Modul 3
Enkapsulasi

1. Tujuan

Setelah melakukan percobaan pada modul ini, mahasiswa memahami konsep:

1. Enkapsulasi (access level modifier, setter dan getter)
2. Constructor
3. Memahami notasi terkait access level modifier pada UML Class Diagram

2. Pendahuluan

Pada pertemuan pertama dan kedua telah dibahas konsep dasar dari pemrograman berbasis objek (PBO), perbedaan antara pemrograman berbasis objek dengan pemrograman struktural, dan konsep class dan object. Selanjutnya pada modul ini akan dibahas konsep enkapsulasi dan notasi yang ada pada UML Class diagram.

2.1. Enkapsulasi

Definisi:

- Penyatuan/penggabungan atribut dan method dari suatu objek menjadi suatu kesatuan
- Pembatasan akses langsung terhadap komponen dari suatu objek

Tujuan enkapsulasi:

- Penyembunyian struktur internal dari suatu objek → information hiding/data hiding
- Melindungi atribut dari perubahan di luar class secara random. Atribut dapat dibuat menjadi *read-only* atau *write-only*
- Mempermudah implementasi perubahan requirements
- Mempermudah pengujian unit sistem

Mekanisme enkapsulasi:

- Mengeset *access level modifier* atribut menjadi private sehingga tidak dapat diakses secara langsung dari luar class
- Menyediakan *getter* dan *setter* sebagai cara untuk mengakses atau memodifikasi private attribute

2.2.1. Access Level Modifier

Terdapat 4 access level modifier yaitu:

- *public* – dapat diakases dari mana saja
- *protected* – dapat diakases di luar package menggunakan subclass (membuat inheritance)
- no modifier (*package-private*) – hanya dapat diakses di dalam package yang sama
- *private* – hanya dapat diakses di dalam kelas yang sama

Attribute dan method memiliki 4 jenis *access level modifier* di atas, tetapi class hanya memiliki 2 jenis *access level modifier* saja, yaitu *public* dan *no modifier*.

Tabel 1. 1 Access Level Modifier

Modifier	Class	Package	Subclass	Outside Package
public	v	v	v	v
protected	v	v	v	
no modifier	v	v		
private	v			

2.2.2. Getter dan Setter

Getter

- Public method yang berfungsi mengembalikan nilai dari atribut private
- Ada return value

Setter

- Public method yang berfungsi untuk memanipulasi nilai dari atribut private
- Tidak ada return value

2.2.3. Read-Only dan Write-Only

Read-only attribute

- Atribut yang hanya memiliki getter, tetapi tidak memiliki setter
- Nilai atribut dapat diakses dari dalam maupun luar class
- Modifikasi nilai atribut hanya dapat dilakukan di dalam class nya saja

Write-only attribute

- Atribut yang hanya memiliki setter, tetapi tidak memiliki getter
- Modifikasi nilai atribut dapat dilakukan dari dalam maupun luar class
- Nilai atribut tersebut hanya dapat diakses dari class nya saja

2.3. Constructor

Constructor merupakan method yang digunakan untuk menginstansiasi objek dari suatu class. Jika tidak dibuat secara eksplisit, java telah menyediakan constructor default tanpa parameter, artinya objek dibuat tanpa meng-assign nilai atribut. Jika terdapat kebutuhan yang mewajibkan beberapa atau nilai atribut harus diberi nilai saat objek dibuat, maka kita perlu mendefinisikan constructor sendiri.

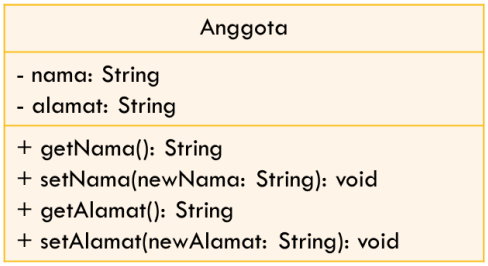
Beberapa aturan deklarasi constructor:

- Nama constructor harus sama dengan nama class
- Constructor tidak memiliki return type

2.4. Notasi UML Class Diagram

Notasi access level modifier pada UML class diagram adalah sebagai berikut:

- Tanda plus (+) untuk public
- Tanda pagar (#) untuk protected
- Tanda minus (-) untuk private
- Untuk no-modifier tidak diberi notasi



Gambar 1. 1 UML Class Diagram

3. Percobaan

3.1 Percobaan 1 - Constructor

1. Buat folder baru dengan nama Praktikum03
2. Misalkan di sebuah sistem informasi, terdapat class **User** yang memiliki atribut username, password, email, dan name. Class User dapat diimplementasikan sebagai berikut.

```
public class User {
    public String username;
    public String email;
    public String password;
    public String name;

    public void displayInfo() {
        System.out.println("Username: " + username);
        System.out.println("Email: " + email);
        System.out.println("Password: " + password);
        System.out.println("Name: " + name);
    }
}
```

3. Buat class UserDemo kemudian cobalah membuat objek baru dengan nama user1. Objek user1 diinstansiasi dengan cara memanggil constructor User(). Meskipun constructor User() tidak secara eksplisit dituliskan pada class User, method ini tetap dapat dipanggil karena telah disediakan secara default oleh Java compiler.

```
public class UserDemo {
    Run | Debug
    public static void main(String[] args) {
        User user1 = new User();
        user1.displayInfo();
    }
}
```

4. Run UserDemo.java maka output yang tampil adalah sebagai berikut.

```
Username: null
Email: null
Password: null
Name: null
```

Proses instansiasi adalah pembuatan objek baru dan inisialisasi nilai atribut dengan nilai default (null untuk atribut bertipe data String, 0 untuk atribut bertipe data numerik, dan false untuk atribut bertipe data boolean).

5. Misalnya terdapat requirement bahwa saat suatu objek user dibuat, user tersebut harus sudah memiliki nilai username dan email. Di samping itu, atribut password diberi nilai default, misalnya "polinema123". Dengan kebutuhan tersebut, kita harus membuat sebuah constructor baru sebagai berikut:

```
public User(String username, String email) {
    this.username = username;
    this.email = email;
    this.password = "polinema123";
}
```

6. Setelah kita menyediakan constructor baru secara eksplisit, maka constructor default yaitu User() tidak dapat digunakan lagi kecuali dituliskan juga pada class User. (*Multiple constructor akan dibahas pada materi overloading & overriding*).

```
public class UserDemo
    Run | Debug
    public static voi
        User user1 = new User();
        user1.displayInfo();
    }
}
```

The constructor User() is undefined Java(134217858)

Praktikum03.User

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

7. Instansiasi objek user baru dengan constructor yang telah dibuat pada langkah no 4 bisa dilakukan dengan cara berikut:

```
public static void main(String[] args) {
    User user1 = new User("annisa.nadya", "annisa.nadya@gmail.com");
    user1.displayInfo();
}
```

8. Hasilnya sebagai berikut:

```
Username: annisa.nadya
Email: annisa.nadya@gmail.com
Password: polinema123
Name: null
```

3.5 Pertanyaan

1. Apa kegunaan dari constructor?
2. Sebutkan keistimewaan constructor dibanding method lain pada suatu class.
3. Lakukan analisa dan buat kesimpulan apakah constructor bisa memiliki access level modifier private?

3.1 Percobaan 1 – Tanpa Enkapsulasi

Didalam percobaan enkapsulasi, buatlah class Motor yang memiliki atribut platNomor, isMesinOn (bernilai true jika mesin sedang menyala dan false jika tidak menyala), dan kecepatan serta method displayInfo() untuk menampilkan status motor. UML class diagram class Motor adalah sebagai berikut:

Motor
+ platNomor: String + isMesinOn: boolean + kecepatan: int
+displayInfo(): void

1. Buat class **Motor**

```java
public class Motor {
    public String platNomor;
    public boolean isMesinOn;
    public int kecepatan;

    public void displayInfo(){
        System.out.println("Plat Nomor: " + this.platNomor);
        System.out.println("Status Mesin: " + (this.isMesinOn ? "On" : "Off"));
        System.out.println("Kecepatan: " + this.kecepatan);
        System.out.println("============================");
    }
}
```

2. Kemudian buat class MotorDemo, ketikkan kode berikut ini.

```java
public class MotorDemo {
    Run | Debug
    public static void main(String[] args) {
        Motor motor1 = new Motor();
        motor1.displayInfo();

        motor1.platNomor = "B 0838 XZ";
        motor1.kecepatan = 50;
        motor1.displayInfo();
    }
}
```

3. Hasilnya adalah sebagai berikut:

```
Plat Nomor: null
Status Mesin: Off
Kecepatan: 0
============================
Plat Nomor: B 0838 XZ
Status Mesin: Off
Kecepatan: 50
============================
```

4. Selanjutnya tambahkan 2 objek motor lagi di class MotorDemo.java

```java
Motor motor2 = new Motor();
motor2.platNomor = "N 9840 AB";
motor2.isMesinOn = true;
motor2.kecepatan = 40;
motor2.displayInfo();

Motor motor3 = new Motor();
motor3.platNomor = "D 8343 CV";
motor3.kecepatan = 60;
motor3.displayInfo();
```

5. Hasilnya sebagai berikut

```
Plat Nomor: B 0838 XZ
Status Mesin: Off
Kecepatan: 50
============================
Plat Nomor: N 9840 AB
Status Mesin: On
Kecepatan: 40
============================
Plat Nomor: D 8343 CV
Status Mesin: Off
Kecepatan: 60
============================
```

6. Dari hasil di atas, adakah yang janggal?

   Pada motor1 dengan plat “B 0838 XZ”, kecepatannya dapat berubah dari 0 ke 50 padahal

mesin motor masih dalam kondisi Off. Bagaimana mungkin atribut kecepatan bernilai 50 padahal mesin masih Off? Hal ini karena belum tersedia kontrol/batasan terhadap atribut kecepatan. Padahal, objek di dunia nyata selalu memiliki batasan/aturan dalam memberi nilai atribut serta mekanisme bagaimana objek tersebut dapat berfungsi/melakukan sesuatu. Misalnya motor yang harus dalam keadaan menyala ketika kecepatan lebih dari 0. Kejanggalan ini juga terjadi pada motor ketiga dengan plat nomor "D 8343 CV".

7. Untuk mengatasi hal tersebut, nilai kecepatan baru perlu dicek terlebih dahulu sebelum di-assign ke atribut kecepatan

```
motor1.platNomor = "B 0838 XZ";

int kecepatanBaru = 50;

if (!motor1.isMesinOn && kecepatanBaru > 0) {
    System.out.println("Kecepatan tidak boleh lebih dari 0 jika mesin off");
}
else {
    motor1.kecepatan = kecepatanBaru;
}

motor1.displayInfo();
```

8. Lakukan pengecekan yang sama untuk motor2 dan motor3

```java
Motor motor2 = new Motor();
motor2.platNomor = "N 9840 AB";
motor2.isMesinOn = true;

kecepatanBaru = 40;

if (!motor2.isMesinOn && kecepatanBaru > 0) {
    System.out.println("Kecepatan tidak boleh lebih dari 0 jika mesin off");
}
else {
    motor2.kecepatan = kecepatanBaru;
}
motor2.displayInfo();

Motor motor3 = new Motor();
motor3.platNomor = "D 8343 CV";
kecepatanBaru = 60;

if (!motor1.isMesinOn && kecepatanBaru > 0) {
    System.out.println("Kecepatan tidak boleh lebih dari 0 jika mesin off");
}
else {
    motor1.kecepatan = kecepatanBaru;
}
motor3.displayInfo();
```

9. Run MotorDemo.java dan perhatikan bahwa sudah terdapat validasi nilai kecepatan terhadap status mesin untuk setiap objek motor

```
Plat Nomor: null
Status Mesin: Off
Kecepatan: 0
============================
Kecepatan tidak boleh lebih dari 0 jika mesin off
Plat Nomor: B 0838 XZ
Status Mesin: Off
Kecepatan: 0
============================
Plat Nomor: N 9840 AB
Status Mesin: On
Kecepatan: 40
============================
Kecepatan tidak boleh lebih dari 0 jika mesin off
Plat Nomor: D 8343 CV
Status Mesin: Off
Kecepatan: 0
============================
```

3.2 Percobaan 2 – Enkapsulasi

1. Bayangkan bahwa developer baru saja ingat bahwa seharusnya kecepatan tidak boleh lebih

dari 0 jika status mesin tidak menyala setelah membuat 20 objek motor di MotorDemo.java, 10 objek motor di MotorDemo2.java, 25 objek MotorDemo3.java? Pengecekan harus dilakukan 55 kali.

2. Lalu, bagaimana kita bisa memperbaiki class Motor diatas agar dapat digunakan dengan baik? Di sinilah pentingnya melakukan enkapsulasi dalam pemrograman berorientasi objek sehingga perubahan requirement hanya perlu dihandle pada "gerbang" yang dikelola di dalam class tersebut.

   Pada OOP, konsep enkapsulasi diimplementasikan dengan cara:

   a. Menyembunyikan atribut internal (platNomor, isMesinOn, dan kecepatan) dari luar/class lain dengan mengubah access level modifier menjadi private
   b. Menyediakan setter dan getter untuk memanipulasi dan mengakses nilai atribut tersebut

Motor
- platNomor: String - isMesinOn: Boolean - kecepatan: int
+displayStatus(): void +setPlatNomor(platNomor:String): void +getPlatNomor(): String +setIsMesinOn(isMesinOn:boolean): void +getIsMesinOn(): boolean +setKecepatan(kecepatan:int): void +getKecepatan(): int

3. Ubah access level modifier menjadi private

```
private String platNomor;
private boolean isMesinOn;
private int kecepatan;
```

4. Setelah berubah menjadi private, atribut platNomor, isMesinOn, dan kecepatan tidak bisa diakses dari luar class (muncul error)

```
Motor motor1 = new Motor();
motor1.displayInfo();

motor1.platNomor = "B 0838 XZ";

int kecepatanBaru = 50;

if (!motor1.isMesinOn && kecepatanBaru > 0) {
    System.out.println("Kecepatan tidak boleh lebih dari 0 jika mesin off");
}
else {
    motor1.kecepatan = kecepatanBaru;
}

motor1.displayInfo();
```

5. Selanjutnya perlu di buat setter dan getter untuk setiap atribut.

```
public String getPlatNomor() {
    return platNomor;
}

public void setPlatNomor(String platNomor) {
    this.platNomor = platNomor;
}

public boolean getIsMesinOn() {
    return isMesinOn;
}

public void setIsMesinOn(boolean isMesinOn) {
    this.isMesinOn = isMesinOn;
}

public int getKecepatan() {
    return kecepatan;
}

public void setKecepatan(int kecepatan) {
    this.kecepatan = kecepatan;
}
```

6. Dengan enkapsulasi, nilai atribut diakses menggunakan getter dan dimanipulasi menggunakan setter sebagai berikut (belum ada validasi nilai kecepatan terhadap status mesin)

```java
public static void main(String[] args) {
    Motor motor1 = new Motor();
    motor1.displayInfo();

    motor1.setPlatNomor("B 0838 XZ");
    motor1.setKecepatan(50);
    motor1.displayInfo();

    Motor motor2 = new Motor();
    motor2.setPlatNomor("N 9840 AB");
    motor2.setIsMesinOn(true);
    motor2.setKecepatan(40);
    motor2.displayInfo();

    Motor motor3 = new Motor();
    motor3.setPlatNomor("D 8343 CV");
    motor3.setKecepatan(60);
    motor3.displayInfo();
}
```

7. Dengan menerapkan enkapsulasi, perubahan requirement di tengah implementasi program dapat dilakukan dengan lebih mudah. Pada setter kecepatan, dilakukan validasi nilai kecepatan terhadap status mesin sebagai berikut:

```
public void setKecepatan(int kecepatan) {
    if (!this.isMesinOn && kecepatan > 0) {
        System.out.println("Kecepatan tidak boleh lebih dari 0 jika mesin off");
    }
    else{
        this.kecepatan = kecepatan;
    }
}
```

8. Run MotorDemo.java. Hasilnya sebagai berikut:

```
Plat Nomor: null
Status Mesin: Off
Kecepatan: 0
============================
Kecepatan tidak boleh lebih dari 0 jika mesin off
Plat Nomor: B 0838 XZ
Status Mesin: Off
Kecepatan: 0
============================
Plat Nomor: N 9840 AB
Status Mesin: On
Kecepatan: 40
============================
Kecepatan tidak boleh lebih dari 0 jika mesin off
Plat Nomor: D 8343 CV
Status Mesin: Off
Kecepatan: 0
============================
```

9. Setter dan getter dipakai sebagai “gerbang” untuk mengakses atau memodifikasi atribut yang bernilai private. Hal ini akan membuat kontrol atau validasi atribut lebih mudah dilakukan. Jika ada perubahan requirement di kemudian hari, misalnya atribut kecepatan tidak boleh bernilai negatif, hanya perlu dilakukan modifikasi pada setKecepatan() tanpa perlu melakukan perubahan berulang kali di seluruh program yang melakukan assignment nilai kecepatan motor. Pengujian terhadap perubahan requirement juga dapat dilakukan pada scope yang lebih kecil, tanpa harus menguji keseluruhan sistem yang dikembangkan.

3.3 Pertanyaan

1. Pada class MotorDemo, saat kita menambah kecepatan untuk pertama kalinya, mengapa muncul peringatan “Kecepatan tidak boleh lebih dari 0 jika mesin off”?
2. Mengapat atribut merek, kecepatan, dan statusMesin sebaiknya diset private?
3. Apa fungsi dari setter dan getter?
4. Ubah class Motor sehingga kecepatan maksimalnya adalah 100
5. Ubah class Motor sehingga kecepatan nya tidak boleh nilai negatif

4. Tugas

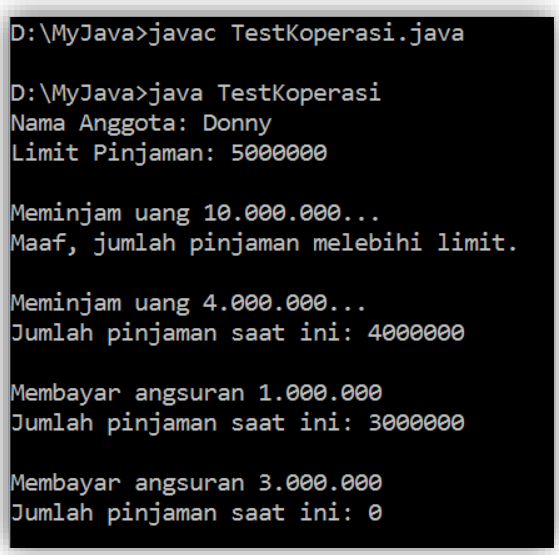
1. Pada sebuah sistem informasi koperasi simpan pinjam, terdapat class Anggota yang memiliki atribut antara lain nomor KTP, nama, limit peminjaman, dan jumlah pinjaman. Anggota dapat meminjam uang dengan limit peminjaman yang ditentukan. Anggota juga dapat mengangsur pinjaman. Ketika anggota tersebut mengangsur pinjaman, maka jumlah pinjaman akan berkurang sesuai dengan nominal yang diangsur.

   Buatlah class Anggota tersebut, berikan atribut, method dan constructor sesuai dengan kebutuhan. Uji dengan TestKoperasi berikut ini untuk memeriksa apakah class Anggota yang anda buat telah sesuai dengan yang diharapkan.

   Perhatikan bahwa nilai atribut pinjaman tidak dapat diubah secara random dari luar class, tetapi hanya dapat diubah melalui method pinjam() dan angsur()

```
public class TestKoperasi
{
        public static void main(String[] args)
        {
            Anggota anggota1 = new Anggota("111333444", "Donny", 5000000);

            System.out.println("Nama Anggota: " + anggota1.getNama());
            System.out.println("Limit Pinjaman: " + anggota1.getLimitPinjaman());

            System.out.println("\nMeminjam uang 10.000.000...");
            anggota1.pinjam(10000000);
            System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

            System.out.println("\nMeminjam uang 4.000.000...");
            anggota1.pinjam(4000000);
            System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

            System.out.println("\nMembayar angsuran 1.000.000");
            anggota1.angsur(1000000);
            System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

            System.out.println("\nMembayar angsuran 3.000.000");
            anggota1.angsur(3000000);
            System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());
        }
}
```

   Hasil yang diharapkan:



2. Modifikasi class Anggota agar nominal yang dapat diangsur minimal adalah 10% dari jumlah pinjaman saat ini. Jika mengangsur kurang dari 10%, maka muncul peringatan “Maaf, angsuran harus 10% dari jumlah pinjaman”.