Dasar-Dasar Pemrograman 2

Lab 06

Inheritance and Polymorphism

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 dan polymorphism.

Inheritance Dapat didefinisikan sebagai suatu proses dimana suatu 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.

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

class BangunDatar {
    int hasil;

    public void hitungLuas() {
```

```
    }

    public void hitungKeliling() {
    }
}

class Persegi extends BangunDatar {
    int x;

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

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

    @Override
    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;
    }

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

    @Override
    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 dari superclass melalui suatu subclass dengan memakai keyword **super**. Silahkan membuat kode seperti berikut, dan coba eksekusikan.

```
// 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 subclass
        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 dibawah ini:

```
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[] args) {
        Subclass s = new Subclass(24);
        s.get(age);
    }
}
```

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.

```
class Mammal {}
public class Dog extends Mammal {
    public static void main(String[] args) {
        Mammal m = new Mammal();
        Dog d = new Dog();
        System.out.println(m instanceof Mammal);
        System.out.println(d instanceof Dog);
        System.out.println(d instanceof Mammal);
    }
}
```

Perhatikan bahwa output dari program akan menjadi seperti berikut:

```
true
true
true
```

Polymorphism dapat didefinisikan sebagai suatu konsep dimana suatu object 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 object 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.

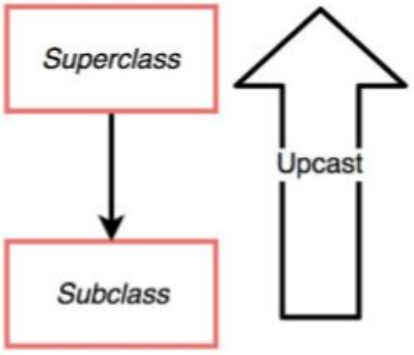
Hasil simulasi kedua kelas tersebut adalah:

```
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

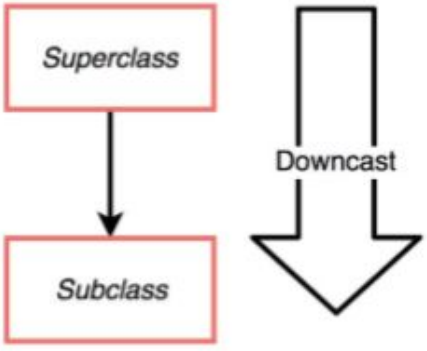


Upcasting adalah melakukan proses pengubahan referensi **subclass ke superclass.**

```
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 variabel 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 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.

Overloading

Method overloading adalah sebuah kemampuan yang membolehkan sebuah class mempunyai 2 atau lebih method dengan nama yang sama, yang membedakan adalah parameternya.

Pada method overloading perbedaan parameter mencakup:

1. Jumlah parameter

2. Tipe data dari parameter
3. Urutan dari tipe data parameter

Berikut ini contoh *Class* yang yang melakukan overloading:

```
public class PenggunaanOverloading {
    public static void main(String[] args) {
        ContohOverloading co  = new ContohOverloading();
        co.jumlah(83,32);
        co.jumlah(34,454,432);
        co.jumlah(34.43,34);
        co.jumlah(28,33.23);
    }
}

public class ContohOverloading {
    public void jumlah (int a, int b){
        System.out.println("Jumlah 2 angka ="+ (a + b));
    }
    //overloading perbedaan jumlah parameter
    public void jumlah (int a, int b, int c){
        System.out.println("Jumlah 3 angka =" + (a + b + c));
    }
    //overloading perbedaan tipe data parameter
    public void jumlah (double a, int b){
        System.out.println("Jumlah 2 angka (double+int) = "+ (a + b));
    }
    //overloading perbedaan urutan tipe data parameter
    public void jumlah (int b, double a){
        System.out.println("Jumlah 2 angka (int+double) = "+ (a + b));
    }
}
```

Overriding

Method overriding merupakan method yang *parent* yang ditulis kembali oleh subclass. Aturan dari method overriding pada Java :

1. Parameter yang terdapat pada method overriding di subclass harus sama dengan parameter yang terdapat pada *parent class*.
2. Aturan hak akses, hak akses method overriding di subclass tidak boleh lebih ketat dibandingkan hak akses method pada *parent class*.

Berikut ini contoh overriding:

```
public class Binatang {
    public void bergerak(){
        System.out.println("Binatang bergerak sesuai kemampuannya");
    }
    public void berkembangBiak(){
        System.out.println("Binatang berkembang biak sesuai kemampuannya");
    }
}

public class Mamalia extends Binatang {
    //overriding method parent class
```

```
    @Override
    public void bergerak(){
        System.out.println("Mamalia bergerak sebagian besar dengan kakinya");
    }
    public void berlari(){
        System.out.println("Sebagian Mamalia dapat berlari");
    }
}
public class PenggunaanOverriding {
    public static void main(String[] args) {
        Binatang b = new Binatang();
        Mamalia m = new Mamalia();
        Binatang bm = new Mamalia();

        b.bergerak();
        m.bergerak();
        bm.bergerak();
        bm.berkembangBiak();
    }
}
```

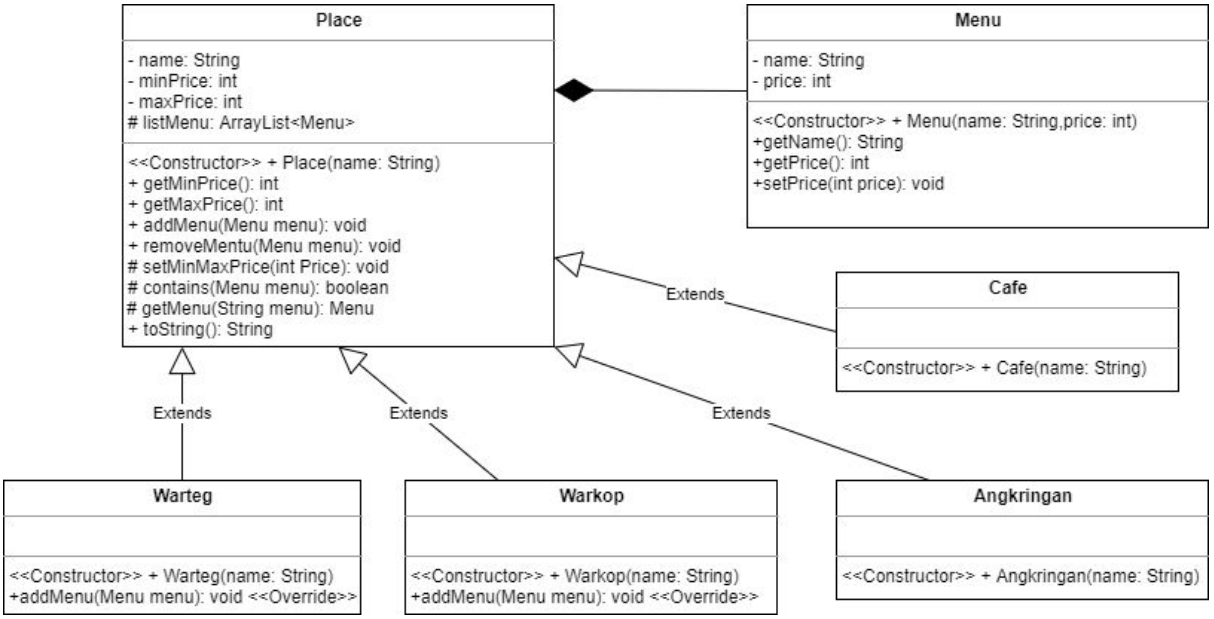
Soal Lab 06

NongSkuy

Babang Ryaas merupakan seorang mahasiswa Fakultas Ilmu Komputer Universitas Indonesia. Selayaknya mahasiswa pada umumnya, dia seringkali menghabiskan waktunya bersama teman-temanya untuk nongkrong di suatu tempat tertentu untuk melepas penat dari kegiatan perkuliahan. Akan tetapi, terkadang Babang Ryaas bersama teman-temanya kadang merasa kesulitan untuk memilih tempat nongkrong lantaran *budget* yang mereka miliki sebagai seorang mahasiswa/mahasiswi tidak menentu jumlahnya.

Berangkat dari permasalahan tersebut, Babang Ryaas berinisiatif untuk membantu teman-temanya menyelesaikan permasalahan tersebut dengan membuat sebuah program kecil-kecilan bernama NongSkuy. Program tersebut memiliki sebuah *database* berisikan tempat-tempat yang bisa dipilih sebagai tempat nongkrong. Program tersebut memiliki fitur-fitur seperti menambahkan/menghapus menu pada suatu tempat dan mencari pilihan tempat yang sesuai dengan *bugdet* yang dimiliki oleh Babang Ryaas beserta teman-temanya. Tugas kalian sekarang ialah membantu Babang Ryaas dalam membuat program tersebut. Babang Ryaas telah menyiapkan sebuah *database* untuk program tersebut agar kalian memastikan kebenaran hasil implementasi kalian pada program tersebut.

Berikut ialah UML class diagram yang digunakan untuk mengimplementasikan aplikasi tersebut:



Berikut beberapa catatan penting yang harus diperhatikan saat mengimplementasi program tersebut:

1) Pada Method **addMenu** di class Place, Anda menambahkan suatu menu ke dalam listMenu yang ada di class Place. Dalam suatu listMenu, **tidak boleh** ada nama menu yang sama. **Jika** menu yang ingin ditambahkan **sudah ada**, maka akan mencetak “Menu dengan nama [Nama Menu] sudah ada”. **Jika belum ada,** tambahkan menu ke listMenu dan cetak “Menu dengan nama [Nama Menu] sudah ditambahkan di [Nama Tempat]”
2) Terdapat perbedaan Method **addMenu** pada **Warteg** dan **Warkop** dengan **Place.** Perbedaannya yaitu addMenu pada **Warteg** dan **Warkop** dapat mengupdate menu yang sudah ada. Jika menu berhasil di update, maka akan mencetak “Menu dengan nama [Nama Menu] telah di update.”
3) Pada Method **removeMenu** di class Place, Anda menghapus suatu menu ke dalam listMenu yang ada di class Place. **Jika** menu yang ingin dihapus **tidak ada**, maka akan mencetak “Menu dengan nama [Nama Menu] tidak ada”. **Jika ada,** hapus menu dari listMenu dan cetak “Menu dengan nama [Nama Menu] sudah dihapus di [Nama Tempat]”
4) Perhatikan bahwa **harga minimum** dan **harga maksimum** dari pada **Place** menyimpan harga minimum dan harga maksimum dari menu yang ada di suatu tempat. **Keduanya** bisa berubah-ubah sesuai dengan menu yang ditambahkan maupun dihapus.

5) Method **searchPlace(int maxPrice)** dimaksudkan untuk mencari tempat yang **memiliki menu dengan harga kurang dari atau sama dengan** parameter maxPrice
6) Method **searchPlace(int minPrice, int maxPrice)** dimaksudkan untuk mencari tempat yang **semua menunya** yang berharga minPrice <= harga <= maxPrice
7) **Class Place** bisa dicetak dengan format "[Nama Tempat] dengan jangkauan harga Rp [Harga Minimum] - Rp [Harga Maksimum]"
8) Main class yang berisi simulator program tersebut terdapat pada NongSkuy.java
9) Diwajibkan untuk menggunakan konsep Inheritance dalam pembuatan implementasi program ini. Selain itu, anda juga harus bersikap teliti terhadap seluruh baris pada output dari program ini

Ouput yang Diharapkan: bisa dilihat pada file outputNongSkuy.txt pada template