# Dasar-Dasar Pemrograman 2

## Tutorial 6 Kelas B, D, & E

### OOP Lanjutan

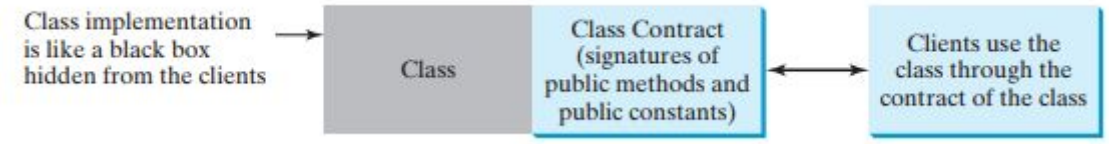
Kamis, 4 April 2019 - 16:00 WIB

Pada Tutorial/Lab sebelumnya, kalian telah mengetahui dasar-dasar dari pemrograman berbasis object pada Java yang terdiri dari class, object, attribute, dan method. Kali ini, kalian akan mempelajari bagaimana cara berpikir secara “object-oriented” dalam memodelkan penyelesaian sebuah masalah.

## Abstraction & Encapsulation

**Abstraction** (abstraksi) adalah konsep yang melibatkan pemisahan antara implementasi sebuah class dengan bagaimana class tersebut digunakan. Dalam praktiknya, pembuat class mendeskripsikan kegunaan dari class tersebut dan bagaimana cara menggunakan method serta data field dari class tersebut kepada user. User hanya perlu untuk apa atau kapan digunakannya method dan field yang disediakan. Implementasi internal dari method dan field “disembunyikan” dari user, sehingga user tidak perlu mengetahuinya. Penyembunyian atau “pembungkusan” implementasi internal ini disebut dengan istilah **encapsulation** (pembungkusan).

Penerapan dari kedua konsep ini dapat diilustrasikan dengan sebagai berikut:




(*Source*: Introduction to Java Programming, Comprehensive Version, 10th Edition. Daniel Y. Lang)

Contoh dari penerapan kedua konsep ini terdapat pada method static **sort()** yang ada pada class **Collections**. Kita tidak perlu mengetahui bagaimana implementasinya secara internal, namun kita hanya perlu mengetahui bahwa method tersebut dapat melakukan *sorting* terhadap elemen-elemen dari sebuah List.

Perlu diketahui bahwa abstraction yang dimaksud di sini adalah dalam artian secara umum, bukan abstraction dalam bentuk Abstract Class atau pun Interface (kedua hal tersebut akan dibahas pada materi berikutnya).

## Class Relationships

Karena Object-Oriented Programming sangat terkait dengan pemodelan dunia nyata, maka suatu class yang merepresentasikan suatu “entitas” dapat memiliki hubungan dengan class-class lainnya.

### Association

Association (asosiasi) adalah hubungan yang menyatakan aktivitas di antara dua class yang saling “berkomunikasi”. Sebagai contoh, dua class **Student** dan **Mentor** memiliki asosiasi sebagai berikut: **Student** dapat diajari oleh banyak **Mentor**, sedangkan **Mentor** dapat mengajari banyak **Student**. Berikut contoh UML diagramnya:


### Aggregation

Aggregation adalah bentuk khusus dari association yang merepresentasikan hubungan kepemilikan (has-a) antara dua object (tidak harus dari dua class yang berbeda). Object pemilik (owner) disebut *aggregating object* (class-nya disebut *aggregating class*) dan object yang dimiliki oleh owner disebut *aggregated object* (class-nya disebut *aggregated class*). Sebagai contoh, antara object **Student** dan **Address** terdapat aggregation berupa **Student** has-a **Address**. Berikut contoh UML diagramnya:


### Composition

Composition adalah bentuk khusus dari aggregation, di mana sebuah *aggregated object* hanya dimiliki oleh suatu *aggregating object* tertentu. Misalkan object **Name** hanya dapat dimiliki oleh object **Student**, bukan object lain. Berikut contoh UML diagramnya:

**Aggregation vs Composition**

Kedua hubungan di atas dapat diimplementasikan dengan cara yang mirip, salah satunya *aggregated object* yang direpresentasikan sebagai data field (attribute) pada *aggregating class*. Lantas, apakah perbedaan antara keduanya?

- Aggregation menyatakan hubungan di mana *aggregated object* dan *aggregating object* dapat berdiri sendiri. Artinya, seandainya hubungan antara keduanya ditiadakan, maka masing-masing dapat berdiri sendiri tanpa harus bergantung satu sama lain. Pada contoh di atas, jika class **Student** dihapus, maka **Address** bisa saja dihubungkan dengan class lain, misalkan **Employee**.

```
Address depok = new Address("Depok");
Student rey = new Student("Rey", depok);
// depok tidak bergantung pada Student (rey)
Employee nida = new Employee("Nida", depok);
```

- Composition berbanding terbalik dengan aggregation, di mana pihak *aggregated object* sangat bergantung terhadap eksistensi dari *aggregating object*. Seandainya hubungan antara keduanya ditiadakan, maka pihak *aggregated object* tidak dapat berdiri sendiri. Pada contoh di atas, jika class **Student** dihapus, maka **Name** tidak dapat berdiri sendiri karena ia sangat bergantung pada **Student**. Artinya, terjadi dependensi yang lebih kuat dari *aggregating object* terhadap owner-nya.

```
public class Student {
  public Student(String firstName, String lastName) {
    this.name = new Name(firstName, lastName);
  }
}

// in main
Student rey = new Student("Reynaldo", "Wijaya");
// kita tidak bisa menggunakan Name("Reynaldo", "Wijaya") di tempat lain!
// jika membuat Student lagi, akan dibuat OBJEK YANG BERBEDA.
Student reyLagi = new Student("Reynaldo", "Wijaya");
```

# Veedios

Dek Depi akan membuat aplikasi baru yang berupa *social-platform* untuk saling berbagi video. Nama app nya adalah *Veedios*. Dalam app ini, ada yang menjadi Publisher (orang yang mengunggah) video untuk dilihat oleh para Subscriber mereka. Lalu, ada yang menjadi Subscriber yang memiliki Timeline mereka masing-masing yang berisikan Video-Video yang diunggah oleh Publisher yang mereka subscribe.

**Kamu diminta untuk membuat program yang mensimulasikan aplikasi Veedios ini.**

## Spesifikasi Program :

Kamu akan diberikan keleluasaan untuk menggunakan OOP sesuai dengan pemahaman kalian (dan kebutuhan pemenuhan permintaan soal). Oleh karena itu, **tidak diberikan template** apapun untuk tutorial kali ini. **Kalian akan diberikan Main class yang akan di run, pastikan kode kalian memenuhi kriteria file ini.**

Berikut sedikit *story* untuk menggambarkan aplikasi ini :

1. **Publisher**
   Publisher menyimpan **koleksi Subscribers** mereka. Publisher bisa melakukan **upload(video)** yang akan menambahkan video yang diupload ke **koleksi video si Publisher** DAN ke **timeline (koleksi video) setiap Subscriber** mereka. +
2. **Subscriber**
   Setiap Subscriber menyimpan **koleksi Publisher** yang mereka subscribed. Setiap Subscriber juga memiliki Timeline **(koleksi video) yang berisi gabungan dari seluruh video-video semua Publisher** yang mereka subscribe. Subscriber juga bisa **menonton video**, hal ini akan dibahas di bagian Video. Dan tentunya harus bisa **subscribe(publisher)** dan **unsubscribe(publisher)**. Subscribe = bisa menerima video baru dari publisher tersebut, Unsubscribe = tidak menerima video baru dari publisher itu lagi.
3. **Video**
   Setiap Video hanya memiliki **title** , **counter berapa kali ia ditonton** , dan **Publisher** . Jika suatu video ditonton oleh seorang Subscriber, maka **counter nya akan bertambah 1**.
4. **Main**
   Main akan disediakan.

Kesimpulan singkat tentang aplikasi ini, kurang lebih gambaran minimal dari requirement di atas (kalian mungkin perlu mengimplementasikan method selain method yang diberikan disini) :

1. **Publisher**
   *Name*, Subsciber[ ] , Video[ ], uploadVideo(Video), printSubscribers()
2. **Subscriber**
   *Name*, Publisher[ ], Video[ ], watch(Video), subscribe(Publisher), unsubscribe(Publisher), printTimeline(), printPublishers()
3. **Video**
   *Title*, *ViewCounter*, getDetails()
   * notes : sekilas terlihat banyak, tapi method-method di atas mirip-mirip dan tidak besar

**Dalam pengerjaan lab kali ini sangat disarankan untuk melihat Main.java yang sudah disediakan untuk mengerti alur dan format output program!**

## Komponen Penilaian :

| Komponen | Penjelasan | Bobot |
|---|---|---|
| Implementasi Subscriber | Implementasi pemenuhan soal sesuai konsep OOP yang telah diajarkan terhadap aspek Subscriber. | 30 % |
| Implementasi Publisher | Implementasi pemenuhan soal sesuai konsep OOP yang telah diajarkan terhadap aspek Publisher. | 30 % |
| Implementasi Video | Implementasi pemenuhan soal sesuai konsep OOP yang telah diajarkan terhadap aspek Video. | 15 % |
| Ketepatan Output | Ketepatan Output ketika Main dijalankan (Pemenuhan requirement client). | 15 % |
| Kerapian | Penulisan program mengikuti kaidah dan konvensi yang telah diajarkan. Program ditulis dengan rapi, terstruktur, dan disertakan oleh dokumentasi secukupnya. | 10 % |

## Deadline :

Selasa, 19 Maret 2019
Pukul **17:40 WIB**

## Persentase Nilai Total Terhadap Lama Keterlambatan :

| dari | hingga | persentase |
|---|---|---|
| 0:00:00 | 0:00:00 | 100% |
| 0:00:01 | 0:10:00 | 85% |
| 0:10:01 | 0:20:00 | 70% |
| 0:20:01 | 0:30:00 | 50% |
| 0:30:01 | 24:00:00 | 25% |

## Format Pengumpulan :

**Zip** semua class yang kamu buat dan kumpulkan di slot pengumpulan yang telah disediakan di SCeLE dengan format :

**[Kode Asdos]_[Nama]_[Kelas]_[NPM]_Lab[X].zip**

**Contoh :**

**JO_JonathanChristopherJakub_C_1706040151_Lab0.zip**

## Acknowledged Lecturers :


- Fariz Darari, S.Kom, M.Sc., Ph.D.


## Authors :


- JO
- SAM
- KC
- DN