus yang diterima karyawan.
- potongan (double): Potongan gaji karyawan (misal untuk pajak, asuransi, dll).

Latihan 1 Sistem Pengelolaan Gaji Karyawan (2)

Constructor:

Konstruktor untuk menginisialisasi nama, ID karyawan, dan gaji pokok. Bonus dan potongan diinisialisasi dengan nilai 0.

Method:

hitungGajiBersih(): Menghitung gaji bersih karyawan dengan rumus (gajiPokok + bonus - potongan).

Setter dan Getter untuk setiap atribut.

Latihan 1 Sistem Pengelolaan Gaji Karyawan (2)

Kelas ManajemenGaji

Metod Utama (main):

- Buatlah minimal 2 objek karyawan.
- Setiap karyawan mendapatkan bonus dan potongan yang berbeda-beda. Gunakan setter untuk mengatur nilai bonus dan potongan tersebut.
- Tampilkan ID karyawan, nama, dan gaji bersih setiap karyawan dengan memanfaatkan metode hitungGajiBersih() dari kelas Karyawan.

Kriteria Tambahan:

- Gunakan enkapsulasi sepenuhnya, pastikan semua atribut di kelas Karyawan bersifat private dan hanya dapat diakses melalui getter dan setter.
- Gunakan konvensi penamaan yang baik untuk variabel, metode, dan kelas.
- Pastikan program Anda memiliki penanganan kesalahan dasar untuk menghindari masukan yang tidak valid (misalnya, gaji pokok tidak boleh negatif).

Latihan 1 Sistem Pengelolaan Gaji Karyawan (2)

Output:

Program seharusnya dapat menampilkan ID karyawan, nama, dan gaji bersih setiap karyawan dalam format yang rapi.

```
ID Karyawan: K001, Nama: Alice, Gaji Bersih: $5300.0
ID Karyawan: K002, Nama: Bob, Gaji Bersih: $6300.0
```

Latihan 2 Aplikasi Penghitung Objek

Anda diminta untuk membuat sebuah aplikasi sederhana dalam Java untuk menghitung jumlah objek yang telah dibuat dari suatu kelas. Ini dapat digunakan untuk memahami bagaimana kata kunci static bekerja, terutama dalam konteks variabel dan metode statis.

Spesifikasi:

Kelas ObjekPenghitung

- Kelas ini harus memiliki satu variabel static yang akan digunakan untuk menyimpan jumlah objek yang telah dibuat dari kelas ini.
- Tambahkan sebuah konstruktor yang meningkatkan nilai variabel static setiap kali objek baru dibuat.
- Sediakan metode static untuk mendapatkan jumlah total objek yang telah dibuat.

Latihan 2 Aplikasi Penghitung Objek

Kelas PenghitungObjek

- Dalam kelas ini, buatlah beberapa objek dari kelas ObjekPenghitung.
- Tampilkan jumlah total objek yang telah dibuat menggunakan metode static yang telah disediakan di kelas ObjekPenghitung.

Instruksi:

- Pastikan Anda menggunakan konsep enkapsulasi dengan benar, terutama dalam pengaksesan variabel static.
- Gunakan komentar untuk menjelaskan bagian kode yang Anda anggap penting, terutama di bagian penggunaan variabel static dan metode static.
- Cobalah untuk membuat kode seefisien mungkin dan gunakan nama variabel yang menjelaskan dengan baik untuk meningkatkan keterbacaan kode.

Latihan 2 Aplikasi Penghitung Objek

Output:

Output harus menunjukkan jumlah total objek ObjekPenghitung yang telah dibuat setelah membuat beberapa objek dari kelas tersebut.

```
Jumlah total objek ObjekPenghitung yang telah dibuat: 3
```

Post Test

Post Test

Pada latihan terakhir pertemuan sebelumnya, kita sudah berhasil membuat sebuah game sederhana, coba kalian buat project baru untuk memperbaiki kode game tersebut:

Soal:

Buatlah game sederhana dimana memiliki 4 buah class dengan ketentuan sebagai berikut:

Ketentuan Class

class Player:

Player memiliki beberapa atribut yang semuanya memiliki access modifier private, yaitu name, baseHealth,baseAttack, incrementHealth,incrementAttack, level, totalDamage, dan isAlive. Player juga memiliki object member yaitu armor dari class Armor dan weapon dari class Weapon. Pada constructor Player memiliki 1 parameter saja, dimana isi constructornya adalah mengassign nilai name dengan parameter, baseHealth 100, baseAttack 100, level 1, incrementHealth 20, incrementAttack 20, dan isAlive true. Player juga menerapkan konsep enkapsulasi. Pada method Player, memiliki method display untuk menampilkan semua atribut dari Player, kemudian method attack, defence, dan maxHealth. Pada method attack terdapat aturan bahwasannya player yang diserang akan berkurang healthnya jika player yang diserang memiliki defence yang lebih rendah dari pada yang menyerang, jika tidak maka tidak ada demage yang didapatkan oleh yang diserang. Setiap penyerangan maka player yang menyerang akan level up. Kemudian pada method defence memiliki pengkondisian dimana jika demage penyerang lebih besar daripada defence yang diserang, maka terdapat delta demage sebesar damage - defence, jika tidak maka delta demagenya 0. Kemudian pada method defence juga terdapat pengecekkan apakah health si yang diserang kurang dari sama dengan 0 atau tidak, jika iya, maka atur variabel isAlivenya ke false dan atur totalDamage menjadi maxHealth. Method maxHealth berisikan baseHealth + level*incrementHealth + armor.hetAddHealth()

Ketentuan Class

class Weapon:

pada Weapon terdapat 2 variabel atau attribut yang memiliki access modifier private, yaitu String name dan integer attack. Weapon juga menearapkan prinsip enkapsulasi. Constructor Weapon memiliki isi mengisi variabelnya sendiri dengan parameter (terdapat 2 parameter pada constructor).

Ketentuan Class

class Armor:

pada Armor terdapat 3 variabel atau attribut yang memiliki access modifier private, yaitu name, strength, dan health. Armor juga menerarapkan prinsip enkapsulasi. Constructor Armor memiliki isi mengisi variabelnya sendiri dengan parameter (terdapat 3 parameter pada constructor).

Ketentuan Class

class Armor:

pada Armor terdapat 3 variabel atau attribut yang memiliki access modifier private, yaitu name, strength, dan health. Armor juga menerarapkan prinsip enkapsulasi. Constructor Armor memiliki isi mengisi variabelnya sendiri dengan parameter (terdapat 3 parameter pada constructor). Pada Armor terdapat method getAddHealth yang berisikan strength*10 + health. Kemudian terdapat method getDefencePower dimana isinya strength*2.

Class Utama

Pada kelas utama buatlah sebuah objek player1, player2, armor1, armor2, weapon1,weapon2. Lakukan penyerangan player1 terhadap player2, dan player2 terhadap player1 kemudian player 2 terhadap player1.

Contoh Tampilan

```
Player          : Marni
Level           : 1
Health          : 270/270
Attack          : 130
Alive           : true

Player          : Issabela
Level           : 1
Health          : 170/170
Attack          : 160
Alive           : true

Marni is attacking Issabela with 130
Issabela defence power = 2
damage earned = 128
```

```
Player          : Issabela
Level           : 1
Health          : 42/170
Attack          : 160
Alive           : true

Issabela is attacking Marni with 160
Marni defence power = 10
damage earned = 150

Player          : Marni
Level           : 2
Health          : 140/290
Attack          : 150
Alive           : true

Issabela is attacking Marni with 180
Marni defence power = 10
damage earned = 170

Player          : Marni
Level           : 2
Health          : 0/290
Attack          : 150
Alive           : false
```

Terima Kasih