



Polimorfisme

KSP JAVA 2023

Pendahuluan

Setelah sebelumnya kita membahas terkait konsep PBO yaitu abstraksi, enkapsulasi dan pewarisan, sekarang akan masuk pada bagian konsep yang keempat yaitu polimorfisme.

Polimorfisme berasal dari kata poly yang berarti **banyak**, dan morfos yang berarti **bentuk**. Dalam PBO polimorfisme dapat diartikan sebagai “**banyak bentuk / banyak rupa**” dimana nantinya sebuah objek dapat memberikan respon dengan caranya masing masing yang berasal dari sumber yang sama. “Banyak bentuk” disini dapat diartikan sebagai meskipun method nya sama namun aksi yang dilakukan berbeda atau dalam kata lain Polimorfisme memungkinkan sebuah objek untuk mengeksekusi perintah yang sama dengan cara yang berbeda-beda, tergantung pada tipe objek tersebut.

Polimorfisme sendiri dibagi menjadi 2 jenis

1. Polimorfisme Statik
2. Polimorfisme Dinamis

Polimorfisme Statik

Polimorfisme ini akan menggunakan konsep overloading dalam penggunaanya, terdapat 2 jenis overloading yaitu overloading constructor dan overloading methods. Polimorfisme statik berfokus pada 1 kelas saja, yang nantinya akan ada beberapa nama method / konstruktor yang sama, namun jumlah atau tipe parameter yang berbeda. Untuk lebih jelas teman teman dapat melihat pada penjelasan dibawah.

1. Overloading constructor
Memungkinkan untuk membuat 2 konstruktor atau lebih dengan nama yang sama di dalam satu kelas, namun memiliki jumlah atau tipe parameter yang berbeda.

```

10
11 public class Person {
12     private String name;
13     private int age;
14
15     public Person(){
16         this.name = "John Doe";
17         this.age = 20;
18     }
19
20     public Person(int age) {
21         this.age = age;
22     }
23
24     public Person(String name, int age) {
25         this.name = name;
26         this.age = age;
27     }
28
29     public void tampilData(){
30         System.out.println("\nNama : " + this.name);
31         System.out.println("\nUmur : " + this.age);
32     }
33
34 }
35
36

```

Dapat dilihat pada gambar diatas kita membuat kelas “Person” yang memiliki 3 buah konstruktor, **yang pertama** adalah konstruktor default dimana saat pembuatan objek kita tidak perlu memberikan nilai atau value pada parameter, sehingga nantinya objek tersebut akan memiliki nilai default name = “john doe” dan age = 20. **Yang kedua** adalah konstruktor dimana ketika ada objek dibuat menggunakan konstruktor tersebut, maka kita hanya akan memberikan nilai umur pada parameternya. **Yang ketiga** adalah konstruktor dimana kita harus memberikan value name dan age pada saat pembuatan objek menggunakan konstruktor tersebut. Berikut adalah contoh pemanggilan ketiga konstruktor tersebut pada main class.

```

13
14 public static void main(String[] args) {
15
16     Person person1 = new Person();
17     Person person2 = new Person(12);
18     Person person3 = new Person("Dandy", 10);
19
20     person1.tampilData();
21
22     System.out.println("=====");
23
24     person2.tampilData();
25
26     System.out.println("=====");
27
28     person3.tampilData();
29
30 }
31
32 }
33

```

```

Nama : John Doe
Umur : 20
=====
Nama : null
Umur : 12
=====
Nama : Dandy
Umur : 10
BUILD SUCCESSFUL (total time: 0 seconds)

```

2. Overloading method

Memungkinkan untuk membuat 2 method atau lebih dengan nama yang sama di dalam satu kelas, namun memiliki jumlah atau tipe parameter yang berbeda.

```
11 public class Calculator {
12
13     public int add(int num1, int num2){
14         return num1 + num2;
15     }
16
17     public int add(int num1, int num2, int num3){
18         return num1 + num2 + num3;
19     }
20
21 }
22
```

Untuk penerapan dari overloading method sama saja seperti overloading konstruktor, dimana dapat dilihat pada gambar diatas jika sebuah objek melakukan pemanggilan method yang diberi **2 nilai parameter** berupa integer, maka objek tersebut akan mengeksekusi method "add" yang pertama, sedangkan jika objek melakukan pemanggilan method yang diberi **3 nilai parameter** berupa integer, maka objek tersebut akan mengeksekusi method "add" yang kedua. Berikut adalah contoh pemanggilan kedua method tersebut pada main class.

```
32 Calculator calculator1 = new Calculator();
33 Calculator calculator2 = new Calculator();
34
35
36 System.out.println("Total method pertama : " + calculator1.add(5, 8));
37 System.out.println("Total method kedua : " + calculator2.add(5, 8, 10));
38
39
```

```
Total method pertama : 13
Total method kedua : 23
BUILD SUCCESSFUL (total time: 0 seconds)
```

Polimorfisme Dinamis

Polimorfisme ini akan menggunakan konsep overriding dalam penggunaannya, konsep ini memungkinkan sebuah kelas turunan untuk menggunakan atau menimpa method yang telah didefinisikan pada kelas induknya. Implementasi konsep overriding dapat dicapai dengan 2 pilihan

1. Menggunakan konsep kelas konkret

Polimorfisme kelas konkret adalah ketika sebuah kelas turunan menggunakan method yang sama pada kelas induk namun merespon method tersebut dengan cara berbeda, dalam arti lain body atau isi method tersebut berbeda. Untuk lebih jelasnya perhatikan gambar di bawah.

```
12 public class Karyawan {
13     protected String nama;
14     protected double gaji;
15
16     public Karyawan(String nama, double gaji) {
17         this.nama = nama;
18         this.gaji = gaji;
19     }
20
21     //Method yang akan dioverride
22     public void showData(){
23         System.out.println("Nama : " + this.nama);
24         System.out.println("gaji : " + this.gaji);
25     }
26 }
```

Pertama kita akan membuat kelas konkret bernama Karyawan yang mempunyai atribut serta method seperti gambar di atas.

```
11 public class Staf extends Karyawan {
12     private double jamLembur;
13
14     public Staf(double jamLembur, String nama, double gaji) {
15         super(nama, gaji);
16         this.jamLembur = jamLembur;
17     }
18
19     public void showData(){
20
21         // Melakukan overriding method dari kelas induk
22         super.showData();
23         System.out.println("Jam Lembur : " + this.jamLembur);
24     }
25 }
26
27
28 public class Manager extends Karyawan {
29     private double tunjanganKesehatan;
30
31     public Manager(double tunjanganKesehatan, String nama, double gaji) {
32         super(nama, gaji);
33         this.tunjanganKesehatan = tunjanganKesehatan;
34     }
35
36     public void showData(){
37
38         // Melakukan overriding method dari kelas induk
39         super.showData();
40         System.out.println("Tunjangan Kesehatan : " + this.tunjanganKesehatan);
41     }
42 }
43 }
```

Dapat dilihat pada gambar di atas, kita membuat sebuah dua kelas baru yaitu Staf dan Manager dan kedua kelas tersebut meng “extends” kelas Karyawan. Pada method showData kedua kelas tersebut melakukan override pada kelas induknya, masing masing kelas memanggil nama method yang sama dari kelas induknya, namun respon dari masing masing kelas berbeda, itulah konsep dari polimorfisme dinamis pada kelas konkret. Berikut contoh penerapannya pada main class.

Tipe data yang digunakan adalah Karyawan

```
Karyawan karyawan1 = new Staf(500000 , "Vincent", 800000);
karyawan1.showData();

System.out.println("-----");

karyawan1 = new Manager(1000000 , "Nessa", 900000);
karyawan1.showData();
```

```

run:
Nama : Vincent
gaji : 8000000.0
Jam Lembur : 500000.0
-----
Nama : Nessa
gaji : 9000000.0
Tunjangan Kesehatan : 1000000.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

Dapat dilihat pada gambar di atas, kelas staf dan kelas manager mempunyai kelas induk yang sama yaitu Karyawan, maka dari itu kita dapat menggunakan tipe data Karyawan saat pembentukan objek untuk kelas staf dan manager dengan nama karyawan1. Penggunaan konsep polimorfisme pada hal ini adalah variabel karyawan1 dapat merespon 2 output yang berbeda, yang pertama adalah untuk staf karena karyawan1 diassign oleh kelas Staf terlebih dahulu, selanjutnya diassign oleh kelas Manager, maka memberikan respon yang berbeda.

2. Menggunakan konsep kelas abstrak

Dari namanya sendiri dapat terlihat bahwa kelas abstrak adalah kelas yang masih bersifat “acak” sehingga method di dalam kelas abstrak tersebut juga masih bersifat abstrak atau acak, sehingga tujuan dari kelas abstrak adalah untuk mendefinisikan sebuah struktur sebagai dasar dari kelas kelas lain yang diturunkan dari kelas abstrak tersebut. Sehingga kelas abstrak tidak dapat “diinstansiasi” atau dalam kata lain sebuah objek tidak dapat terbentuk dari sebuah kelas abstrak.

Seperti yang sudah disinggung diatas bahwa kelas abstrak dapat memiliki method biasa dan method abstrak. Method abstrak sendiri merupakan method yang tidak memiliki implementasi di dalamnya, dimana hanya memiliki nama dan parameter saja. Method abstrak harus diimplementasikan oleh kelas turunan yang mewarisi dari kelas abstrak.

Berikut contoh dari implementasi overriding menggunakan konsep kelas abstrak

Langkah 1

```

10 //Pendeklarian kelas abstrak
11 public abstract class BangunDatar {
12
13     //Method 1 yang akan di override
14     public abstract double hitungLuas();
15
16     //Method 2 yang akan di override
17     public abstract double hitungKeliling();
18 }
19

```

Langkah pertama adalah melakukan pembentukan kelas induk yang bersifat abstrak dan memiliki method abstrak yang nantinya akan dioverride oleh kelas turunannya. Dapat dilihat

pada gambar di atas kita membuat kelas dengan nama "BangunDatar" dimana kelas tersebut bersifat abstrak yang memiliki 2 buah method abstrak yaitu hitungLuas dan hitungKeliling, hal ini dapat terjadi karena setiap bangun datar pasti memiliki rumus hitungLuas dan hitungKeliling masing – masing jadi kita tetapkan kedua method tersebut menjadi method abstrak pada kelas "BangunDatar".

Langkah 2

Langkah kedua adalah melakukan pembuatan kelas turunan yang mempunyai kelas induk "BangunDatar", semisal kita membuat kelas turunan dengan nama Segitiga.

```
8 | Segitiga is not abstract and does not override abstract method hitungKeliling() in bangunDatar
9 | ----
10 | (Alt-Enter shows hints)
11 | public class Segitiga extends bangunDatar {
12 |
13 |
14 | }
```

Syarat dari kelas turunan ketika mempunyai kelas induk dengan sifat abstrak yaitu kelas turunan tersebut harus mengimplementasi / mengoverride method abstrak yang ada pada kelas induk, jika tidak maka IDE akan mengeluarkan error seperti pada gambar diatas.

```
10 |
11 | public class Segitiga extends BangunDatar {
12 |     private double alas;
13 |     private double tinggi;
14 |
15 |     public Segitiga(double alas, double tinggi) {
16 |         this.alas = alas;
17 |         this.tinggi = tinggi;
18 |     }
19 |
20 |     @Override
21 |     public double hitungLuas() {
22 |         return 0.5 * alas * tinggi;
23 |     }
24 |
25 |     @Override
26 |     public double hitungKeliling() {
27 |         return alas * 3;
28 |     }
29 |
30 | }
```

Selanjutnya adalah melakukan implementasi / override method dari kelas induk seperti pada gambar diatas, dikarenakan kelas turunanya adalah Segitiga maka di dalam kedua method tersebut kita menuliskan perhitungan sesuai dengan rumus Segitiga dan menambahkan atribut sesuai yang dibutuhkan oleh bangun datar Segitiga. Berikut contoh penerapannya pada main class.

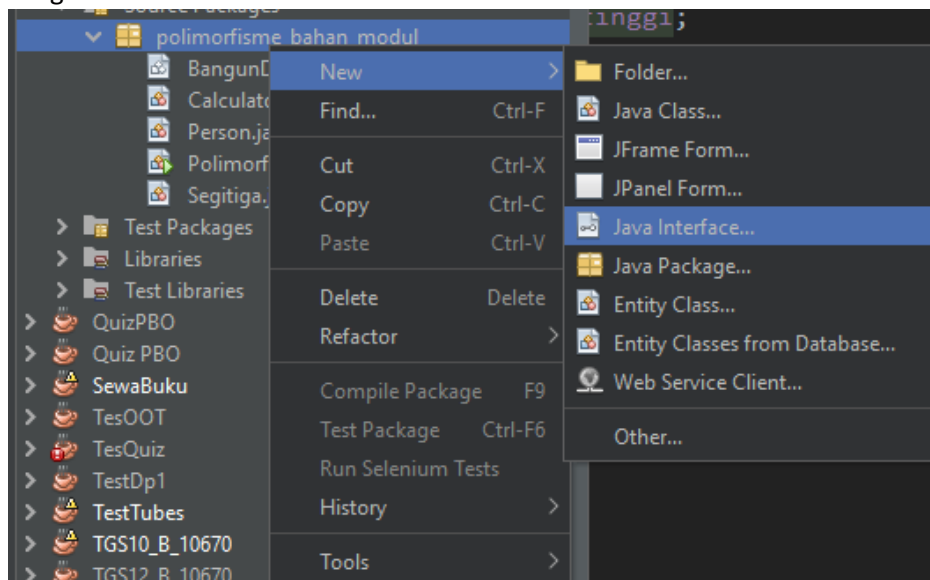
```
41 | Segitiga segitiga1 = new Segitiga(5,5);
42 | System.out.println("Luas : " + segitiga1.hitungLuas());
43 | System.out.println("Keliling : " + segitiga1.hitungKeliling());
```

```
Luas : 12.5
Keliling : 15.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Menggunakan Interface

Interface digunakan untuk mendefinisikan sebuah kerangka kerja atau kontrak untuk implementasi sebuah kelas. Konsep dari interface sendiri mirip dengan kelas abstrak, nantinya di dalam sebuah interface akan ada method abstrak yang harus diimplementasi oleh kelas yang meng implementasi kelas interface tersebut. Berikut langkah – langkah untuk membentuk kelas interface

Langkah 1



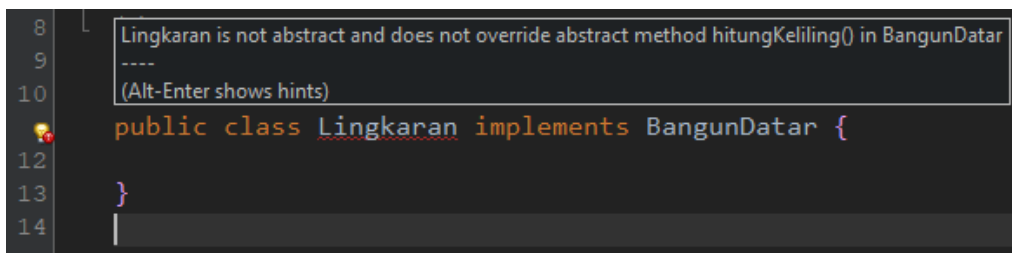
Ketika membuat file klik pada java interface, lalu berinama interface tersebut.

Langkah 2

```
11 public interface BangunDatar {
12
13     //Konstanta
14     double PHI = 3.14;
15
16     //Method abstract
17     double hitungLuas();
18     double hitungKeliling();
19
20 }
21
```


Dapat dilihat pada gambar diatas kita membentuk sebuah interface bernama BangunDatar. Interface hanya bisa diisi oleh variabel static dan final serta method yang bersifat abstrak, sehingga pada gambar di atas jika kita menulis variabel seperti pada gambar maka by default variabel tersebut akan bersifat public, static dan final, sama juga jika kita menulis method seperti pada gambar di atas maka by default akan bersifat public dan abstract.

Langkah 3



```
8 |
9 |
10 | Lingkaran is not abstract and does not override abstract method hitungKeliling() in BangunDatar
11 | ----
12 | (Alt-Enter shows hints)
13 | public class Lingkaran implements BangunDatar {
14 |
15 | }
```

Sama seperti kelas abstract, jika ada kelas yang mengimplementasi sebuah Interface, maka kelas tersebut harus mengimplementasi semua method yang ditulis pada Interface.



```
10 |
11 | public class Lingkaran implements BangunDatar {
12 |
13 |     private int jari_jari;
14 |
15 |     public Lingkaran(int jari_jari) {
16 |         this.jari_jari = jari_jari;
17 |     }
18 |
19 |
20 |     @Override
21 |     public double hitungLuas() {
22 |         return PHI * this.jari_jari * this.jari_jari;
23 |     }
24 |
25 |     @Override
26 |     public double hitungKeliling() {
27 |         return 2 * PHI * this.jari_jari;
28 |     }
29 |
30 |
31 | }
32 |
```

Dapat dilihat pada gambar diatas bahwa kelas lingkaran mengakses variabel berupa PHI yang dimiliki oleh Interface BangunDatar dan juga mengimplementasi method dari Interface tersebut. Berikut contoh penerapannya pada main class.

```

17     Lingkaran lingkaran1 = new Lingkaran(14);
18     System.out.println("Luas : " + lingkaran1.hitungLuas());
19     System.out.println("Keliling : " + lingkaran1.hitungKeliling());

```

```

Luas : 615.44
Keliling : 87.92
BUILD SUCCESSFUL (total time: 0 seconds)

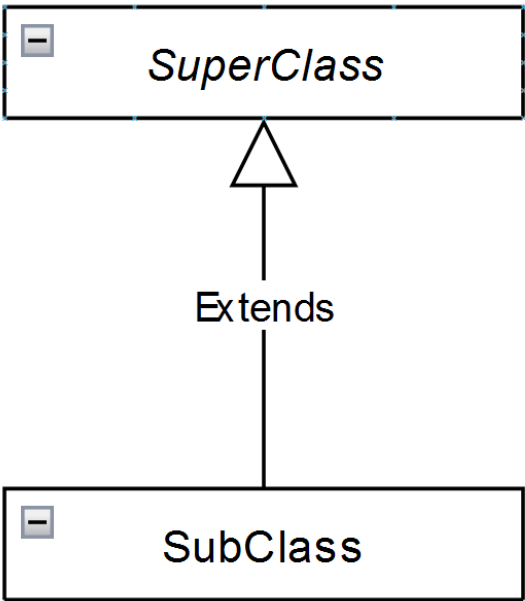
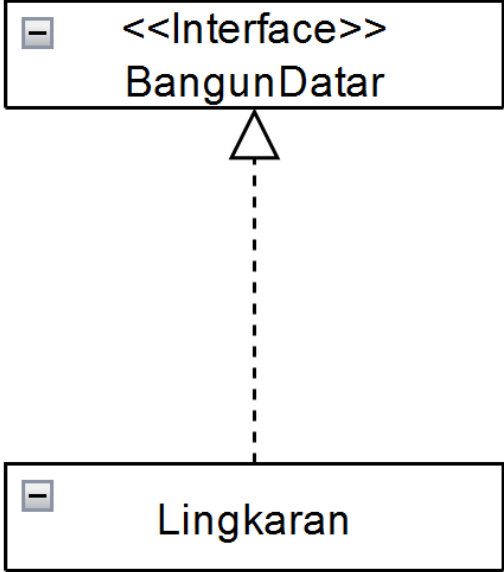
```

QNA Section

Pada bagian ini akan ada beberapa tambahan beserta kesimpulan tentang modul polimorfisme ini.

Q. Apa perbedaan antara kelas abstrak dan Interface?

A. Untuk menjawab pertanyaan tersebut mari kita buat dalam bentuk tabel supaya teman-teman dapat memahami lebih lagi terkait kedua hal tersebut.

Penggunaan simbol pada diagram kelas	
Kelas Abstrak	Interface
 <p>Diagram kelas abstrak sama seperti diagram kelas biasa yaitu panah tertutup, namun yang menandakan bahwa itu adalah kelas abstrak adalah nama dari kelas abstrak tersebut miring</p>	 <p>Diagram Interface memiliki garis putus - putus</p>
Apa saja yang dapat dilakukan oleh Kelas Abstrak maupun Interface?	
Kelas Abstrak	Interface
<ul style="list-style-type: none"> - Kelas abstrak tidak dapat memiliki sebuah objek, hanya kelas turunan dari kelas abstrak tersebut yang dapat memiliki objek 	<ul style="list-style-type: none"> - Pada dasarnya Interface bukanlah kelas, melainkan hanya kumpulan dari method, maka tidak dapat memiliki sebuah objek.

<pre> 9 10 //Pendeklarasian kelas abstrak 11 public abstract class BangunDatar { 12 13 //atribut 14 private int panjang; 15 private int lebar; 16 17 //Method ini tidak wajib di override 18 public void showValue(){ 19 System.out.println("Panjang : " + this.panjang); 20 System.out.println("Lebar : " + this.lebar); 21 } 22 23 //Method 1 yang akan di override 24 public abstract double hitungLuas(); 25 26 //Method 2 yang akan di override 27 public abstract double hitungKeliling(); 28 } 29 </pre>	<pre> public interface BangunDatar { //Konstanta double PHI = 3.14; //Method abstract double hitungLuas(); double hitungKeliling(); } </pre> <p>Default dari atribut pada Interface adalah public, final dan static Default dari method pada Interface adalah abstrak</p>
<p>Kelas abstrak dapat diisi dengan atribut serta method yang tidak perlu dioverride, sehingga perbedaan dengan kelas biasa hanyalah</p> <ol style="list-style-type: none"> 1. Kelas abstrak tidak dapat memiliki objek 2. Dapat membentuk method abstrak <p>Terlepas dari itu sifat dari kelas abstrak sama seperti kelas biasa begitu juga dengan sifat pewarisanya ke kelas turunan.</p>	<pre> public class Segitiga extends BangunDatar { private double alas; } </pre> <p>Kelas turunan hanya dapat mempunyai 1 kelas abstrak saja, sama seperti kelas induk pada umumnya.</p>
	<pre> public class SubClassInterface implements Interface1, Interface2 { } </pre> <p>Kelas turunan dapat mengimplemen lebih dari 1 Interface. (Multiple inheritance)</p>

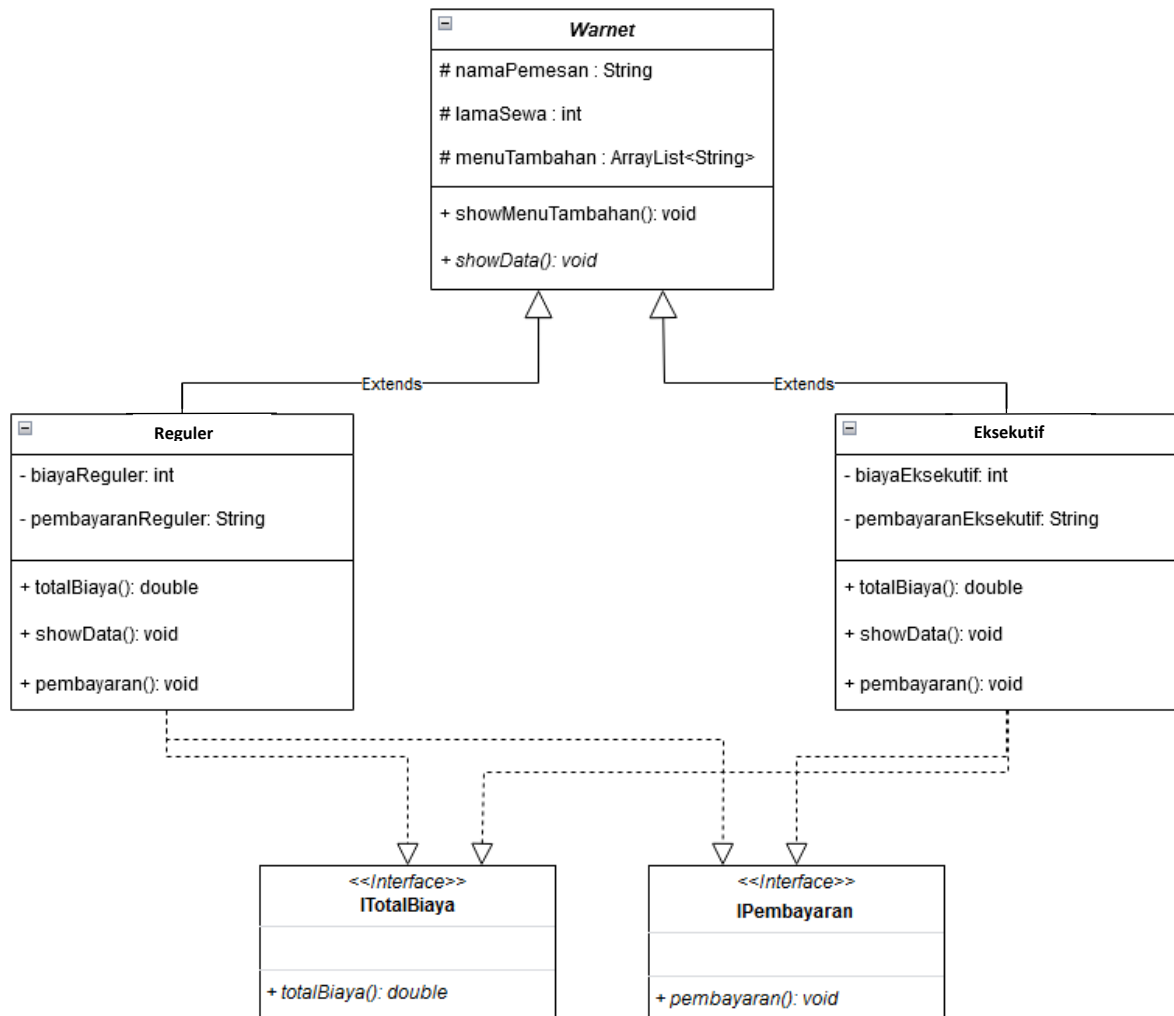
Q. Apakah kelas turunan dapat mempunyai kelas induk abstrak sekaligus mengimplemen sebuah Interface?

A. Bisa, berikut contoh kasus dimana kelas turunan akan menggunakan kedua hal tersebut.

Guided / Challenge

Warnet Tono

Anda diminta oleh Tono untuk mengelola sebuah warnet pada bagian pendataan pengguna dari warnet tersebut, sehingga anda diminta untuk membuatkan program tersebut, bentuk dari diagram mengenai program yang akan dibentuk adalah sebagai berikut



Beberapa ketentuan yang harus diperhatikan

- Buatlah 1 kelas abstrak, 2 kelas konkret dan 2 interface seperti pada diagram di atas, dan pastikan relasi antar kelas/interface sama seperti diagram di atas.
- Pada kelas abstrak warnet akan ada overloading konstruktor, yang pertama user tidak memesan menu tambahan, yang kedua user memesan menu tambahan tersebut.
- Method `totalBiaya` akan dihitung dari (`lamaSewa * biayaRegular + biaya masing masing paket`).
 - o Untuk paket Regular
 - `biayaRegular = 8000`
 - `biayaTambahan = 5000`
 - o Untuk paket Eksekutif

- biayaReguler = 10000
- biayaTambahan = 8000
- Method showData akan menampilkan nama pemesan, total biaya dan pembayaran
- Method pembayaran hanyalah void yang berisi string + pembayaranReguler/pembayaranEksekutif
- Berikut adalah contoh output jika program dirun

```
run:
---- User Reguler ----
Nama Pemesan : John
Total Biaya : 29000.0
Pembayaran untuk warnet paket reguler adalah menggunakan Cash
Menu Tambahan : Indomie, Es Teh,

---- User Eksekutif ----
Nama Pemesan : Doe
Total Biaya : 38000.0
Pembayaran untuk warnet paket eksekutif adalah menggunakan Debit
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bagi teman-teman yang ingin mengerjakan selamat mengerjakan!, bagi yang tidak atau bingung dapat langsung menonton video penjelasan modul ini ya, pembahasan soal ada pada bagian akhir video. Untuk kode program dapat didownload pada tautan di bawah, Terima kasih!.

Source code: https://github.com/Dandy-Candra/GDPolimorfisme_KSP_JAVA