



Praktikum Pemrograman Berorientasi Objek

Informatika

Universitas Negeri Padang

TIM DOSEN ©2022

JOBSHEET (JS-11)

Inheritance dan Encapsulation

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Informatika	INF1.62.3003	2 x 50 Menit

TUJUAN PRAKTIKUM

1. Mahasiswa mampu mendeskripsikan, memahami dan menggunakan konsep *Inheritance* dalam pemrograman Java.
2. Mahasiswa mampu mendeskripsikan, memahami dan menggunakan konsep *Encapsulation* dalam pemrograman Java.

HARDWARE & SOFTWARE

1. Personal Computer
2. IDE: NetBeans
3. JDK & JRE

TEORI SINGKAT

A. Inheritance

1. Inheritance (Pewarisan)

Inheritance merupakan proses pewarisan data dan method dari suatu class yang telah ada kepada suatu class baru. Class yang mewariskan disebut dengan superclass / parent class / base class, sedangkan class yang mewarisi (class yang baru) disebut dengan subclass / child class / derived class. Akan tetapi subclass tidak dapat mewarisi anggota private dari superclass-nya.

Dengan inheritance, class yang baru (subclass) akan mirip dengan class yang lama (superclass), namun memiliki karakteristik yang baru. Dalam Java, subclass hanya bisa memiliki satu superclass (single inheritance) sedangkan superclass bisa memiliki satu subclass atau lebih. Untuk menerapkan inheritance, digunakan statement “**extends**”.

```
namaSubclass extends namaSuperclass
{
    ..... //definisi class
}
```

2. Keyword super

Keyword “**super**” digunakan oleh subclass untuk memanggil konstruktor yang berada pada superclass-nya. Contoh untuk memanggil konstruktor milik superclass-nya:

```
super()
super(parameter)
```

Contoh untuk memanggil method milik superclass-nya:

```
super.namaMethod(parameter)
```

3. Keyword this

Kata kunci “**this**” sangat berguna untuk menunjukkan suatu member dalam class-nya sendiri. Kata kunci “**this**” dapat digunakan untuk data member, untuk method, dan untuk konstruktor. Adapun format penulisannya adalah:

```
this.data_member    -> merujuk pada data member
this.nama_method    -> merujuk pada method
this()              -> merujuk pada konstruktor
```

4. Overriding

Overriding adalah menulis kembali method sama persis, mulai dari nama method dan isinya dan mengimplementasi kembali di subclassnya. Overriding dipakai ketika menggunakan method yang sama tapi berbeda implementasinya atau dengan kata lain, overriding adalah suatu keadaan dimana method pada subclass menolak method pada parent class-nya. Overriding memiliki ciri-ciri sebagai berikut:

- a. Nama method harus sama
- b. Daftar parameter harus sama
- c. Return type harus sama

Jadi overriding method mempunyai nama method yang sama, jumlah parameter dan tipe parameter serta nilai kembalian (return) method yang di-override. Jika kita ingin memanggil metode superclass pada subclass dapat menggunakan statemen berikut:

```
super.namaMethod(parameter)
```

B. Encapsulation

1. Enkapsulasi/ Pembungkusan (Encapsulation)

Enkapsulasi merupakan suatu cara pembungkusan data dan method yang menyusun suatu kelas sehingga kelas dapat dipandang sebagai suatu modul dan cara bagaimana menyembunyikan informasi detail dari suatu class (information hiding). Dalam OOP, enkapsulasi sangat penting untuk keamanan serta menghindari kesalahan pemrograman, enkapsulasi dimaksudkan untuk menjaga suatu proses program agar tidak dapat diakses secara sembarangan atau diintervensi oleh program lain. Konsep enkapsulasi sangat penting dilakukan untuk menjaga kebutuhan program agar dapat diakses sewaktu-waktu, sekaligus menjaga program tersebut.

Dua hal yang mendasar dalam enkapsulasi yakni:

a. Information hiding

Sebelumnya untuk mengakses atribut atau method menggunakan objek secara langsung. Hal ini karena akses kontrol yang diberikan pada atribut dan method di dalam kelas tersebut adalah *public*. Untuk menyembunyikan informasi dari suatu kelas sehingga anggota kelas tersebut tidak dapat diakses kelas lain yaitu dengan memberi hak akses *private* pada atributnya. Proses ini disebut dengan *information hiding*.

b. Interface to access data

Interface to access data ini merupakan cara melakukan perubahan terhadap atribut yang disembunyikan, caranya adalah dengan membuat suatu *interface* berupa *method* untuk menginisialisasi atau merubah nilai dari suatu atribut tersebut. Manfaat utama teknik *encapsulation* adalah kita mampu memodifikasi kode tanpa merusak kode yang telah digunakan pada *class* lain.

Enkapsulasi memiliki manfaat sebagai berikut:

a. Modularitas

Source code dari sebuah *class* dapat dikelola secara independen dari *source code class* yang lain. Perubahan internal pada sebuah *class* tidak akan berpengaruh bagi *class* yang menggunakannya.

b. *Information Hiding*

Penyembunyian informasi yang tidak perlu diketahui objek lain.

2. Assesor dan Mutator Method

Assesor method adalah *method* yang digunakan untuk membaca nilai variabel pada *class*, baik berupa *instance* maupun *static*. Sebuah *accessor method* umumnya dimulai dengan penulisan **get<namaInstanceVariabel>**. *Method* ini juga mempunyai sebuah *return value*.

Mutator method adalah *method* yang digunakan untuk memberi atau mengubah nilai variabel dalam kelas, baik itu berupa *instance* maupun *static* variabel. Sebuah *mutator method* umumnya tertulis **set<namaInstanceVariabel>**. *Method* ini tidak menghasilkan balikan nilai atau *return value*.

3. Keuntungan menerapkan Encapsulasi

a. Bersifat independen

Suatu modul yang terencapsulasi dengan baik akan bersifat independen, sehingga tidak akan terikat pada bagian tertentu dari program.

b. Bersifat transparan

Bila melakukan modifikasi pada suatu modul, maka perubahan tersebut akan dirasakan juga oleh bagian program yang menggunakan modul tersebut.

c. Menghindari efek diluar perencanaan

Modul yang terencapsulasi dengan baik hanya akan berinteraksi dengan bagian program lainnya melalui variable-variabel input/output yang telah didefinisikan sebelumnya.

d. Melindungi listing program

Saat program didistribusikan pada khalayak, untuk melindungi listing program Anda dapat menerapkan prinsip enkapsulasi. Di sini pengguna hanya dapat menggunakan program melalui *variable input* atau *output* yang didefinisikan tanpa disertai bagaimana proses yang terjadi di dalam modul tersebut.

LATIHAN 1 : Inheritance

Buatlah ketiga *class* berikut dalam 3 file yang berbeda.

```
Pertama.java x
Source History
1 package Latihan1_Inheritance;
2
3 ...5 lines
8
9 /**...4 lines */
10 public class Pertama {
11     private int a = 10;
12
13     protected void terprotek() {
14         System.out.println("Method ini hanya untuk anaknya");
15     }
16
17     public void info() {
18         System.out.println("a = " + a);
19         System.out.println("Dipanggil pada = " + this.getClass().getName());
20     }
21 }
22
23
24
```

```
Kedua.java x
Source History
1 package Latihan1_Inheritance;
2
3 ...5 lines
8
9 /**...4 lines */
10 public class Kedua extends Pertama {
11     private int b = 8;
12
13     protected void BacaSuper() {
14         System.out.println("Nilai b = " + b);
15         terprotek();
16         info();
17     }
18 }
19
20
21
```

```
TestPertamaKedua.java x
Source History
1 package Latihan1_Inheritance;
2
3 ...5 lines
8
9 /**...4 lines */
13 public class TestPertamaKedua {
14     public static void main(String args[]) {
15         Kedua D2 = new Kedua();
16         D2.BacaSuper();
17         D2.info();
18
19         Pertama S1 = new Pertama();
20         S1.terprotek();
21         S1.info();
22     }
23 }
```

LATIHAN 2 : Inheritance

Penggunaan keyword *super* untuk memanggil konstruktor dari kelas induk.

Buatlah ketiga *class* berikut dalam 3 file yang berbeda.

```
Person.java x
Source History
1 ...5 lines
6 package Latihan2_Inheritance;
7
8 /**...4 lines */
9 public class Person {
13     protected String name;
14     protected int age;
15
16     public Person(String name, int age) {
17         this.name = name;
18         this.age = age;
19     }
20     //metode
21
22     public void info() {
23         System.out.println("Nama : " + this.name);
24         System.out.println("Usia : " + this.age);
25     }
26     //akhir kelas Program
27 }
```

```
Employ.java x
Source History
1 ...5 lines
6 package Latihan2_Inheritance;
7
8 /**...4 lines */
12 public class Employ extends Person{
13     private String noKaryawan;
14
15     //konstruktor
16     public Employ (String noKaryawan, String name, int age) {
17         super(name, age);
18         this.noKaryawan = noKaryawan;
19     }
20
21     //metode
22     public void info() {
23         System.out.println("No. karyawan : " + this.noKaryawan);
24         super.info();
25     }
26 }
```

```
KonstruktorSuperKelas.java x
Source History
1 ...5 lines
6 package Latihan2_Inheritance;
7
8 /**...4 lines */
12 public class KonstruktorSuperKelas {
13     public static void main(String[] args) {
14         Employ programmer1 = new Employ("12345678", "Budi", 33);
15         programmer1.info();
16     }
17 }
```

LATIHAN 3 : Inheritance

```
Hewan.java x
Source History
1 ...5 lines
6 package Latihan3_Inheritance;
7
8 /**...4 lines */
12 public class Hewan {
13     public static void testClassMethod() {
14         System.out.println("The Class Method in Hewan");
15     }
16
17     public void testInstanceMethod() {
18         System.out.println("The Instance Method in Hewan");
19     }
20 }
```

```
Gajah.java x
Source History
1  ...5 lines
6  package Latihan3_Inheritance;
7
8  /**...4 lines */
12 public class Gajah extends Hewan {
13     //meng-overwrite method pada class Hewan
14     public static void testClassMethod() {
15         System.out.println("The Class Method in Hewan");
16     }
17
18     //meng-overwrite method pada class Hewan
19     public void testInstanceMethod() {
20         System.out.println("The Instance Method in Gajah");
21     }
22
23     public static void main(String args[]) {
24         Gajah myGajah = new Gajah();
25         Hewan myHewan = myGajah;
26         Hewan.testClassMethod();
27         myHewan.testInstanceMethod();
28     }
29 }
```

LATIHAN 4 : Inheritance

Buatlah program berikut kemudian perhatikan keyword *super* pada program.

```
A.java x
Source History
1  ...5 lines
6  package Latihan4_Inheritance;
7
8  /**...4 lines */
9  public class A {
13     private int a;
14
15     public void setA(int nilai) {
16         a = nilai;
17     }
18
19     public int getA() {
20         return a;
21     }
22
23     public void tampilkanNilai() {
24         System.out.println("Nilai a = " + getA());
25     }
26 }
```



```
B.java x
Source History
1  ...5 lines
6  package Latihan4_Inheritance;
7
8  /**...4 lines */
12 public class B extends A {
13     private int b;
14
15     public void setB(int nilai){
16         b = nilai;
17     }
18
19     public int getB() {
20         return b;
21     }
22     //melakukan override terhadap method tampilkanNilai()
23     //yang terhadap pada kelas A
24     public void tampilkanNilai(){
25         super.tampilkanNilai(); //memanggil method dalam kelas A
26         System.out.println("Nilai b = " + getB());
27     }
28 }
```

```
DemoOverride2.java x
Source History
1  ...5 lines
6  package Latihan4_Inheritance;
7
8  /**...4 lines */
12 public class DemoOverride2 {
13     public static void main(String args[]){
14         B obj = new B();
15         obj.setA(50);
16         obj.setB(150);
17         //akan memanggil method yang terdapat pada kelas B
18         obj.tampilkanNilai();
19     }
20 }
```

LATIHAN 5 : Encapsulation

Contoh program tanpa encapsulation :

BUS
- Penumpang : int - maxPenumpang : int
+ cetak()

```
Bus.java
Source History
1  ...5 lines
6  package Latihan5_Encapsulation;
7
8  /**...4 lines */
12 public class Bus {
13     public int penumpang;
14     public int maxPenumpang;
15
16     public void cetak() {
17         System.out.println("Penumpang Bus sekarang adalah " + penumpang);
18         System.out.println("Penumpang maksimum seharusnya adalah " + maxPenumpang);
19     }
20 }
```

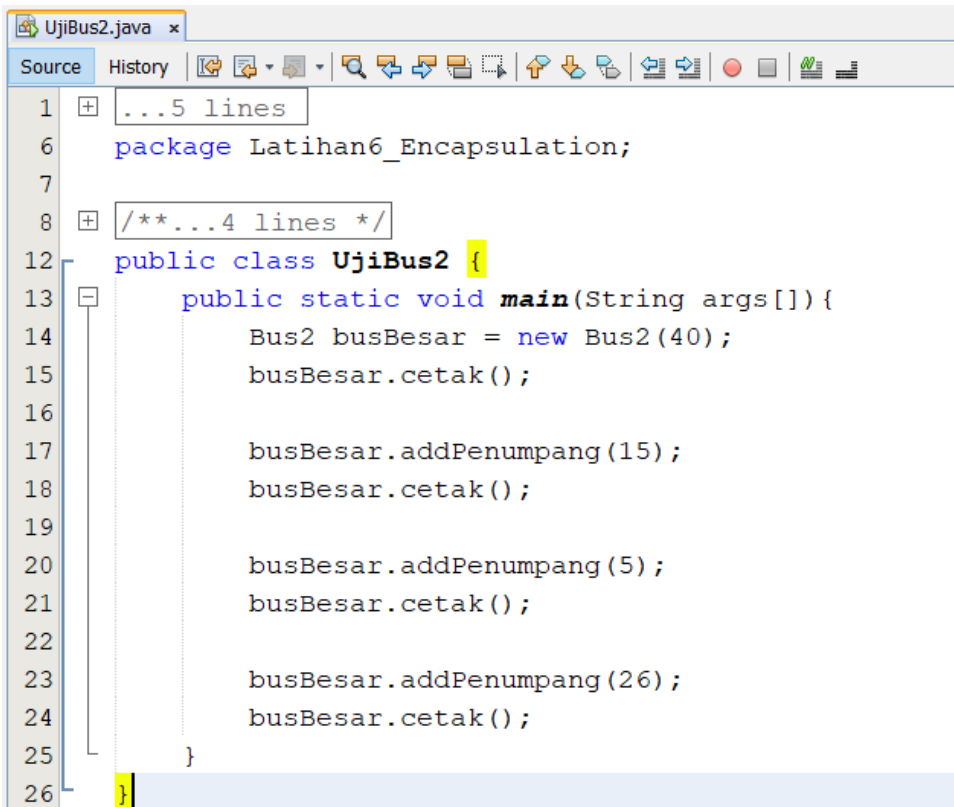
```
UjiBus.java
Source History
1  ...5 lines
6  package Latihan5_Encapsulation;
7
8  /**...4 lines */
12 public class UjiBus {
13     public static void main(String args[]) {
14         //membuat objek busMini dari kelas Bus
15         Bus busMini = new Bus();
16
17         //memasukkan nilai jumlah penumpang dan
18         //penumpang maksimal ke dalam obkkel busMini
19         busMini.penumpang = 5;
20         busMini.maxPenumpang = 15;
21
22         //memanggil method cetak pada kelas Bus
23         busMini.cetak();
24
25         //menambahkan jumlah penumpang pada busMini
26         busMini.penumpang = busMini.penumpang + 5;
27         //memanggil method cetak pada kelas Bus
28         busMini.cetak();
29
30         //mengurangi jumlah penumpang pada busMini
31         busMini.penumpang = busMini.penumpang - 2;
32         //memanggil method cetak pada kelas Bus
33         busMini.cetak();
34
35         //menambahkan jumlah penumpang pada busMini
36         busMini.penumpang = busMini.penumpang + 8;
37         busMini.cetak();
38     }
39 }
```

LATIHAN 6 : Encapsulation

Contoh program dengan encapsulation :

BUS2
- Penumpang : int - maxPenumpang : int
+ Bus(maxPenumpang : int) + addPenumpang(penumpang : int) + cetak()

```
Bus2.java x
Source History
1  ...5 lines
6  package Latihan6_Encapsulation;
7
8  /**...4 lines */
12 public class Bus2 {
13     private int penumpang;
14     private int maxPenumpang;
15
16     //konstruktor
17     public Bus2(int maxPenumpang) {
18         this.maxPenumpang = maxPenumpang;
19         penumpang = 0;
20     }
21
22     public void addPenumpang(int penumpang) {
23         int temp;
24         temp = this.penumpang + penumpang;
25         if (temp > maxPenumpang) {
26             System.out.println("Penumpang melebihi kuota");
27         }
28         else {
29             this.penumpang = temp;
30         }
31     }
32
33     public void cetak() {
34         System.out.println("Penumpang bus sekarang : " + penumpang);
35         System.out.println("Penumpang maks seharusnya : " + maxPenumpang);
36     }
37 }
```



```
1  ...5 lines
6  package Latihan6_Encapsulation;
7
8  /**...4 lines */
12 public class UjiBus2 {
13     public static void main(String args[]) {
14         Bus2 busBesar = new Bus2(40);
15         busBesar.cetak();
16
17         busBesar.addPenumpang(15);
18         busBesar.cetak();
19
20         busBesar.addPenumpang(5);
21         busBesar.cetak();
22
23         busBesar.addPenumpang(26);
24         busBesar.cetak();
25     }
26 }
```

DAFTAR PUSTAKA

1. Cipta Ramadhani. 2015. Dasar Algoritma & Struktur Data dengan Bahasa Java. Yogyakarta: ANDI.
2. Denny Kurniadi. 2017. Pemrograman Berorientasi Objek dengan Bahasa Pemrograman Java. Padang: UNP.
3. Wu, C. Thomas. 2010. *An Introduction to Object–Oriented Programming with Java 5th Edition*. C. USA: McGraw – Hill Education.
4. Nemeyer, Patrick and Luck, Daniel. 2013. *Learning Java 4th Edition*. O'Reilly
5. Sharan, Kishori. 2014. *Beginning Java 8 Fundamentals*. Apress.
6. Schildt, Herbert. 2014. *Java: The Complete Reference 9th Edition*. McGraw – Hill Education.