

VERSION 1.3

FEBRUARI, 2023



# PEMROGRAMAN BERORIENTASI OBJEK

MODUL 4 - ABSTRACTION & INTERFACE

DISUSUN OLEH:

Muhammad Nizar Zulmi Rohmatulloh

Jody Yuantoro

DIAUDIT OLEH:

Aminudin, S.Kom., M.Cs.

PRESENTED BY: TIM LAB. IT

UNIVERSITAS MUHAMMADIYAH MALANG

## PEMROGRAMAN BERORIENTASI OBJEK

---

### TUJUAN

1. Mahasiswa dapat memahami konsep abstraction.
2. Mahasiswa dapat memahami abstract class & abstract method.
3. Mahasiswa dapat memahami interface.

---

### TARGET MODUL

1. Mahasiswa dapat membuat dan mengimplementasikan abstract class.
2. Mahasiswa dapat membuat dan mengimplementasikan interface.
3. Mahasiswa dapat membuat dan mengimplementasikan relasi antar kelas.

---

### PERSIAPAN

1. Java Development Kit.
2. Text Editor / IDE (Visual Studio Code, Netbeans, IntelliJ IDEA, atau yang lainnya).

---

### KEYWORDS

- |                   |                   |
|-------------------|-------------------|
| ● Abstract Class  | ● Concrete Method |
| ● Abstract Method | ● Interface       |
| ● Concrete Class  | ● Class Relation  |

## TEORI

- **Abstraction**

*Abstract* adalah suatu ide yang bukan objek materi (tidak memiliki bentuk fisik), lawan kata *abstract* adalah *concrete*. Dalam konteks PBO, *abstraction* adalah konsep yang memungkinkan kita untuk menyembunyikan kompleksitas dari sebuah objek dan hanya menampilkan fungsionalitas yang penting dalam sebuah interaksi.

- **Abstract Class**

*Abstract class* adalah kelas yang dideklarasikan dengan *keyword* *abstract*, dan sifatnya tidak dapat dibuat menjadi suatu objek (tidak dapat diinstansiasi). *Abstract class* biasanya digunakan untuk generalisasi objek yang sangat umum, misalnya Hewan, Kendaraan, *Database*, dan lain sebagainya.

Abstract class berguna untuk diturunkan / diwarisi kelas lainnya menggunakan *keyword* *extends*, penggunaan utamanya adalah untuk polimorfisme (*Polymorphism*) yang akan dipelajari di modul selanjutnya. Dalam suatu abstract class dapat berisi atribut (variabel), *abstract method* maupun *concrete method*.

Pembuatan abstract class:

```
public abstract class NamaKelas {  
  
    // Dapat berisi atribut / variabel.  
    // Dapat berisi method.  
    // Constructor untuk kelas yang mewarisi.  
  
}
```

- **Abstract Method**

*Abstract Method* adalah method yang bersifat abstrak yang ditandai dengan penambahan *keyword* *abstract* pada deklarasinya serta tidak memiliki implementasi

*body* / isi. Deklarasi *abstract method* langsung diakhiri tanda titik koma (;), tanpa tanda kurung kurawal ({ ... }). Semua *abstract method* yang ada akan dipaksakan untuk ada di kelas turunannya (harus di-*override*).

Pembuatan *abstract method*:

```
[modifier] abstract [returnType] namaMethod([optionalReturnType] [optionalParameter]);
```

Dalam bentuk utuh:

```
public abstract class NamaKelas {
    // Abstract Method
    public abstract int namaMethod1(int param1, float param2);

    // Concrete Method
    public int namaMethod2(int param1, int param2) {
        return param1 + param2;
    }
}
```

### → Contoh Implementasi Abstraction

Pada dasarnya setiap objek yang berbeda sering memiliki kesamaan atau kemiripan tertentu. Kita ambil contoh kelas hewan. Misalnya, kucing dan anjing memiliki kemiripan sebagai hewan. Tetapi hewan adalah hal yang sangat umum. Hewan sendirinya bukanlah suatu objek. Tentu saat kita menyebutkan 'hewan bersuara' kita tidak tau suara hewan tersebut apa karena ia tidak merujuk kepada hal yang lebih spesifik. Oleh karena itu, hewan dapat diubah menjadi kelas abstrak.

Pertama-tama dalam proyek anda buatlah kelas Main pada file Main.java yang berisi main method:

```

1  package edu.praktikum.pbo;
2
3  public class Main {
4      Run | Debug
5      public static void main(String[] args) {
6      }
7  }

```

Kemudian buatlah kelas Hewan pada file Hewan.java

```

1  package edu.praktikum.pbo;
2
3  public abstract class Hewan {
4
5      public abstract void bersuara();
6
7  }

```

Untuk membuktikan bahwa kelas Hewan tidak dapat dijadikan objek akan kita coba instansiasi kelas Hewan di kelas Main.

```

1  package edu.praktikum.pbo;
2
3  public class Main {
4      Run | Debug
5      public static void main(String[] args) {
6          Hewan hewan = new Hewan();
7      }

```

⊗ Main.java 1 of 1 problem

Cannot instantiate the type Hewan Java(16777373)

```

6  }
7  }

```

Terbukti, akan muncul error “*Cannot instantiate the type Hewan*” dikarenakan kelas Hewan adalah abstract class.

Kita akan membuat kelas lain, Kucing pada file Kucing.java dan Anjing pada file Anjing.java yang masing-masing akan meng-*extends* kelas Hewan.

Perhatikan apa yang terjadi saat awal kita meng-*extends* class Hewan, ada merah-merah error guys:

```
1 package edu.praktikum.pbo;
2
3 public class Kucing extends Hewan {
4 }
```

Dengan error “The type Kucing must implement the inherited abstract method Hewan.bersuara()”.

edu.praktikum.pbo.Kucing

The type Kucing must implement the inherited abstract method  
Hewan.bersuara() Java(67109264)

[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

Apa yang terjadi? kelas Kucing harus mengimplementasikan abstract method yang ada pada abstract class Hewan yaitu method bersuara. Mari kita coba perbaiki, kalian bisa manual mengetik atau secara otomatis di VS Code kalian bisa meng-klik Quick Fix (Ctrl+.) seperti di atas atau di IDE lain cari 💡 kemudian klik ***add unimplemented method***. IDE akan otomatis megenerate.

Perhatikan anotasi @Override sebelum method bersuara, hal ini diperlukan karena class Kucing merupakan turunan dari class Hewan yang juga memiliki method bersuara meskipun method tersebut bersifat abstract.

```
1  package edu.praktikum.pbo;
2
3  public class Kucing extends Hewan {
4
5      @Override
6      public void bersuara() {
7          // TODO Auto-generated method stub
8      }
9
10 }
11 }
```

Akan kita isi method bersuara di atas dengan suara kucing, ya “meow”.

```
1  package edu.praktikum.pbo;
2
3  public class Kucing extends Hewan {
4
5      @Override
6      public void bersuara() {
7          System.out.println(x: "Meow");
8      }
9
10 }
```

Lakukan yang sama pada kelas Anjing dengan suara anjing.

```

1  package edu.praktikum.pbo;
2
3  public class Anjing extends Hewan {
4
5      @Override
6      public void bersuara() {
7          System.out.println(x: "Woof");
8      }
9
10 }
```

Kemudian kita bisa membuat objek dari kelas Kucing dan Anjing.

```

1  package edu.praktikum.pbo;
2
3  public class Main {
4      Run | Debug
5      public static void main(String[] args) {
6
7          Kucing kucing = new Kucing();
8          Anjing anjing = new Anjing();
9
10         kucing.bersuara();
11         anjing.bersuara();
12     }
13 }
```

Coba jalankan kode tersebut apa yang terjadi? Kalau benar, akan muncul “Meow” diikuti dengan “Woof” sesuai urutan pemanggilan method pada objek.

Apa yang dapat kita simpulkan? Kucing dan anjing sama-sama seekor hewan dan keduanya dapat bersuara, tetapi tentu saja suara dari keduanya berbeda. T...T...Tapi.. kita bisa kan membuat kedua objek tersebut tanpa membuat abstract class? Tentu saja kita



bisa, kita akan melihat lebih jauh di modul selanjutnya tentang polimorfisme (*Polymorphism*).

- **Interface**

Masih berkaitan dengan abstraction. Interface adalah satu cara untuk mencapai abstraction secara menyeluruh (*total abstraction*). Namun, interface bukan class, meskipun terlihat sama. Interface digunakan untuk mendefinisikan suatu sifat-sifat (*behaviours*, berupa method) suatu class.

Persamaannya dengan abstract class adalah keduanya tidak dapat diinstansiasi menjadi objek, lantas apa perbedaannya?

Abstract Class	Interface
Bisa memiliki abstract & concrete method.	Hanya abstract method.
Bisa memiliki method static dan final.	Method tidak boleh bersifat static dan final.
Access modifier perlu ditulis sendiri.	<b>Secara implisit</b> semua method adalah public abstract.
Bisa memiliki <i>constants</i> dan <i>instance variables</i> .	Hanya dapat memiliki <i>constants</i> karena secara implisit semua variable dalam interface adalah public static final.
Hanya dapat meng- <b>extends</b> <u>satu</u> abstract class lainnya (tidak bisa multiple inheritance).	Dapat meng- <b>extends</b> lebih dari 1 interface.
Dapat meng- <b>implements</b> lebih dari satu interface.	Tidak dapat meng- <b>implements</b> interface lain.

Umumnya, *abstract class* bisa menjawab pertanyaan “Suatu objek apakah itu?”. Sedangkan, *interface* bisa menjawab pertanyaan “Apa yang objek itu bisa lakukan?”.

Misalnya, burung adalah hewan dan burung bisa terbang, pesawat juga bisa terbang, lantas apakah pesawat adalah hewan? tentu bukan, keduanya memiliki sifat / *behaviour* sama-sama bisa terbang, kita dapat membuat interface `Flyable` untuk kedua objek tersebut. Penggunaan *interface* tidak selalu seperti di atas, selebihnya tentunya tergantung *use-case*.

Contoh pendeklarasian *interface*:

```
interface Flyable {

    // Dapat berisi variabel

    // sudah defaultnya public abstract
    void fly();

}
```

### → Contoh Implementasi Interface

Kode di bawah ini menggunakan abstract class **Hewan** pada bagian implementasi abstract class.

Flyable.java

```
3  public interface Flyable {
4
5      String ABILITY = "bisa terbang";
6
7      void fly();
8
9  }
```

## Burung.java

```
3  ✓ public class Burung extends Hewan implements Flyable {  
4  
5      @Override  
6  ✓  public void bersuara() {  
7      |      System.out.println(x:"tweet tweet");  
8      |  }  
9  
10     @Override  
11  ✓  public void fly() {  
12      |      System.out.println("chirp chirp " + ABILITY);  
13      |  }  
14  
15  }
```

## Pesawat.java

```
3  public class Pesawat implements Flyable {  
4  
5      @Override  
6      public void fly() {  
7          |      System.out.println("Swhoooshhh " + ABILITY);  
8          |  }  
9  
10 }
```

App.java

```

3  public class App {
4
5      Run | Debug
6      public static void main(String[] args) {
7
8          Burung burung = new Burung();
9          Pesawat pesawat = new Pesawat();
10         burung.bersuara(); // Output: tweet tweet
11         burung.fly(); // Output: chirp chirp bisa terbang
12         pesawat.fly(); // Output: Swhoooshhh bisa terbang
13     }
14
15 }

```

- **Is-a relation & Operator instanceof**

Di Java kita bisa melihat suatu objek adalah suatu *instance* dari *class* apa dengan operator `instanceof` yang akan mengembalikan nilai boolean. Menggunakan potongan kode sebelumnya cobalah berikut:

```

1  package edu.praktikum.pbo;
2
3  public class Main {
4      Run | Debug
5      public static void main(String[] args) {
6
7          Kucing kucing = new Kucing();
8
9          System.out.println("Apakah Kucing IS-A Object = " + (kucing instanceof Object));
10         System.out.println("Apakah Kucing IS-A Hewan = " + (kucing instanceof Hewan));
11         System.out.println("Apakah Kucing IS-A Kucing = " + (kucing instanceof Kucing));
12     }
13 }

```

Output dari kode di atas adalah berikut:

```
Apakah Kucing IS-A Object = true
Apakah Kucing IS-A Hewan = true
Apakah Kucing IS-A Kucing = true
```

Namun, coba tambahkan pada baris ke-11 kode berikut:

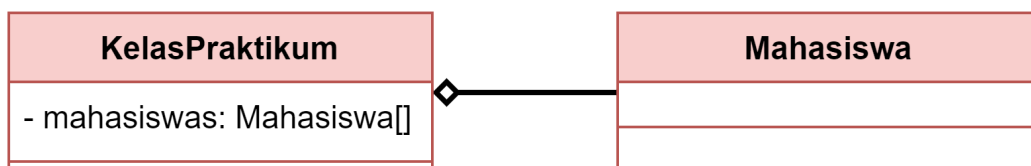
```
System.out.println("Apakah Kucing IS-A Anjing = " + (kucing instanceof Anjing));
```

Bahkan sebelum kalian *run* IDE kalian akan menunjukkan kepada kalian bahwa kode tersebut akan error. Jika tetap kalian *run*, *compiler* akan menampilkan error “*Incompatible conditional operand types Kucing and Anjing*”. Ini dikarenakan Kucing meskipun Hewan ia bukanlah Anjing.

- **Has-a relation**

Dalam PBO, konsep "has-a" relation atau agregasi (*aggregation*) mengacu pada hubungan antara dua kelas, di mana suatu objek dari kelas A memiliki satu atau lebih objek dari kelas B sebagai member atau atribut dari dirinya.

Misalnya, Kelas Praktikum memiliki (banyak) Mahasiswa. Jika digambarkan dalam class diagram adalah sebagai berikut:



Atau dalam kode Java:

Mahasiswa.java

```

3  public class Mahasiswa {
4
5      String nim, nama;
6
7      public Mahasiswa(String nim, String nama) {
8          this.nim = nim;
9          this.nama = nama;
10     }
11
12 }

```

KelasPraktikum.java

```

3  ∨ public class KelasPraktikum {
4
5      String name;
6      // Anggap satu mahasiswa saja untuk simplifikasi
7      Mahasiswa mahasiswa;
8
9  ∨ public KelasPraktikum(String name, Mahasiswa mahasiswa) {
10     this.name = name;
11     this.mahasiswa = mahasiswa;
12 }
13
14 }

```

App.java

```

3 public class App {
4
5     Run | Debug
6     public static void main(String[] args) {
7
8         Mahasiswa mahasiswa = new Mahasiswa(nim:"202310370311001", nama:"Satrio Piningit");
9         KelasPraktikum kelasPraktikum = new KelasPraktikum(name:"PBO Z", mahasiswa);
10
11         System.out.println(
12             "Kelas Praktikum " + kelasPraktikum.name
13             + " memiliki mahasiswa bernama " + kelasPraktikum.mahasiswa.nama);
14     }
15 }
16 }







```

Output:

Kelas Praktikum PBO Z memiliki mahasiswa Satrio Piningit

Perhatikan objek **mahasiswa** menjadi bagian dari objek **kelasPraktikum**. Dengan demikian **KelasPraktikum** *has-a* (memiliki sebuah) **Mahasiswa**.

- Relasi lain

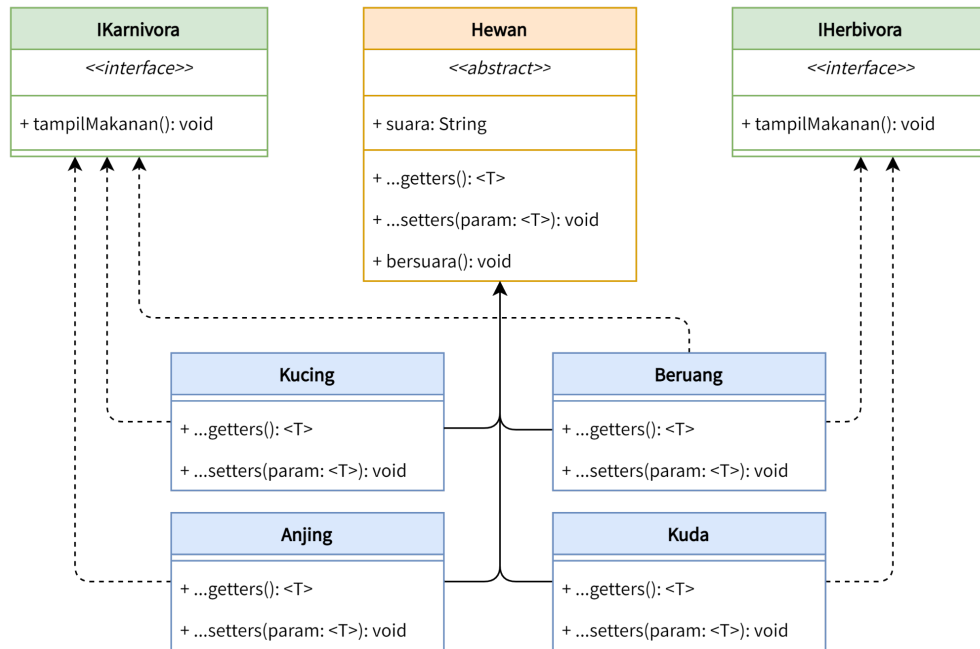
Relationship	UML Connector
Inheritance	
Interface inheritance	
Dependency	
Aggregation	
Association	
Directed association	

Sebenarnya masih ada relasi lain dalam class diagram, dengan pengantar 2 relasi di atas, mungkin tanpa disadari saat Anda mengerjakan proyek Java, Anda sudah mengimplementasikan relasi yang lain. Untuk memahami teori relasi yang lain, silakan Anda berselancar di internet.

## CODELAB

Berdasarkan contoh implementasi abstract class di atas:

- Tambahkan kelas hewan Kuda dan Beruang. Dengan kucing dan anjing, total ada 4 hewan.
- Kategorikan hewan-hewan tersebut berdasarkan jenis makanannya.
- Kira-kira di bawah inilah diagramnya.



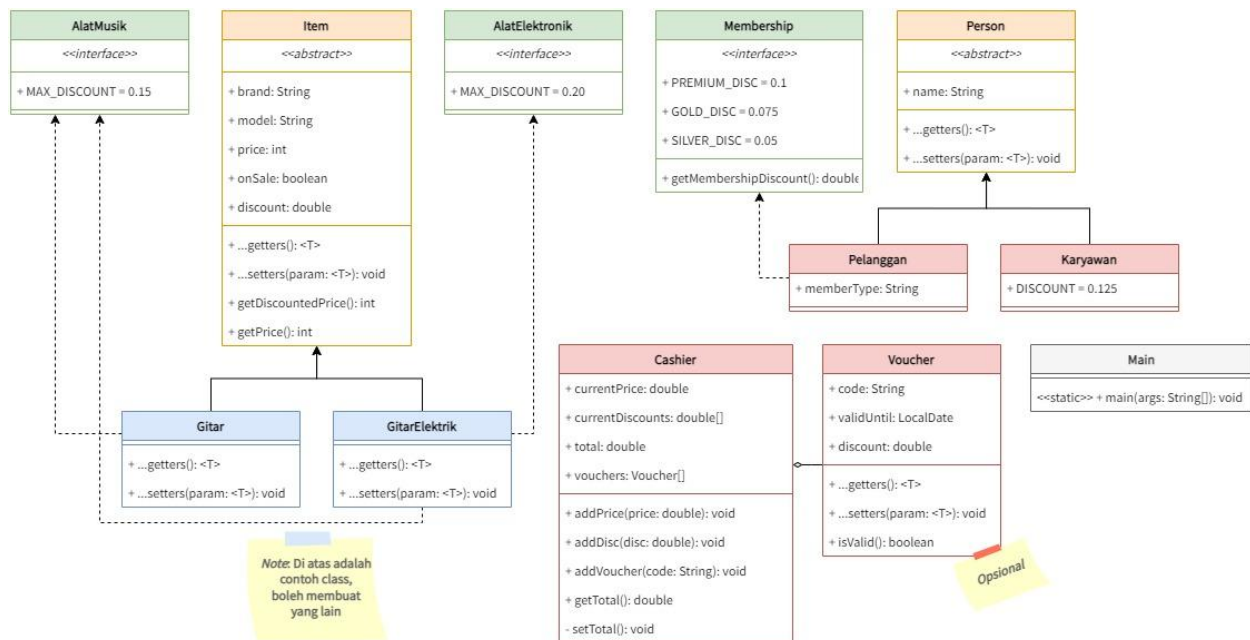
- Tampilkan objek dari semuanya seperti berikut:

```

Hewan : Beruang
Jenis : Karnivora + Herbivora
Makanan : Daging + Tumbuhan
Suara : Haummmm
  
```



## TUGAS



1. Anda diminta untuk membuat sistem kasir suatu toko yang menjual alat musik dan barang elektronik. Toko tersebut memiliki sistem membership untuk pelanggan yang dibagi menjadi 3: Premium, Gold, dan Silver, masing-masing memiliki potongan harga 10%, 7.5% dan 5% dan jika tidak menjadi member maka tidak mendapatkan potongan (diperiksa dengan atribut `memberType`). Selain itu, karyawan toko tersebut jika membeli juga dapat memiliki potongan sendiri yaitu 12.5%.
2. Barang-barang tertentu dapat diberi potongan tertentu sesuai kategorinya (menggunakan *interface* seperti di diagram). Ketentuannya alat musik dapat memiliki diskon maksimal 15% dan barang elektronik maksimal 20%. Penghitungan diskon ini dicek terlebih dahulu atribut `onSale`-nya apabila `true` maka diskon bisa diterapkan begitu pula sebaliknya.
3. **[Optional]** Toko tersebut dapat menerapkan sistem voucher. Anda dapat menentukan sendiri kode vouchernya dan besaran potongannya. Ketentuan lainnya adalah kode voucher memiliki rentang tanggal berlaku, dan hanya bisa di-*apply* apabila tanggal pada PC anda ada dalam rentang tanggal tersebut. Gunakan fungsi yang sudah disediakan di **java.time.LocalDate** perhatikan apakah tanggal di PC anda sebelum tanggal yang tertera

di atribut `validUntil` jika tidak voucher tidak valid tetapi program tetap berjalan. Pengecekan ini dilakukan di method `isValid()` pada class **Voucher**.

**Hint:**

- Harga per barang diambil dengan method `getDiscountedPrice()` pada kelas Item dan harus sudah dihitung dengan diskon per barang-nya.
- Objek dari class **Cashier** digunakan untuk melakukan operasi Kasir.
  - Method `addPrice()` digunakan untuk menambah harga dari barang yang ditambahkan
  - Method `addDisc()` digunakan untuk menambahkan potongan harga dari membership (dan karyawan) dan voucher.
  - Method `getTotal()` akan menampilkan harga akhir.
  - Method `addVoucher(code)` akan menerima parameter kode voucher inputan dari user. Kemudian membandingkan dengan semua voucher yang ada pada atribut **vouchers**. Jika voucher valid, akan memanggil method `addDisc()` dengan paramater discount yang di dapat dari atribut voucher.
- Semua instansiasi terjadi di kelas **Main** pada method `main()`;

4. Jelaskan kepada asisten Anda kapan harus menggunakan abstract class atau interface!

**RUBRIK PENILAIAN**

ASPEK PENILAIAN	POIN
Codelab	5
Implementasi tugas	35
Implementasi tugas opsional	15
Pemahaman	45
<b>TOTAL</b>	<b>100</b>

**Selamat Mengerjakan  
Tetap Semangat**