Dasar-Dasar Pemrograman 2

Tutorial 6 Kelas C dan F

OOP Lanjutan

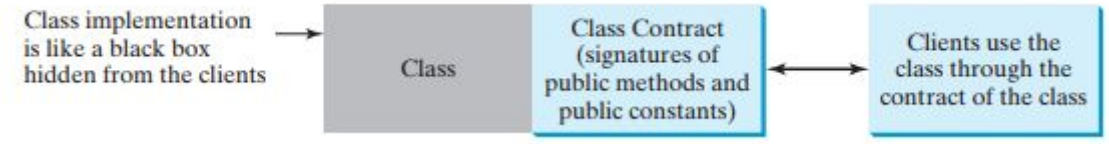
Selasa, 2 April 2019 - 16:00 WIB

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 **Person**, 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");
```

DnD - Dungeon dan Dek Depe

Source : /www.reddit.com/r/TheElderMemesV/comments/8k4ih2/when_youre_the_first_awake_at_a_sleepover_and/

“Hey, kamu. Akhirnya bangun juga. Kamu mencoba untuk melewati boss itu, kan? Kamu ingin bermain DnD tapi masih kurang level.” - Dek Depe

Dek Depe menemukan kamu terbangun dari tidurmu karena kamu tertidur saat ingin bermain DnD - Dungeons and Dek Depe. Kelelahan karena mengulang level yang sama berkali - kali. Kamu kurang level dan skill - skill kamu belum di *upgrade*.

Sudah berhari - hari kamu bermain DnD namun belum bisa melewati satu boss itu. Dek Depe ingin membantu kamu menyelesaikan game itu. Dek Depe ingin refreshing dari deadline dan tugas - tugas yang ada.(continue)

Spesifikasi Program :

*Kamu **wajib** menggunakan template yang disediakan, method dan attribut di bawah ini harus diimplementasikan pada template. Selain itu, template sudah berisi method lainnya yang mungkin dapat kamu manfaatkan dalam menyelesaikan tugas ini.

CLASS:

- Hero
- Boss
- Weapon
- Element

SPESIFIKASI CLASS **HERO:**

- Setiap **Hero** memiliki:
    - Name
    - Weapon
- Setiap **Hero** dapat menyerang **Boss** dengan **Weapon** yang dimiliki **Hero** tersebut dan setiap kali penyerangan akan mencetak:

```
<Hero name> menyerang <Boss name> dengan senjata <Weapon Name> sebesar <Weapon Total Damage>, HP <Boss name> sekarang <Boss HP>.
```

(Apabila **Hero** menyerang **Boss** dengan HP 0, maka **Hero** yang tersisa di dalam list tersebut tidak perlu lagi menyerang **Boss** tersebut)

- Menambakan **Weapon** yang *total damage*-nya merupakan *damage* dari *attack* Hero tersebut dan akan mencetak:

```
<Hero Name> berhasil memakai <Weapon Name> !
```

Jika **Hero** sudah pernah memiliki **Weapon**, maka **Hero** tersebut tidak bisa menambahkan **Weapon** lagi dan akan mencetak:

```
<Hero Name> sudah memiliki senjata.
```

SPESIFIKASI CLASS BOSS:

- Setiap **Boss** memiliki:
    - Name
    - HP (Health Point)
- **Boss** tidak dapat melakukan apapun selain menyaksikan dirinya diserang.
- HP **Boss** akan berkurang sebesar *damage* yang ia terima dari tiap **Hero** setiap kali diserang.
- HP minimum **Boss** adalah 0.

SPESIFIKASI CLASS WEAPON:

- Setiap **Weapon** memiliki:
  - Name
  - Element
  - Raw Damage
- Setiap **Weapon** dapat dipasang **Element** dan akan mencetak:

```
Elemen <Jenis Element> telah terpasang di <Nama Weapon>
```

- **Weapon** yang sudah pernah dipasang elemen tidak bisa ditambahkan element baru dan akan mencetak:

```
<Nama Weapon> sudah memiliki Elemen
```

- Meskipun tidak bisa ditambahkan elemen baru, tetapi setiap **Weapon** dapat diganti **Element**-nya dengan elemen yang berbeda maupun yang sama dan akan mencetak:

```
Elemen <Nama Weapon> diganti menjadi <Jenis Element>
```

- Damage total yang dimiliki **Weapon** merupakan gabungan dari *raw damage* yang dimiliki **Weapon** dengan *damage* yang dihasilkan oleh **Element**.

SPESIFIKASI CLASS ELEMENT

- Setiap element memiliki :
  - Type
  - Element Damage.
- Hanya ada 4 elemen dalam program yaitu element **Water, Earth, Fire, Air**.
- Setiap element memiliki nilai element damage yang unik untuk setiap tipe dengan spesifikasi sebagai berikut :
  - Air : 20
  - Tanah : 30
  - Api : 50
  - Angin : 25

Jalankan File Simulator.java yang disediakan pada Template untuk mengecek kebenaran program

Contoh Input dan Output

```
Masukan jumlah hero : 4
<Member 1>
Atta
<Member 2>
Budi
<Member 3>
Chelly
<Member 4>
Duboi
Masukan jumlah weapon : 9
<Weapon 1>
Rocket Launcher;50
<Weapon 2>
Keris 7 Turunan;100
<Weapon 3>
Palu Besi;40
<Weapon 4>
Chaos Blade;80
<Weapon 5>
Arit Sakti;40
<Weapon 6>
Kusabimaru;100
<Weapon 7>
Infinity Gauntlet;70
<Weapon 8>
Virtuous Treaty;30
<Weapon 9>
Excalibur;1500
!! Boss datang !!
<Boss>
Dek Depe;420
!! Masa Persiapan !!
PASANG ELEMENT;Keris 7 Turunan;Api
Elemen Api telah terpasang di Keris 7 Turunan !
PASANG ELEMENT;Infinity Gauntlet;Angin
Elemen Angin telah terpasang di Infinity Gauntlet !
PASANG ELEMENT;Virtuous Treaty;Air
Elemen Air telah terpasang di Virtuous Treaty!
PASANG ELEMENT;Chaos Blade;Api
Elemen Api telah terpasang di Chaos Blade !
PASANG ELEMENT;Virtuous Treaty;Api
Virtuous Treaty sudah memiliki Element !
PASANG ELEMENT;Rocket Launcher;Air
Elemen Air telah terpasang di Rocket Launcher !
PASANG ELEMENT;Excalibur;Tanah
Elemen Tanah telah terpasang di Excalibur !
PASANG ELEMENT;Kusabimaru;Tanah
```

```
Elemen Tanah telah terpasang di Kusabimaru !
GANTI ELEMENT;Excalibur;Api
Elemen Excalibur diganti menjadi Api !
PASANG SENJATA;Atta;Keris 7 Turunan
Atta berhasil memakai Keris 7 Turunan Api !
PASANG SENJATA;Chelly;Kusabimaru
Chelly berhasil memakai Kusabimaru Tanah !
PASANG SENJATA;Chelly;Excalibur
Chelly sudah memiliki senjata !
MULAI MENYERANG
Tidak bisa menyerang karena belum semua hero memakai senjata
PASANG SENJATA;Budi;Chaos Blade
Budi berhasil memakai Chaos Blade Api !
PASANG SENJATA;Duboi;Palu Besi
Duboi berhasil memakai Palu Besi !
MULAI MENYERANG
Atta menyerang Dek Depe dengan senjata Keris 7 Turunan Api sebesar 150,
HP Dek Depe sekarang 270
Budi menyerang Dek Depe dengan senjata Chaos Blade Api sebesar 130, HP
Dek Depe sekarang 140
Chelly menyerang Dek Depe dengan senjata Kusabimaru Tanah sebesar 130,
HP Dek Depe sekarang 10
Duboi menyerang Dek Depe dengan senjata Palu Besi sebesar 40, HP Dek
Depe sekarang 0
```

Pastikan untuk meng-compile setiap program yang diubah. Untuk mempermudah, bisa menggunakan `javac *.java`.

Untuk mengecek kebenaran program, Anda dapat melakukan redirect input ke input.txt (ada di dalam template) dengan cara `java Simulator < input.txt` dan bandingkan hasilnya dengan output.txt (misal dengan menggunakan *diffchecker*).

Komponen Penilaian :

Komponen	Penjelasan	Bobot
Implementasi Class Boss	Melakukan implementasi pada bagian-bagian yang diminta dalam *class Boss*, dengan menggunakan paradigma *object oriented*	5%
Implementasi Class Element	Melakukan implementasi pada bagian-bagian yang diminta dalam *class Element*, dengan menggunakan paradigma *object oriented*	5%
Implementasi Class Hero	Melakukan implementasi pada bagian-bagian yang diminta dalam *class Hero*, dengan menggunakan paradigma *object oriented*	10%
Implementasi Class Simulator	Melakukan implementasi pada bagian-bagian yang diminta dalam *class Simulator*, dengan menggunakan paradigma *object oriented*	20%
Implementasi Class Weapon	Melakukan implementasi pada bagian-bagian yang diminta dalam *class Weapon*, dengan menggunakan paradigma *object oriented*	10%
Kebenaran Program	Kesesuaian Output program dengan hasil yang diinginkan	35%
Kerapian Kode	Penulisan program mengikuti kaidah dan konvensi yang telah diajarkan. Program ditulis dengan rapi, terstruktur, dan disertakan oleh dokumentasi secukupnya	15%

Deadline :Selasa, 2 April 2019

Pukul **17:40 WIB**

Penalti Terlambat

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 :

Kumpulkan hasil zip template yang sudah diimplementasikan kode Anda di slot pengumpulan yang telah disediakan di SCeLE dengan format :

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

Contoh:
TES_DemoSuremo_C_1234567891_Lab6.zip

Selamat Bekerja !