



PEMBUNGKUSAN (ENCAPSULATION)

Pertemuan ke-9

Sub-CPMK

- *Mahasiswa mampu menyelesaikan masalah dengan menggunakan salah satu pilar OOP yaitu Pembungkusan dalam konsep Pemrograman Berorientasi Objek (PBO). (C4, A4).*

Materi

1. Tentang Pembungkusan
2. Manfaat Pembungkusan
3. Konsep Pembungkusan
4. Mengapa Pembungkusan



1. Tentang Pembungkusan

- Mengenai Pembungkusan (*encapsulation*), pembungkusan merupakan salah satu dari empat pilar utama pemrograman berorientasi objek.
- Pembungkusan adalah proses pemaketan data bersama method-nya, pembungkusan bermanfaat untuk menyembunyikan rincian-rincian implementasi dari pemakai.



2. Manfaat Pembungkusan

- Pembungkusan dapat memberikan manfaat pada kosep pemrograman berorientasi objek, manfaat yang didapat dari pembukusan ada 2 (dua) yaitu:
 - Penyembunyian informasi (Information Hiding).
 - Modularitas.

2.1 Information Hiding

- Penyembunyian informasi (information hiding), hal ini mengacu kepada perlindungan terhadap implementasi objek internal. Objek tersebut dari interface public dan bagian private yang merupakan kombinasi data dan metode internal. Manfaat utamanya adalah bagian internal dapat berubah tanpa mempengaruhi bagian-bagian program yang lain.

2.2 Modularitas

- Modularitas berarti objek dapat dikelola secara independen. Karena kode sumber bagian internal objek dikelola secara terpisah dari antarmuka, maka Kita bebas melakukan modifikasi yang tidak menyebabkan masalah pada bagian-bagian lain dari sistem. Manfaat ini mempermudah mendistribusikan objek-objek dari sistem.



3. Konsep Pembungkusan

- Pembungkusan adalah cara "membungkus" data dan method yang menyusun kelas dan menyembunyikan dari dunia luar, dalam pemrograman menyembunyikan detail ini dikenal dengan "information hiding". Menyembunyikan semua data dan fungsi-fungsi yang berhubungan ke dalam sebuah kelas disebut dengan pembungkusan "encapsulation".

Lanj...

- Dalam mengimplementasikan konsep pembungkusan pada pemrograman berorientasi objek dapat diimplementasikan dengan cara:
 - property method.
 - interface.

3.1 Property Method

- Hal ini membuat segala perubahan dalam kelas bisa dikontrol oleh kelas yang bersangkutan dan hasilnya bersifat transparan bagi objek pengguna objek tersebut.
- Dalam menggunakan property method dapat menggunakan konsep **Get** dan **Set**.
- Penggunaan Get dan Set sesuai dengan kebutuhan, bisa menggunakan Get, bisa menggunakan Set atau Get dan Set.

3.1 Property Method (Lanj..)

Nama class: clsDokter

```
1.  package dokter;
2.  public class clsDokter
3.  {
4.      private String IdDokter;
5.      private String Nama;
6.      private int Gaji;
7.
8.      public void setIdDokter(String newValue)
9.      {
10.         IdDokter = newValue;
11.     }
12.     public String getIdDokter()
13.     {
14.         return IdDokter;
15.     }
```

3.1 Property Method (Lanj..)

```
16.     public void setName(String newValue)
17.     {
18.         Nama = newValue;
19.     }
20.     public String getName()
21.     {
22.         return Nama;
23.     }
24.     public void setGaji(int newValue)
25.     {
26.         Gaji = newValue;
27.     }
28.     public int getGaji()
29.     {
30.         return Gaji;
31.     }
```

3.1 Property Method (Lanj..)

```
32.     public float Tunjangan()  
33.     {  
34.         return Gaji/100*10;  
35.     }  
36.     public float TotalGaji()  
37.     {  
38.         return Gaji+Tunjangan();  
39.     }  
40. }
```

3.1 Property Method (Lanj..)

Main program: Dokter

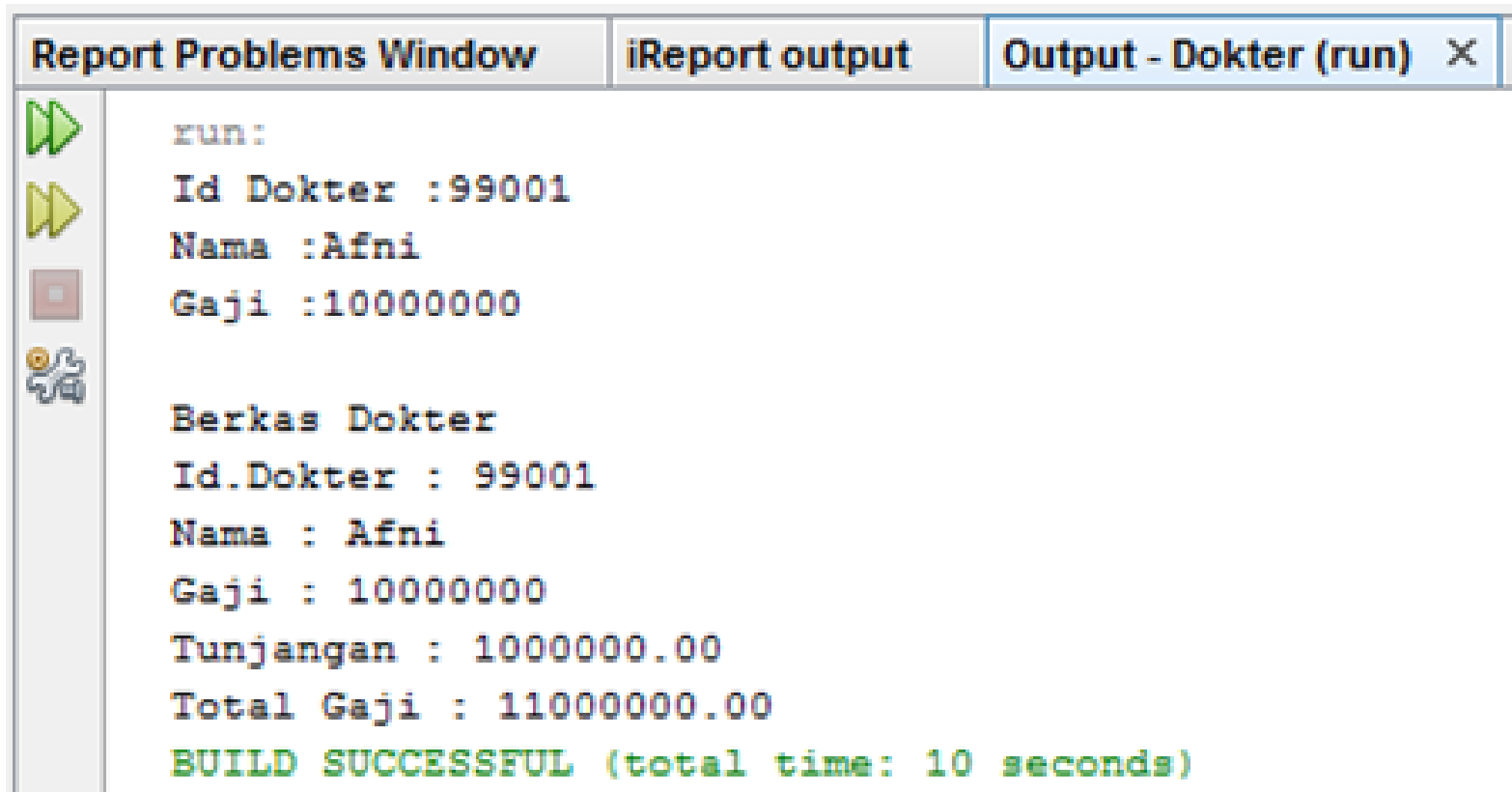
```
1.  package dokter;
2.  import java.util.Scanner;
3.
4.  public class Dokter
5.  {
6.
7.      public static void main(String[] args)
8.      {
9.          // TODO code application logic here
10.         Scanner input = new Scanner(System.in);
11.         clsDokter objDokter = new clsDokter();
12.
13.         System.out.print("Id Dokter :");
14.         objDokter.setIdDokter(input.nextLine());
```


3.1 Property Method (Lanj..)

```
15.      System.out.print("Nama :");
16.      objDokter.setNama(input.nextLine());
17.      System.out.print("Gaji :");
18.      objDokter.setGaji (input.nextInt());
19.      System.out.println();
20.      System.out.println("Berkas Dokter");
21.      System.out.println("Id.Dokter : "+
    objDokter.getIdDokter());
22.      System.out.println("Nama : " + objDokter.getNama());
23.      System.out.println("Gaji : " + objDokter.getGaji());
24.      System.out.printf("Tunjangan : %.2f\n",
    objDokter.Tunjangan());
25.      System.out.printf("Total Gaji : %.2f\n",
    objDokter.TotalGaji());
26.  }
27. }
```

3.1 Property Method (Lanj..)

Contoh keluaran program seperti berikut.



```
run:
Id Dokter :99001
Nama :Afni
Gaji :10000000

Berkas Dokter
Id.Dokter : 99001
Nama : Afni
Gaji : 10000000
Tunjangan : 1000000.00
Total Gaji : 11000000.00
BUILD SUCCESSFUL (total time: 10 seconds)
```

3.2 Interface

- Kelas implementasi menyembunyikan proses sebenarnya dari fungsi yang ada dalam interface.
- Objek pengguna hanya berhubungan dengan interface dan kelas mana yang akan digunakan untuk mengimplementasikannya bersifat transparan bagi objek pengguna.

3.2 Interface (Lanj..)

Nama interface: intDokter

```
1. package dokter;  
2.  
3. public interface intDokter  
4. {  
5.     double Tunjangan(int mGaji);  
6.     double TotalGaji(int mGaji, double mTunjangan);  
7. }
```

3.2 Interface (Lanj..)

Nama class: clsDokter

```
1.  package dokter;
2.  public class clsDokter implements intDokter
3.  {
4.      public String IdDokter;
5.      public String Nama;
6.      public int Gaji;
7.
8.      @Override
9.      public double Tunjangan(int mGaji)
10.     {
11.         return mGaji/100*10;
12.     }
```

3.2 Interface (Lanj..)

```
13.  
14.     @Override  
15.     public double TotalGaji(int mGaji, double mTunjangan)  
16.     {  
17.         mTunjangan = Tunjangan(mGaji);  
18.         return mGaji + (mTunjangan);  
19.     }  
20. }
```

3.2 Interface (Lanj..)

Main Program: Dokter

```
1.  package dokter;
2.  import java.io.*;
3.  import java.util.Scanner;
4.
5.  public class Dokter {
6.
7.      public static void main(String[] args) throws
        IOException
8.      {
9.          // TODO code application logic here
10.         InputStreamReader reader = new
        InputStreamReader(System.in);
11.         BufferedReader input = new
        BufferedReader(reader);
```

3.2 Interface (Lanj..)

```
12. Scanner masukan = new Scanner (System.in);
13.
14. clsDokter objDokter = new clsDokter();
15.
16. System.out.print("Masukkan Id Dokter:");
17. objDokter.IdDokter = input.readLine();
18. System.out.print("Masukkan Nama Dokter:");
19. objDokter>Nama = input.readLine();
20. System.out.print("Masukkan Gaji Dokter:");
21. objDokter.Gaji = masukan.nextInt();
22. System.out.println();
23. System.out.println("Id Dokter:
    "+objDokter.IdDokter);
```


3.2 Interface (Lanj..)

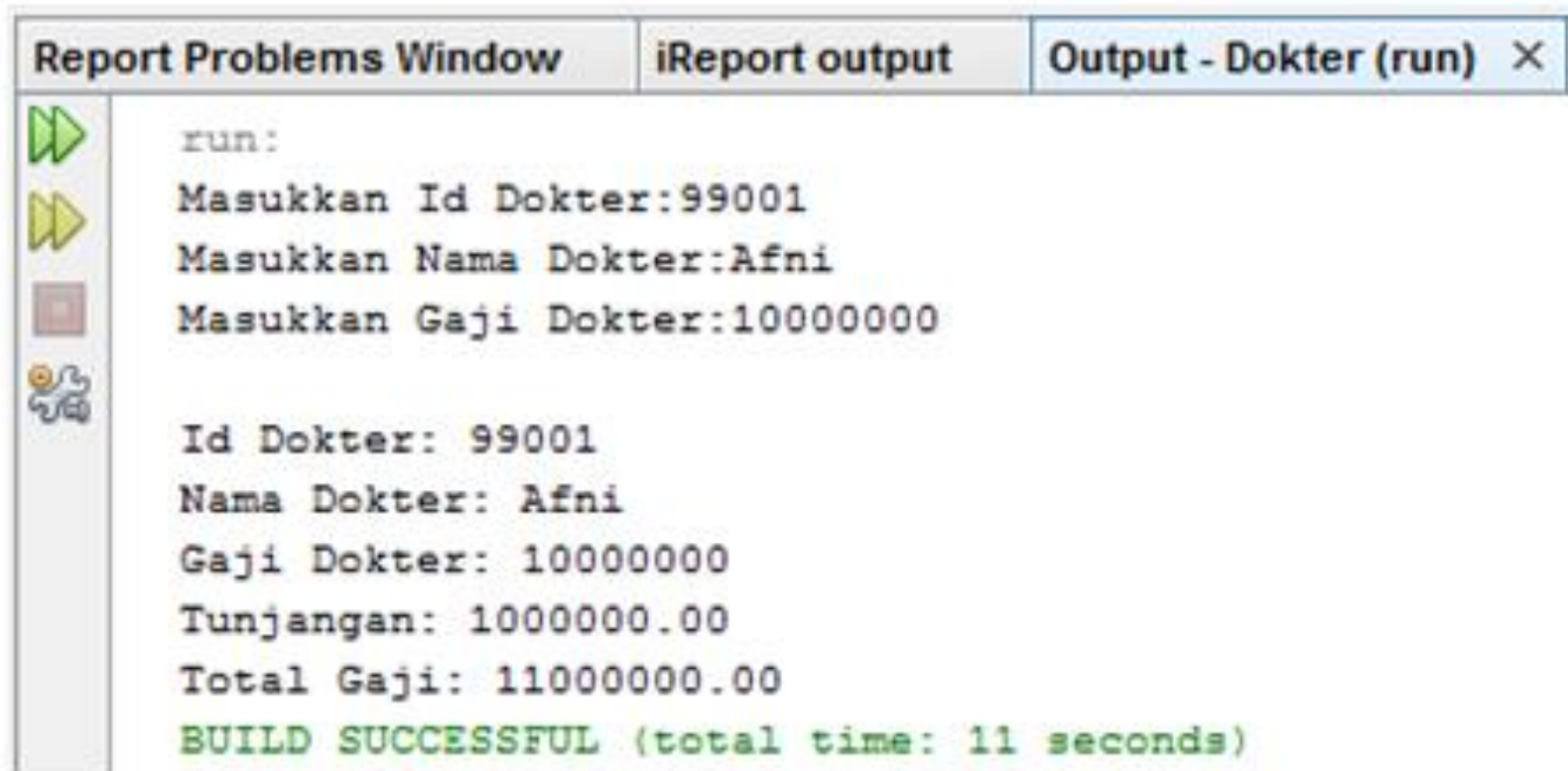
```
24.         System.out.println("Nama Dokter:
        "+objDokter>Nama);
25.         System.out.println("Gaji Dokter:
        "+objDokter.Gaji);
26.         System.out.printf("Tunjangan:
        %.2f\n",objDokter.Tunjangan(objDokter.Gaji));
27.         System.out.printf("Total Gaji: %.2f\n"
        ,objDokter.TotalGaji(objDokter.Gaji,objDokter.Tunjangan(o
        bjDokter.Gaji)));
28.     }
29. }
```

3.2 Interface (Lanj..)

```
24.         System.out.println("Nama Dokter:
        "+objDokter.Nama);
25.         System.out.println("Gaji Dokter:
        "+objDokter.Gaji);
26.         System.out.printf("Tunjangan:
        %.2f\n",objDokter.Tunjangan(objDokter.Gaji));
27.         System.out.printf("Total Gaji: %.2f\n"
        ,objDokter.TotalGaji(objDokter.Gaji,objDokter.Tunjangan
        (objDokter.Gaji)));
28.     }
29. }
```

3.2 Interface (Lanj..)

- Contoh keluarah program seperti berikut.



```
run:
Masukkan Id Dokter:99001
Masukkan Nama Dokter:Afni
Masukkan Gaji Dokter:10000000

Id Dokter: 99001
Nama Dokter: Afni
Gaji Dokter: 10000000
Tunjangan: 1000000.00
Total Gaji: 11000000.00
BUILD SUCCESSFUL (total time: 11 seconds)
```



4. Mengapa Pembungkusan

- Terkadang kita menerapkan konsep pembungkusan dalam program, hal ini dikarenakan:
 - **Bersifat independen**, suatu modul yang terkapsulasi dengan baik akan bersifat independen dari modul lainnya sehingga dapat digunakan pada bagian manapun dari program. Ia tidak akan terikat pada bagian tertentu dari program.

Lanj...

- **Bersifat transparan**, bila melakukan modifikasi pada suatu modul, maka perubahan tersebut akan dirasakan juga oleh bagian program yang menggunakan modul tersebut.
- **Menghindari efek yang diluar perancangan**, modul yang terkapsulasi dengan baik hanya akan berinteraksi dengan bagian program lainnya melalui variabel-variabel input/output yang telah didefinisikan sebelumnya. dengan demikian, akan mereduksi kemungkinan adanya hasil imbas pemrosesan yang diluar perencanaan semula.

Ringkasan:

- Salah satu dari empat pilar pemrograman berorientasi objek adalah pembungkusan (*encapsulation*).
- Pembungkusan digunakan untuk melindungi data, sehingga dengan konsep pembungkusan akan mereduksi kemungkinan adanya hasil imbas pemrosesan yang diluar perencanaan semula.

Latihan Mandiri

- Buatlah sebuah program berorientasi objek yang mengimplementasikan konsep pembungkusan dengan konsep property method.
- Buatlah sebuah program berorientasi objek yang mengimplementasikan konsep pembungkusan dengan konsep interface.



TERIMA KASIH

U N I V E R S I T A S B U N D A M U L I A