Dasar-Dasar Pemrograman 2

Tutorial 7 Kelas C dan F

Inheritance

Selasa, 9 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 memodelkan masalah dengan menggunakan inheritance.

Inheritance dapat didefinisikan sebagai suatu proses di mana satu kelas memerlukan suatu properti (method atau atribut) dari kelas lainnya. Penggunaan inheritance sebuah informasi data dapat di-manage dalam struktur hierarchical. Suatu kelas yang meng-inherit properti dari kelas lain dikenal sebagai subclass dan kelas yang di-inherit dikenal sebagai superclass.

Example of Inheritance

Mari kita coba mengimplementasikan inheritance pada program sederhana berikut ini

```java
public class ProgramBangunDatar {
    public static void main(String[] args) {
        Persegi persegi = new Persegi(4);
        PersegiPanjang persegiPanjang = new PersegiPanjang(4, 2);
        persegi.hitungLuas();
        persegiPanjang.hitungLuas();
        persegi.hitungKeliling();
        persegiPanjang.hitungKeliling();
    }
}

class BangunDatar implements HitungBangunDatar {
    int hasil;

    public void hitungLuas() {
    }

    public void hitungKeliling() {
    }
}

class Persegi extends BangunDatar {
    int x;

    public Persegi(int x) {
        this.x = x;
    }

    public void hitungLuas() {
        hasil = x * x;
        System.out.println("Luas dari persegi adalah: " + hasil);
    }

    public void hitungKeliling() {
        hasil = x * 4;
        System.out.println("Keliling dari persegi adalah: " + hasil);
    }
}

class PersegiPanjang extends BangunDatar {
    int x, y;

    public PersegiPanjang(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void hitungLuas() {
        hasil = x * y;
        System.out.println("Luas dari persegi panjang adalah: " + hasil);
    }

    public void hitungKeliling() {
        hasil = (2 * x) + (2 * y);
        System.out.println("Keliling dari persegi panjang adalah: " + hasil);
    }
}
```

Perhatikan bahwa kelas Persegi dan *PersegiPanjang* (subclass) meng-inherit kelas *BangunDatar* (superclass), dan mengambil atribut hasil dan method hitungKeliling() serta hitungLuas() dari superclass-nya. Cara subclass

meng-inherit yaitu dengan keyword extends. Kita juga dapat memanggil method superclass atau mengambil atribut superclass melalui suatu subclass dengan memakai keyword super. Silahkan membuat kode seperti berikut, dan coba kalian eksekusikan.

```java
// Reference: https://www.tutorialspoint.com/java/java_inheritance.htm
class Super_class {
    int num = 20;
    // display method of superclass
    public void display() {
        System.out.println("This is the display method of superclass");
    }
}

public class Sub_class extends Super_class {
    int num = 10;
    // display method of sub class
    public void display() {
        System.out.println("This is the display method of subclass");
    }
    public void my_method() {
        // Instantiating subclass
        Sub_class sub = new Sub_class();
        // Invoking the display() method of sub class
        sub.display();
        // Invoking the display() method of superclass
        super.display();
        // printing the value of variable num of subclass
        System.out.println("value of the variable named num in sub class:"+ sub.num);
        // printing the value of variable num of superclass
        System.out.println("value of the variable named num in super class:"+ super.num);
    }
    public static void main(String args[]) {
        Sub_class obj = new Sub_class();
        obj.my_method();
    }
}
```

Selain dengan memanggil method dari superclass, kita juga bisa meng-construct superclass dengan menggunakan method 'super' pada constructor subclass. Silahkan buat program seperti di bawah ini:

```java
class Superclass {
    int age;

    Superclass(int age) {
        this.age = age;
    }
    public void getAge() {
        System.out.println("The value of the variable named age in super class is: " +age);
    }
}

public class Subclass extends Superclass {
    Subclass(int age) {
        super(age);
    }
    public static void main(String argd[]) {
        Subclass s = new Subclass(24);
        s.getAge();
    }
}
```

Perhatikan kembali bahwa method super pada constructor kelas *Subclass* digunakan untuk meng-construct kelas *Superclass*. Ini sangat berguna ketika superclass mempunyai semua atribut yang ada di subclass sehingga kita hanya perlu memanggil method super. Dan terakhir, kita juga dapat mengecek apakah suatu kelas merupakan instance dari kelas yang lain dengan memakai keyword instanceof. Silahkan mengimplementasikan program sederhana berikut untuk lebih memahami fungsionalitas instanceof

```java
// Reference: https://www.tutorialspoint.com/java/java_inheritance.htm
interface Animal{}
class Mammal implements Animal{}
public class Dog extends Mammal {
    public static void main(String args[]) {
        Mammal m = new Mammal();
        Dog d = new Dog();
        System.out.println(m instanceof Animal);
        System.out.println(d instanceof Mammal);
        System.out.println(d instanceof Animal);
    }
}
```

Perhatikan bahwa output dari program akan menjadi seperti berikut:

```
true
true
true
```

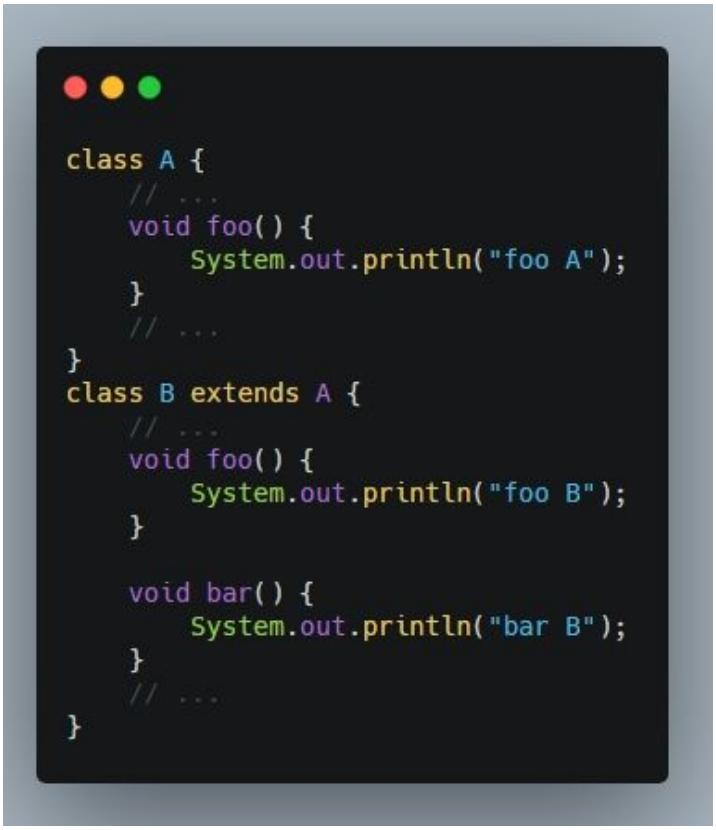
Hal tersebut menunjukkan bahwa m merupakan instance dari kelas Animal, dan d merupakan instance dari kelas Mammal serta Animal. Keyword instance sangat berguna untuk mengetahui apakah suatu kelas meng-extend suatu kelas atau meng-implement suatu interface.

Polymorphism?

Polymorphism dapat didefinisikan sebagai suatu konsep dimana suatu objek dapat memiliki banyak bentuk. Contoh nyatanya adalah H2O dapat memiliki bentuk cair, padat, dan gas.

Pada OOP, **Polymorphism** adalah kemampuan suatu variabel atau argumen untuk me-*refer* kepada suatu objek dari bermacam-macam kelas [Meyer pp. 224].

Lebih Lanjut tentang Polymorphism



```
class A {
    // ...
    void foo() {
        System.out.println("foo A");
    }
    // ...
}
class B extends A {
    // ...
    void foo() {
        System.out.println("foo B");
    }

    void bar() {
        System.out.println("bar B");
    }
    // ...
}
```

- Kelas A merupakan **superclass** dari kelas B.
- Kelas B merupakan **subclass** dari kelas A.

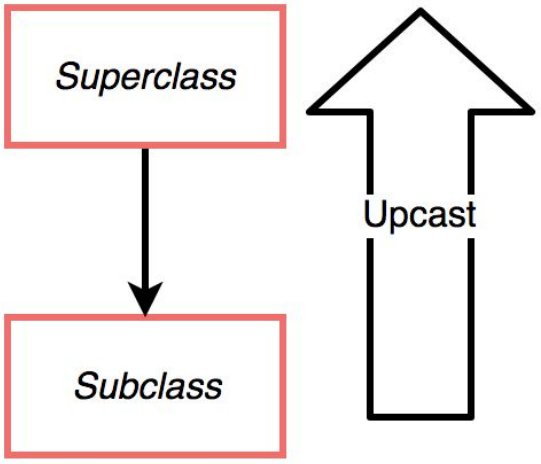
Hasil simulasi kedua kelas tersebut adalah:

```
A a = new A();
B b = new B();

a.foo(); // foo A
b.foo(); // foo B

b.bar(); // bar B
```

Dynamic

Dynamic disini artinya variabel x akan memiliki tipe yang berbeda-beda seiring program berjalan. Contohnya adalah proses upcasting dan downcasting di bawah ini.

Upcasting

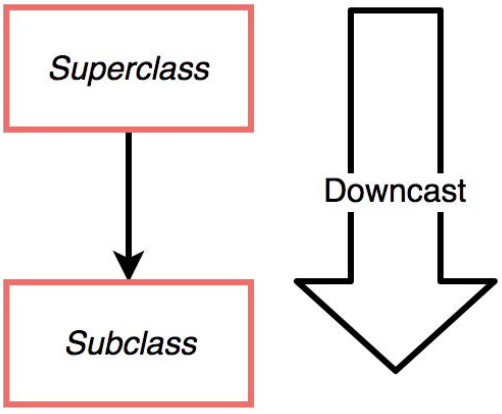


Upcasting adalah melakukan proses pengubahan referensi **subclass ke superclass**.

```
A a = new A();
B b = new B();

a.foo() // foo A

a = b; // Upcasting

a.foo(); // foo B
```

Perhatikan bahwa pada kondisi diatas, kita tidak dapat memanggil a.bar() karena Kelas A tidak memiliki method tersebut. Dengan begitu kita dapat menyimpulkan bahwa method yang spesifik pada kelas B akan menjadi **inactive**.

Downcasting



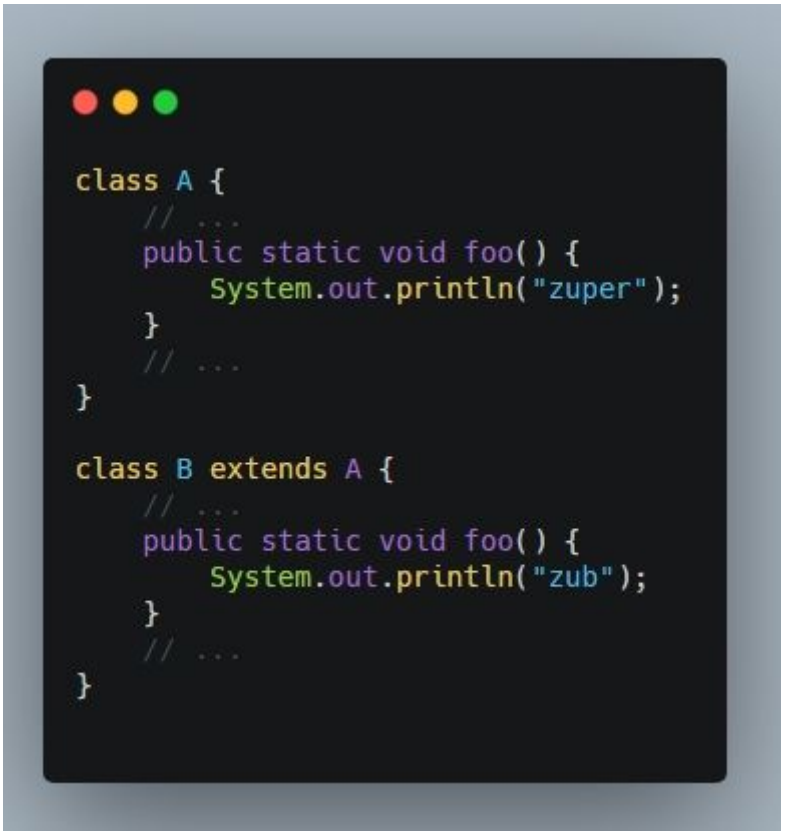
Downcasting adalah melakukan proses pengubahan referensi **superclass ke subclass**.

```
A a = new B(); // Upcasting

B b = (B) a; // Downcasting

b.bar(); // bar B
```

Proses **downcasting** dari variabel a ke variable b bertipe Kelas B memberi kita akses untuk memanggil method bar() yang khusus pada Kelas B. **PERHATIKAN** bahwa kita bisa memanggil b.bar() namun tidak dapat memanggil a.bar() (lihat penjelasan pada upcasting).

Static

Static type binding ini terjadi ketika proses compile terjadi. Hal ini karena perubahan type terjadi pada method yang **static**.



```
class A {
    // ...
    public static void foo() {
        System.out.println("zuper");
    }
    // ...
}

class B extends A {
    // ...
    public static void foo() {
        System.out.println("zub");
    }
    // ...
}
```

Bila kita melakukan simulasi pada kelas-kelas tersebut:

```
A.foo() // zuper
B.foo() // zuper
```

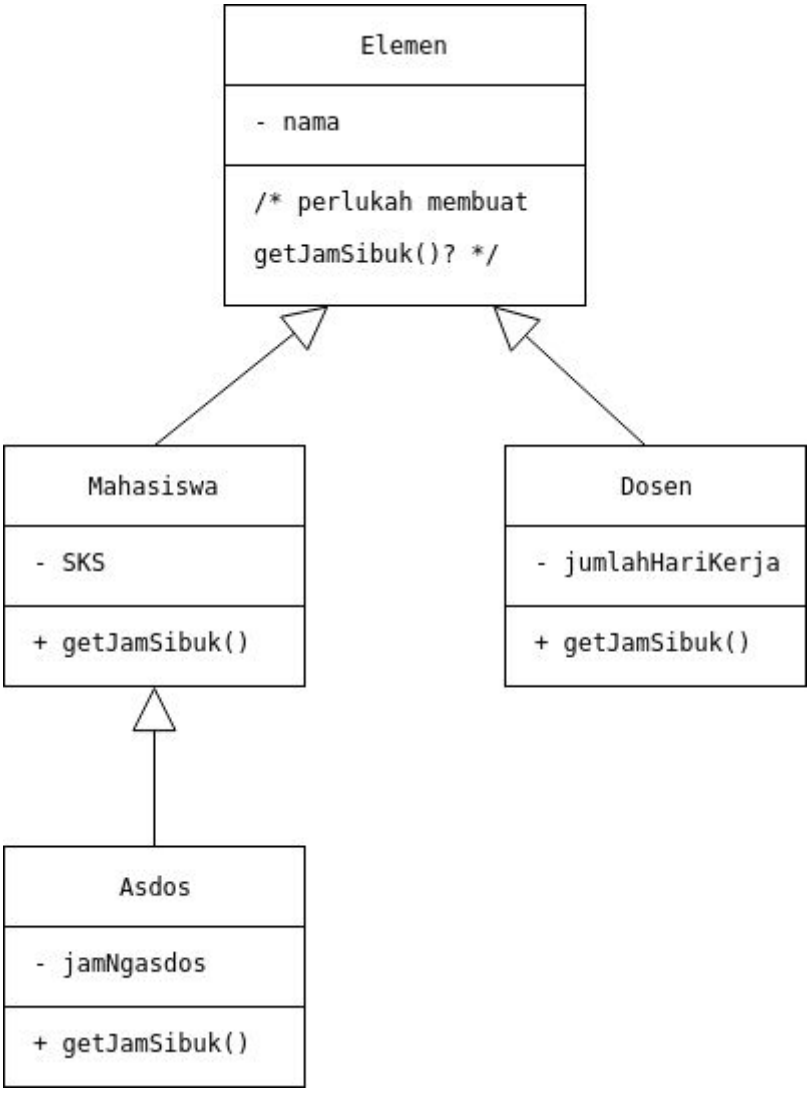
Method static foo() pada kelas B menjadi tidak aktif dan digantikan oleh method static foo() milik superclassnya.

Menghitung Jam Sibuk

Feynaldo merupakan mahasiswa Fasilkom yang suka mengeluh (jangan ditiru!). Setiap hari, dia selalu mengeluh bahwa tugasnya terlalu banyak. Di sisi lain, dia juga tidak pernah kapok untuk mengambil SKS yang banyak. Dek Depe yang heran dengan tingkah Feynaldo ini kemudian penasaran, **berapa jam sibuk seorang elemen Fasilkom dalam seminggu?** Dia ingin mengetahui jam sibuk dosen, mahasiswa, dan asdos.

Class yang Harus Dibuat

Buatlah class dengan diagram sebagai berikut:



Elemen

- Memiliki atribut nama
- Perhatikan bahwa di Simulator.java, Elemen tidak pernah diinstantiasi.

Dosen

- Memiliki atribut jumlahHariKerja
- jamSibuk dihitung dengan formula: jumlahHariKerja * 8

Mahasiswa

- Memiliki atribut SKS
- jamSibuk dihitung dengan formula: SKS * 3

Asdos

- Memiliki jamNgasdos
- jamSibuk dihitung dengan jamSibuk sebagai mahasiswa ditambah jamNgasdos

Ketentuan yang harus kamu penuhi untuk mendapat nilai sempurna dalam tutorial kali ini yaitu

- Menggunakan getJamSibuk() milik Mahasiswa untuk menghitung getJamSibuk() milik Asdos (hint: super).
- Memanfaatkan *constructor* superclass untuk membuat constructor subclass (hint: super).
- Jangan mengganti *access modifier* atribut yang telah ada pada template.

Selain ketentuan di atas, kamu bebas mengimplementasikan class-class tersebut asalkan output sesuai **tanpa mengubah** Simulator.java selain yang dikosongkan untuk kamu isi (TODO). Kamu bisa menyimpan/tidak jamSibuk, menggunakan/tidak toString() parent class, dan banyak hal lain yang kamu anggap cocok.

Output Simulator.java yang diharapkan adalah:

```
Fairuzikun adalah seorang asdos dengan jam sibuk 73
Raja OP Damanik adalah seorang dosen dengan jam sibuk 40
Angel-chan adalah seorang asdos dengan jam sibuk 64
```

Firman adalah seorang mahasiswa dengan jam sibuk 60
Nid to pass this sem adalah seorang mahasiswa dengan jam sibuk 69
Nivotko adalah seorang dosen dengan jam sibuk 24
Total jam sibuk elemen Fasilkom adalah 330

Komponen Penilaian :

Komponen	Penjelasan	Bobot
Implementasi Elemen	Dapat mengimplementasi class Elemen yang mengikuti sifat polymorphism sehingga Simulator.java dapat berjalan.	20 %
Implementasi Dosen	Dapat mengimplementasi class Dosen sehingga behaviour yang diinginkan tercapai.	15 %
Implementasi Mahasiswa	Dapat mengimplementasi class Mahasiswa sehingga behaviour yang diinginkan tercapai.	15%
Implementasi Asdos	Dapat mengimplementasi class Asdos sehingga behaviour yang diinginkan tercapai.	15%
Ketepatan Output	Ketepatan Output ketika Main dijalankan (Pemenuhan requirement client).	25%
Kerapian	Penulisan program mengikuti kaidah dan konvensi yang telah diajarkan. Program ditulis dengan rapi, terstruktur, dan disertakan oleh dokumentasi secukupnya.	10%

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_Lab7.zip

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%

Review

Setelah menyelesaikan tutorial kali ini, kalian diharapkan mampu menjawab pertanyaan-pertanyaan berikut. Jika ada yang tidak yakin, kamu dapat mencobanya sendiri pada kodemu namun **pastikan kode yang sudah selesai telah tersubmit** agar tidak telat dan percobaanmu tidak "mengotori" file yang harus dikumpulkan.

Pernyataan	ya/tidak
Untuk memanggil constructor superclass, dapat menggunakan super().	
Pemanggilan super() pada constructor harus dilakukan di awal, sebelum assignment atribut lain.	
Super hanya dapat digunakan sebagai constructor.	
Atribut private dapat diakses subclass.	
Subclass dapat mengakses method yang berada pada superclass, misal Mahasiswa memanggil getNama() padahal Mahasiswa tidak mendefinisikan getNama() namun Elemen mendefinisikan getNama().	
Semua object pasti memiliki toString(), walaupun tidak mendifinisikannya.	
Sebuah subclass dapat mengimplementasi method dengan nama yang telah ada pada superclass.	
Untuk meng-override method dari superclass, harus menggunakan annotation @Override.	
Object subclass dapat disimpan pada variabel dengan tipe subclass, misalnya object Mahasiswa pada variabel Elemen.	

ArrayList<Tipe> elemenFasilkom pada Simulator.java dapat diganti dengan List<Tipe> elemenFasilkom.	
Semua object dapat disimpan pada variabel dengan tipe Object, misal Object x = new A(); dengan A merupakan class apa pun.	
Object x = 3; akan menghasilkan compile error.	