

FUNDAMEN

PENGEMBANGAN APLIKASI

Pertemuan 10



UNIVERSITAS
ISLAM
INDONESIA

ENCAPSULATION DAN MULTICLASS

Program Studi Informatika - Program Sarjana
Fakultas Teknologi Industri



TOPIK MATERI



1. Encapsulation
2. Setter
3. Getter
4. Multiclass
5. Hubungan antar *class*

ENCAPSULATION



OBJECT ORIENTED PROGRAMMING (OOP)



Empat konsep dasar Object Oriented Programming (OOP):

1. **Encapsulation**
2. Inheritance
3. Polymorphism
4. Abstraction

ENCAPSULATION

1. **Encapsulation** adalah mekanisme “**membungkus**” *attributes* dan *methods* **menjadi satu kesatuan**.
 - *Attribute* “**disembunyikan**” dari class lain atau **tidak mengizinkan** attribute suatu class diakses secara bebas/langsung.
 - Itulah alasan **encapsulation** dikenal juga sebagai **data hiding**.
 - Attribute **hanya** dapat diakses melalui *methods* yang ada **di class yang sama** dengan attribute.
2. *Methods* yang digunakan untuk mengakses dan memanipulasi attribute biasanya menggunakan format **getXxx()** & **setXxx()** atau sering disebut **getter** & **setter**.

ENCAPSULATION

Syarat encapsulation:

1. **Attribute** dideklarasikan dengan kata kunci **private**.
2. **Methods** **getXxx()** dan **setXxx()** dideklarasikan dengan kata kunci **public**.



TUJUAN ENCAPSULATION



1. Menentukan pemberian hak akses pada setiap attribut atau *method*.
2. Melindungi informasi attribut yang dirasa penting yang ada di dalam class.

Contoh: attribut seperti “**NIK**”, yang mana data ini tidak boleh terekspos ke objek lain dalam program maka perlu dienkapsulasi.

METHOD GETTER & SETTER

1. *Attribute* diakses dan dimanipulasi melalui methods **getXxx()** dan **setXxx()**
2. *Method Getter:*
 - Bertugas **mengakses** atau mengambil nilai dari attribute.
 - Biasanya berupa method **non-void**.
3. *Method Setter:*
 - Bertugas **memanipulasi** atau memberi nilai ke attribute.
 - Biasanya berupa method void.

ENCAPSULATION - EXAMPLE

Gunakan folder O10_encapsulation

Contoh:

1. Buat *class* dengan nama **Kaus** yang memiliki dua attribute : **warna** dan **strip**.
2. Ingat, untuk dapat membungkus data, attribute harus diberi kata kunci **private**.

```
3 public class Kaus {  
4     private String warna;  
5     private String strip;  
6  
7 }
```

3. Dengan deklarasi seperti itu, kita coba untuk mengakses *attribute* secara langsung melalui objek kelas tersebut, apa yang terjadi?

4. Misal di **main()** method terdapat objek **k** dan dimaksudkan untuk memberi nilai *attribute* **warna**, seperti ini:

```
3 public class O10_Encapsulation {  
4  
5     public static void main(String[] args){  
6         Kaus k = new Kaus();  
7         k.warna = "putih";  
8     }  
9 }
```

4. Ternyata tidak diperbolehkan lagi dalam konsep *encapsulation*.
5. Selain itu, ada pesan **error** karena *attribute* telah diberi kata kunci **private**.

ENCAPSULATION — EXAMPLE (CONT.)

Gunakan folder O10_encapsulation

7. Attribute di class **Kaus** (**warna** dan **strip**) **tidak boleh** diakses secara langsung.

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Kaus k = new Kaus();
7         k.warna = "putih";
8     }
9 }
```

8. Kita akan menambahkan methods **getWarna()** dan **setwarna()** di class **Kaus** untuk memanipulasi dan mengakses **attribute warna**.

9. Ingat, *methods* tersebut harus diberi kata kunci **public**.

```
3 public class Kaus {
4     private String warna;
5     private String strip;
6
7     public String getWarna() {
8         return warna;
9     }
10
11     public void setWarna(String warna) {
12         this.warna = warna;
13     }
14 }
```

10. Kode di **main()** *method* menjadi:

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Kaus k = new Kaus();
7         k.setWarna("putih");
8     }
9 }
```

ENCAPSULATION — EXAMPLE (CONT.)

Gunakan folder O10_encapsulation

11. Begitu pula jika ingin mengambil nilai attribute di class, attribute di class tidak boleh diakses secara langsung, seperti:

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Kaus k = new Kaus();
7         k.warna = "putih";
8         System.out.println(k.warna);
9     }
10 }
```

12. Ada pesan eror karena attribute telah diberi kata kunci private

13. Tapi harus menggunakan *method* **getWarna()** untuk mengakses *attribute* **warna**, seperti:

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Kaus k = new Kaus();
7         k.setWarna("putih");
8         System.out.println(k.getWarna());
9     }
10 }
```

14. Hasil eksekusinya:

```
Output - O10_Encapsulation (run) x
run:
putih
BUILD SUCCESSFUL (total time: 1 second)
```



ENCAPSULATION



Manfaat *encapsulation* adalah:

1. Menjaga suatu atribut class di dalam program agar **tidak dapat diakses secara sembarangan** atau diintervensi oleh program lain
2. *Information hiding*
3. Mengontrol data



MULTICLASS



MULTICLASS (MULTIPLE CLASSES)

1. *Multiclass* mengizinkan kita untuk **membuat lebih dari satu class di dalam satu program.**
2. Sebenarnya kita dapat **membuat lebih dari satu class** dalam satu berkas ***.java**, namun hal tersebut **tidak direkomendasikan** karena membuat kode sulit dibaca.
3. Oleh karena itu, pada contoh kali ini, setiap satu berkas ***.java** hanya akan memiliki satu *class*.

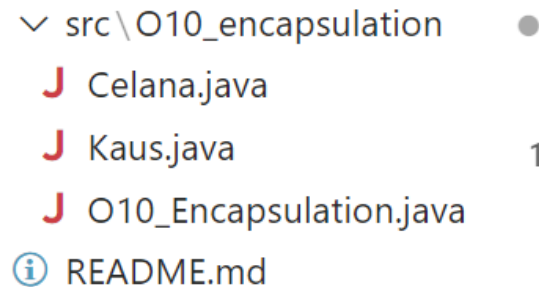
MULTICLASS (MULTIPLE CLASSES)

1. Setiap *class* akan memiliki **constructor** dan **method** masing-masing.
2. Dalam **main()** *method*, kita dapat **membuat objek** yang berasal dari banyak *class*, termasuk **mengakses methods** dari masing-masing *class* tersebut.
3. Suatu *class* juga dapat menjadi **instance variable** atau *attribute* untuk *class* lain.

MULTICLASS - EXAMPLE

Gunakan folder O10_encapsulation

1. Buat class dengan nama **Celana** di project sebelumnya.



```
src\O10_encapsulation
├── Celana.java
├── Kaus.java
├── O10_Encapsulation.java
└── README.md
```

2. Class **Celana** memiliki attribute **warna**, serta method Getter dan Setter.

```
3 public class Celana {
4     private String warna;
5
6     public String getWarna() {
7         return warna;
8     }
9
10    public void setWarna(String warna){
11        this.warna = warna;
12    }
13 }
```

3. Jadi kita memiliki **dua** class (selain class yang memiliki di **main()** method).
4. Masing-masing class tersebut dapat kita buat objeknya sendiri-sendiri di **main()** method.
 - Objek **c** untuk class **Celana**.
 - Objek **k** untuk class **Kaus**.

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Celana c = new Celana();
7         Kaus k = new Kaus();
8     }
9 }
```


MULTICLASS — EXAMPLE (CONT.)

Gunakan folder O10_encapsulation

5. Setiap objek tersebut juga dapat mengakses *method* dari kelas masing-masing di **main()** *method*.

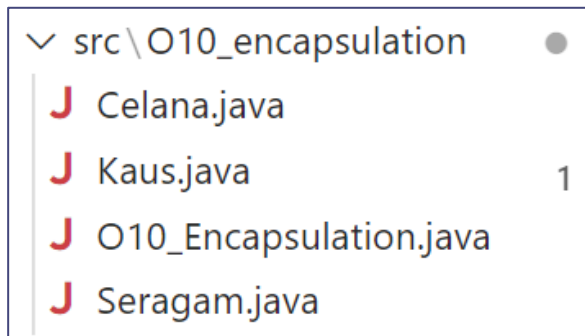
```
3 public class O10_Encapsulation {  
4  
5     public static void main(String[] args){  
6         Celana c = new Celana();  
7         Kaus k = new Kaus();  
8  
9         k.setWarna("putih");  
10        System.out.println("Warna kaus: "+k.getWarna());  
11  
12        c.setWarna("merah");  
13        System.out.println("Warna celana: "+c.getWarna());  
14    }  
15 }
```

```
Output - O10_Encapsulation (run) x  
run:  
Warna kaus: putih  
Warna celana: merah  
BUILD SUCCESSFUL (total time: 0 seconds)
```

MULTICLASS — EXAMPLE 2

Implementasi suatu class menjadi instance variable atau attribute untuk class lain

1. Buat class dengan nama **Seragam** di project sebelumnya.



2. Class **Seragam** memiliki dua attribute yaitu **kaus** dan **celana**.

```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6 }
```

3. Selain menggunakan *method Setter*, untuk inisialisasi nilai awal pada objek, kita dapat menggunakan **constructor**, seperti:

```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6
7     public Seragam(Kaus kaus, Celana celana){
8         this.kaus = kaus;
9         this.celana = celana;
10    }
11 }
```

4. Di **main()** method, objek **s** dari class **Seragam** diinstansiasi dengan *argument* berupa objek **k** dan **c**.

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Celana c = new Celana();
7         Kaus k = new Kaus();
8         Seragam s = new Seragam(k, c);
9     }
10 }
```

MULTICLASS — EXAMPLE 2 (CONT.)

Implementasi suatu class menjadi instance variable atau attribute untuk class lain

5. Untuk dapat melihat hasilnya maka berikutnya *method* untuk menampilkan atau **ToString** ditambahkan di masing-masing *class*.

```
3 public class Kaus {
4     private String warna;
5     private String strip;
6
7     public String getWarna() {
8         return warna;
9     }
10
11    public void setWarna(String warna) {
12        this.warna = warna;
13    }
14
15    String kausToString() {
16        return "kaus "+this.warna+" strip "+this.strip;
17    }
18 }
```

```
3 public class Celana {
4     private String warna;
5
6     public String getWarna() {
7         return warna;
8     }
9
10    public void setWarna(String warna){
11        this.warna = warna;
12    }
13
14    String celanaToString(){
15        return "celana "+this.warna;
16    }
17 }
```

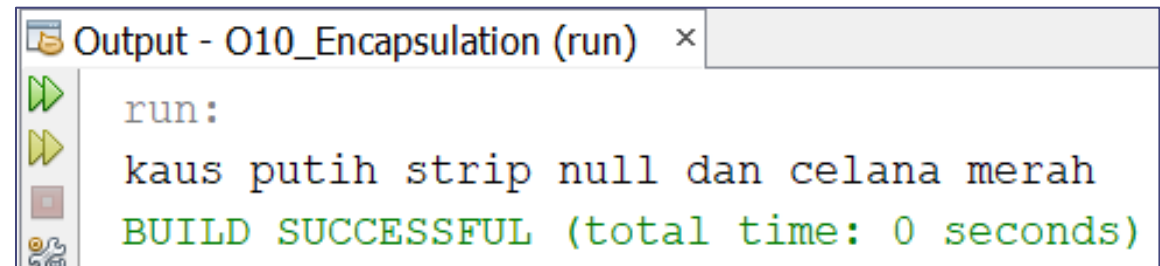
```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6
7     public Seragam(Kaus kaus, Celana celana){
8         this.kaus = kaus;
9         this.celana = celana;
10    }
11
12    public String seragamToString() {
13        return this.kaus.kausToString()+" dan "+this.celana.celanaToString();
14    }
15 }
```

MULTICLASS — EXAMPLE 2 (CONT.)

Implementasi suatu class menjadi instance variable atau attribute untuk class lain

6. Kode di **main()** method, untuk memberi warna kaus dan celana di objek **s** dengan cara memberi warna pada masing-masing objek **k** dan **c** terlebih dahulu.
7. Kemudian untuk menampilkan warna kaus dan celana di objek **s**, dapat dilakukan dengan memanggil method **seragamToString()** dari class **Seragam**.

```
3 public class O10_Encapsulation {
4
5     public static void main(String[] args){
6         Celana c = new Celana();
7         Kaus k = new Kaus();
8         Seragam s = new Seragam(k, c);
9
10        k.setWarna("putih");
11        c.setWarna("merah");
12        System.out.println(s.seragamToString());
13    }
14 }
```



Output - O10_Encapsulation (run) ×

run:
kaus putih strip null dan celana merah
BUILD SUCCESSFUL (total time: 0 seconds)

CLASS DIAGRAM

CLASS DIAGRAM - CLASS KAUS

```
3  public class Kaus {  
4      private String warna;  
5      private String strip;  
6  
7      public String getWarna() {  
8          return warna;  
9      }  
10  
11     public void setWarna(String warna) {  
12         this.warna = warna;  
13     }  
14  
15     String kausToString() {  
16         return "kaus "+this.warna+" strip "+this.strip;  
17     }  
18 }
```

Kaus
-warna: String -strip: String
+getWarna(): String +setWarna(warna: String): void ~kausToString(): String

CLASS DIAGRAM - CLASS CELANA

```
3 public class Celana {  
4     private String warna;  
5  
6     public String getWarna() {  
7         return warna;  
8     }  
9  
10    public void setWarna(String warna){  
11        this.warna = warna;  
12    }  
13  
14    String celanaToString(){  
15        return "celana "+this.warna;  
16    }  
17 }
```

Celana
-warna: String
+getWarna(): String +setWarna(warna: String): void ~celanaToString(): String



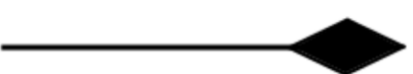
CLASS DIAGRAM - CLASS SERAGAM

```
3  public class Seragam {  
4      private Kaus kaus;  
5      private Celana celana;  
6  
7      public Seragam(Kaus kaus, Celana celana){  
8          this.kaus = kaus;  
9          this.celana = celana;  
10     }  
11  
12     public String seragamToString() {  
13         return this.kaus.kausToString()+" dan "+this.celana.celanaToString();  
14     }  
15 }
```

Seragam
-kaus: Kaus -celana: Celana
+Seragam(kaus: Kaus, celana: Celana) +seragamToString(): String

HUBUNGAN ANTARKELAS

HUBUNGAN ANTARKELAS

1. **Class diagram** adalah bagian dari UML (*Unified Modeling Language*) yang menggambarkan struktur dan deskripsi serta **hubungan antar class** diagram tersebut.
2. Ada berbagai macam hubungan/relasi antarkelas.
3. Namun, yang kita bahas kali ini hanya tiga yaitu:
 - a. Association : 
 - b. Aggregation : 
 - c. Composition : 



ASSOCIATION

1. Suatu class *tidak memiliki attribute* yang bertipe class lain.
2. Menggambarkan hubungan antarkelas yang *tidak saling memiliki*, hanya mengambil nilai dari *attribute class* lain.
3. Contoh **Dosen** dan **Mahasiswa**.
 - Banyak **Mahasiswa** dapat berasosiasi dengan satu **Dosen** dan satu **Mahasiswa** dapat berasosiasi dengan banyak **Dosen**.
 - *Object* keduanya dapat dibuat dan dihancurkan *secara mandiri* tanpa berpengaruh pada eksistensi *object* lain.

ASSOCIATION - EXAMPLE

```
3 public class Dosen {
4     private String namaDosen;
5     private int nMhsBimbingan;
6     private int nimMhsBimbingan[];
7
8     public Dosen(String nama){
9         this.namaDosen = nama;
10        this.nMhsBimbingan = 0;
11    }
12
13    public void setNimMhsBimbingan(int nim) {
14        this.nimMhsBimbingan[nMhsBimbingan] = nim;
15        nMhsBimbingan++;
16    }
17 }
```

```
3 public class Mahasiswa {
4     private int nimMahasiswa;
5     private String namaMahasiswa;
6
7     public Mahasiswa(int nim, String nama){
8         this.nimMahasiswa = nim;
9         this.namaMahasiswa = nama;
10    }
11
12    public int getNimMahasiswa() {
13        return nimMahasiswa;
14    }
15 }
```

Kode di **main()** method

```
3 public class Perkuliahan {
4     public static void main(String[] args) {
5         Mahasiswa mhs1 = new Mahasiswa(11, "Nussa");
6         Mahasiswa mhs2 = new Mahasiswa(12, "Rara");
7         Dosen dsn = new Dosen("ABC");
8         dsn.setNimMhsBimbingan(mhs1.getNimMahasiswa());
9         dsn.setNimMhsBimbingan(mhs2.getNimMahasiswa());
10    }
11 }
```

- Kode di atas menunjukkan, Class **Dosen** dan Class **Mahasiswa** tidak saling memiliki,
- Class **Dosen** maupun Class **Mahasiswa** tidak memiliki *attribute* yang bertipe *class* lain, hanya mengambil nilai *attribute* dari *class* lain, yaitu di baris 8 dan 9 di **main()** method.

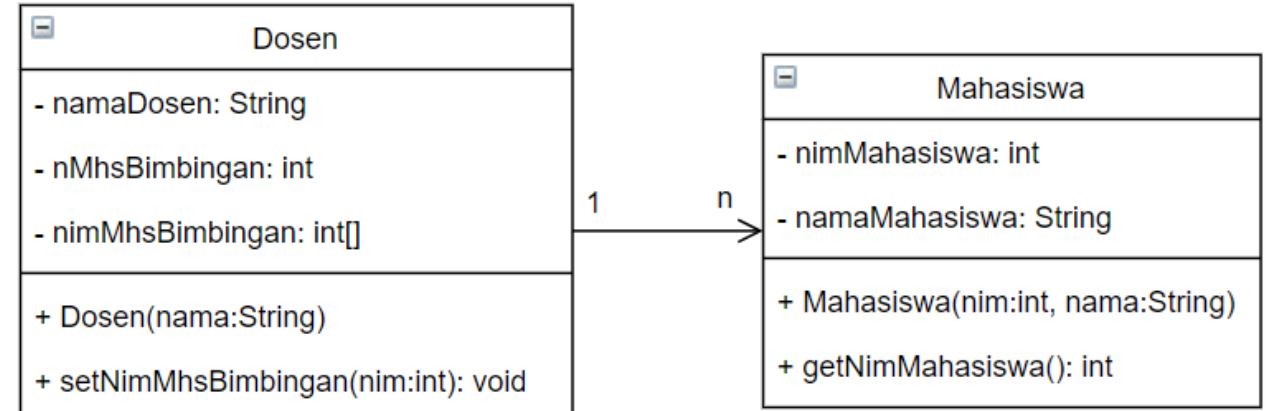
ASSOCIATION - EXAMPLE

```
3 public class Dosen {
4     private String namaDosen;
5     private int nMhsBimbingan;
6     private int nimMhsBimbingan[];
7
8     public Dosen(String nama){
9         this.namaDosen = nama;
10        this.nMhsBimbingan = 0;
11    }
12
13    public void setNimMhsBimbingan(int nim) {
14        this.nimMhsBimbingan[nMhsBimbingan] = nim;
15        nMhsBimbingan++;
16    }
17 }
```

```
3 public class Mahasiswa {
4     private int nimMahasiswa;
5     private String namaMahasiswa;
6
7     public Mahasiswa(int nim, String nama){
8         this.nimMahasiswa = nim;
9         this.namaMahasiswa = nama;
10    }
11
12    public int getNimMahasiswa() {
13        return nimMahasiswa;
14    }
15 }
```

```
3 public class Perkuliahan {
4     public static void main(String[] args) {
5         Mahasiswa mhs1 = new Mahasiswa(11, "Nussa");
6         Mahasiswa mhs2 = new Mahasiswa(12, "Rara");
7         Dosen dsn = new Dosen("ABC");
8         dsn.setNimMhsBimbingan(mhs1.getNimMahasiswa());
9         dsn.setNimMhsBimbingan(mhs2.getNimMahasiswa());
10    }
11 }
```

Class diagram:



AGGREGATION

1. Menggambarkan hubungan dua *class* yang salah satunya merupakan bagian dari *class* lainnya.
2. **Aggregation** ditunjukkan dengan adanya *class* lain yang menjadi *attribute* dari suatu *class* (*attribute* suatu *class* mempunyai 'tipe' *class* lain).
3. Saat objek “wadah” (contoh: objek **s** dari *class* **Seragam**) dimusnahkan, objek “muatan” (contoh: objek **k** dari *class* **Kaus**) tetap ada (dapat berdiri sendiri)
 - Pada kasus ini, *class* **Seragam** memiliki attribut yang berasal dari *class* **Kaus**

AGGREGATION — EXAMPLE

```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6
7     public Seragam(Kaus kaus, Celana celana){
8         this.kaus = kaus;
9         this.celana = celana;
10    }
11
12    public String seragamToString() {
13        return this.kaus.kausToString()+" dan "+this.celana.celanaToString();
14    }
15 }
```

Class **Seragam** memiliki *attribute* yang bertipe class lain, yaitu class **Kaus** dan class **Celana**.

Objek “**muatan**” (seperti objek **k** dari class **Kaus**) tetap ada saat objek “**wadah**” (seperti objek **s** dari class **Seragam**) dihancurkan karena objek **k** *diinstansiasi* di luar objek **s**.

```
3 public class 010_Encapsulation {
4
5     public static void main(String[] args){
6         Celana c = new Celana();
7         Kaus k = new Kaus();
8         Seragam s = new Seragam(k, c);
9
10        k.setWarna("putih");
11        c.setWarna("merah");
12        System.out.println(s.seragamToString());
13    }
14 }
```

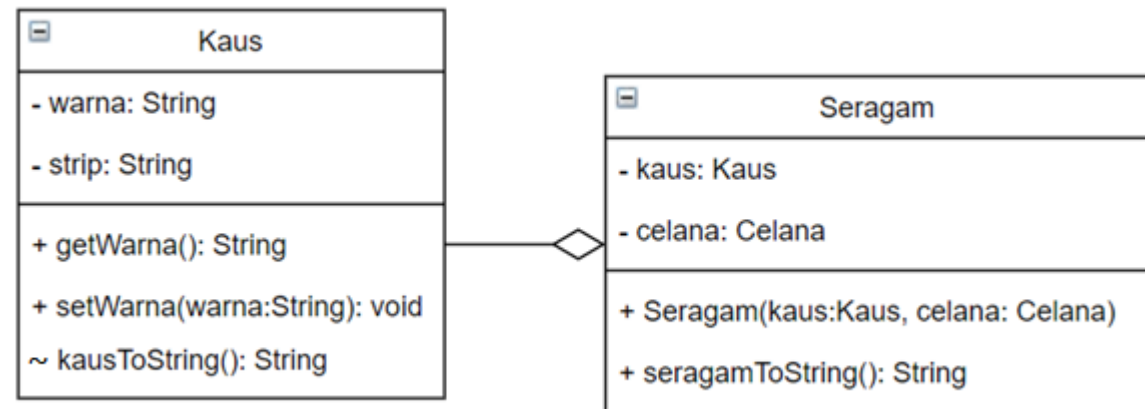
AGGREGATION — EXAMPLE

```
3 public class Kaus {
4     private String warna;
5     private String strip;
6
7     public String getWarna() {
8         return warna;
9     }
10
11    public void setWarna(String warna) {
12        this.warna = warna;
13    }
14
15    String kausToString() {
16        return "kaus "+this.warna+" strip "+this.strip;
17    }
18 }
```

```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6
7     public Seragam(Kaus kaus, Celana celana){
8         this.kaus = kaus;
9         this.celana = celana;
10    }
11
12    public String seragamToString() {
13        return this.kaus.kausToString()+" dan "+this.celana.celanaToString();
14    }
15 }
```

```
3 public class 010_Encapsulation {
4
5     public static void main(String[] args){
6         Celana c = new Celana();
7         Kaus k = new Kaus();
8         Seragam s = new Seragam(k, c);
9
10        k.setWarna("putih");
11        c.setWarna("merah");
12        System.out.println(s.seragamToString());
13    }
14 }
```

Class diagram:



COMPOSITION

1. Menggambarkan hubungan antara dua class di mana class yang satu *merupakan bagian dari class yang lain*.
2. Suatu class memiliki *attribute* yang bertipe class lain.
3. Bentuk khusus dari Aggregation, bedanya:
 - Objek “**muatan**” (contoh: objek **k** dari class **Kaus**) **dibuat saat** objek “**wadah**” (contoh: objek **s** dari class **Seragam**) dibuat.
 - Saat objek “**wadah**” **dimusnahkan** maka objek “**muatan**” juga ikut musnah.

COMPOSITION — EXAMPLE

1. Mengubah beberapa kode di class **Seragam** di contoh materi “Multiclass”.

```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6
7     public Seragam(){
8         this.kaus = new Kaus();
9         this.celana = new Celana();
10    }
11
12    public String seragamToString() {
13        return this.kaus.kausToString()+" dan "+this.celana.celanaToString();
14    }
15 }
```

2. Mengubah beberapa kode di **main()** method

```
3 public class 010_Encapsulation {
4
5     public static void main(String[] args){
6         Seragam s = new Seragam();
7         System.out.println(s.seragamToString());
8     }
9 }
```

3. Dapat dilihat di **main()** method, sudah **tidak ada** lagi instansiasi objek **k** dari class **Kaus** dan objek **c** dari class **Celana**.
4. Karena objek dari class **Kaus** dan objek dari class **Celana** diinstansiasi di dalam *constructor* class **Seragam**.
5. Karena itu, objek “muatan” (seperti objek kaus dari class Kaus) akan musnah saat objek “wadah” (seperti objek **s** dari class **Seragam**) dihancurkan.

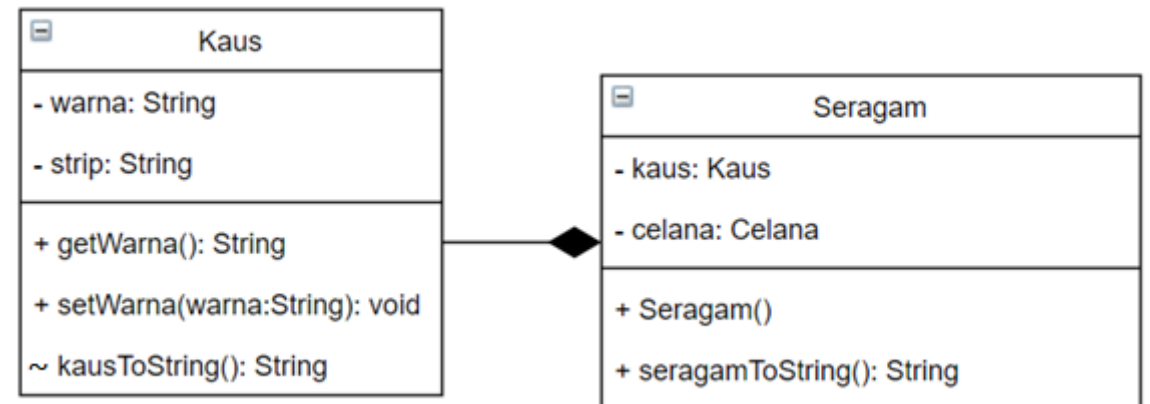
COMPOSITION — EXAMPLE

```
3 public class Kaus {
4     private String warna;
5     private String strip;
6
7     public String getWarna() {
8         return warna;
9     }
10
11    public void setWarna(String warna) {
12        this.warna = warna;
13    }
14
15    String kausToString() {
16        return "kaus "+this.warna+" strip "+this.strip;
17    }
18 }
```

```
3 public class Seragam {
4     private Kaus kaus;
5     private Celana celana;
6
7     public Seragam(){
8         this.kaus = new Kaus();
9         this.celana = new Celana();
10    }
11
12    public String seragamToString() {
13        return this.kaus.kausToString()+" dan "+this.celana.celanaToString();
14    }
15 }
```

```
3 public class 010_Encapsulation {
4
5     public static void main(String[] args){
6         Seragam s = new Seragam();
7         System.out.println(s.seragamToString());
8     }
9 }
```

Class diagram:



“TERIMA KASIH”

FUNDAMEN **PENGEMBANGAN APLIKASI**

Pertemuan 10

Program Studi Informatika - Program Sarjana
Fakultas Teknologi Industri