PRAKTIKUM
PEMROGRAMAN BERBASIS OBJEK

MODUL
Interface & Abstract Class

Disusun Oleh:
Tim Dosen Pengampu Mata Kuliah

PROGRAM STUDI S1 SISTEM INFORMASI
FAKULTAS TEKNOLOGI INFORMASI DAN BISNIS
INSTITUT TEKNOLOGI TELKOM SURABAYA
2023

DAFTAR ISI

MODUL
INTERFACE & ABSTRACT CLASS

Pada praktikum kali ini melanjutkan belajar tentang konsep interface dan abstract class pada pemrograman berbasis objek dengan Java. Namun sebelumnya diberikan terlebih dahulu penjelasan tentang nested class. Kegiatan ini agar mahasiswa dapat memahami, menguraikan serta menerapkan interface dan abstract class menggunakan bahasa pemrograman java.

1. Pengenalan Konsep Nested Class

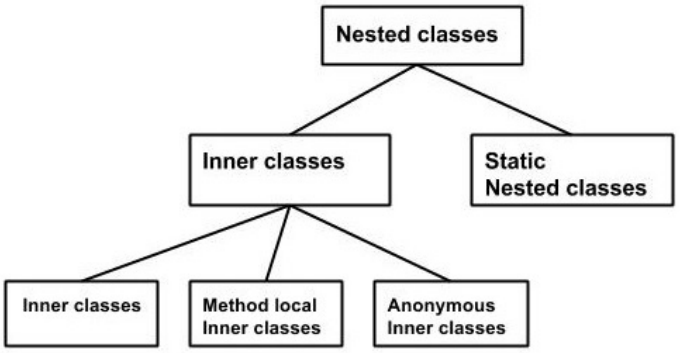
Di Java, seperti halnya sebuah method yang bisa berada di dalam sebuah class, sebuah class juga bisa berada di dalam class tertentu yang lainnya. Menulis class di dalam class yang lainnya diperbolehkan di Java. Class yang ditulis di dalam class disebut dengan **nested class**, dan class yang menampung **inner class** disebut **outer class**.

Berikut adalah syntax untuk menulis nested class. Di sini, class **OuterDemo** adalah **outer class** dan class **InnerDemo** adalah **inner class**.

```
class OuterDemo {
    class InnerDemo {
    }
}
```

Perhatikan gambar di bawah ini. Gambar tersebut memperlihatkan bahwa nested class dapat dibagi menjadi 2 jenis, yaitu:

1. **Non-static nested class (Inner Class):** tidak menggunakan keyword **“static”**.
2. **Static nested class:** menggunakan keyword **“static”**.



1.1 Inner Class (Non-Static Nested Class)

Inner class adalah diantara mekanisme keamanan di dalam Java. Kita tahu buah class tidak dapat diasosiasikan dengan access modifier **private**, tetapi jika kita memiliki class di dalam class, maka yang berada di dalam class tersebut dapat kita berikan keyword **“private”** sebagai access modifier nya.

Inner class dapat dibagi menjadi 3 jenis tergantung pada bagaimana dan di mana anda mendefinisikannya. Tiga jenis tersebut adalah sebagai berikut:

1. Inner Class
2. Method-local Inner Class
3. Anonymous Inner Class

Inner Class

Membuat inner class cukup sederhana. Anda hanya perlu menulis class di dalam class. Tidak seperti class biasa, inner class bisa bersifat private dan setelah anda mendeklarasikan inner class dengan access modifier private, inner class tersebut tidak dapat diakses dari objek di luar class tersebut.

Berikut ini adalah program untuk membuat inner class dan mengaksesnya. Dalam contoh yang diberikan, kami menjadikan inner class menjadi private dan mengakses kelas melalui sebuah method.

```
class OuterDemo {
    int num;

    // Inner Class
    private class InnerDemo {
        public void print() {
            System.out.println("Ini adalah sebuah Inner Class");
        }
    }

    // Mengakses Inner Class nya melalui method
    void displayInner() {
        InnerDemo inner = new InnerDemo();
        inner.print();
    }
 }

public class BasicInner {

    public static void main(String args[]) {
        // Membuat object outer class
        OuterDemo outer = new OuterDemo();

        // Mengakses method displayInner
        outer.displayInner();
    }
}
```

Di sini Anda dapat mengamati bahwa **OuterDemo** adalah **outer class**, **InnerDemo** adalah **inner class**, **displayInner()** adalah method di mana kita membuat instance (object) inner class, dan method ini dipanggil dari method **main()**.

Mengakses Atribut Private

Seperti yang disebutkan sebelumnya, inner class juga digunakan untuk mengakses atribut private dari sebuah class. Misalkan, sebuah class memiliki atribut private untuk mengaksesnya. Tulis inner class di dalamnya, kembalikan atribut private dari method di dalam inner class, katakanlah method **getValue()**, dan akhirnya dari kelas lain (dari mana Anda ingin mengakses atribut private tersebut) panggil metode **getValue()** dari inner class nya.

Untuk membuat instance (object) inner class, awalnya anda harus membuat instance outer class. Setelah itu, dengan menggunakan objek outer class, anda dapat membuat instance inner class dengan cara seperti berikut ini:

```
OuterDemo outer = new OuterDemo();
OuterDemo.InnerDemo inner = outer.new InnerDemo();
```

Kode program berikut ini menunjukkan cara mengakses atribut private dari sebuah class menggunakan pendekatan inner class.

```
class OuterDemo {
    // Atribut private dari outer class
    private int num = 175;

    // Inner Class
    public class InnerDemo {
        public int getNum() {
            System.out.println("Ini adalah method getNum() dari Inner Class");
            return num;
        }
    }
 }

public class PrivateMember {
    public static void main(String args[]) {
        // Buat object outer class
        OuterDemo outer = new OuterDemo();

        // Buat object inner class
        OuterDemo.InnerDemo inner = outer.new InnerDemo();

        System.out.println(inner.getNum());
    }
}
```

Method-local Inner Class

Di Java, kita bisa menulis sebuah class di dalam sebuah method dan ini akan menjadi **local type**. Seperti **local variable**, ruang lingkup inner class dibatasi dalam method. Class dalam method-local dapat dibuat hanya dalam metode di mana kelas dalam didefinisikan. Kode program berikut ini menunjukkan cara menggunakan **method-local inner class**:

```java
public class OuterClass {
    // Method dari Outer Class
    void myMethod() {
       int num = 23;

       // Method-local Inner Class
       class MethodInnerDemo {
          public void print() {
             System.out.println("Ini adalah method inner class " + num);
          }
       } // end of inner class

       // Mengakses inner class
       MethodInnerDemo inner = new MethodInnerDemo();
       inner.print();
    }

    public static void main(String args[]) {
        OuterClass outer = new OuterClass();
       outer.myMethod();

    }
}
```

Anonymous Inner Class

Inner class yang dideklarasikan tanpa nama class dikenal sebagai **Anonymous Inner Class**. Dalam kasus anonymous inner class, kami mendeklarasikan dan memberi contoh pada saat yang sama. Umumnya, mereka digunakan kapan pun anda perlu meng-override method class atau interface. Sintaks anonymous class adalah sebagai berikut:

```java
AnonymousInner an_inner = new AnonymousInner() {
   public void my_method() {
      ........
      ........
   }
};
```

Kode program berikut ini menunjukkan cara meng-override method class menggunakan anonymous inner class.

```java
abstract class AnonymousInner {
    public abstract void myMethod();
}

public class OuterClass {
    public static void main(String args[]) {
        AnonymousInner inner = new AnonymousInner() {
            public void myMethod() {
                System.out.println("Ini adalah contoh dari Anonymous Inner Class");
            }
        };
```

```
        inner.myMethod();
    }
}
```

1.2 Static Nested Class

Static nested class adalah nested class yang merupakan static atribut dari outer class. Itu dapat diakses **tanpa** membuat instance (object) outer class. Sama seperti static atribut, static nested class tidak memiliki akses ke variabel dan metode outer class. Syntax static nested class adalah sebagai berikut:

```
class MyOuter {
   static class Nested_Demo {
   }
}
```

Membuat instance (object) static nested class sedikit berbeda dengan membuat instance (object) inner class. Kode program berikut menunjukkan cara menggunakan static nested class:

```
public class StaticNested {
    static class NestedDemo {
        public void myMethod() {
            System.out.println("Ini adalah nested class");
        }
    }

    public static void main(String args[]) {
        StaticNested.NestedDemo nested = new StaticNested.NestedDemo();
        nested.myMethod();
    }
}
```

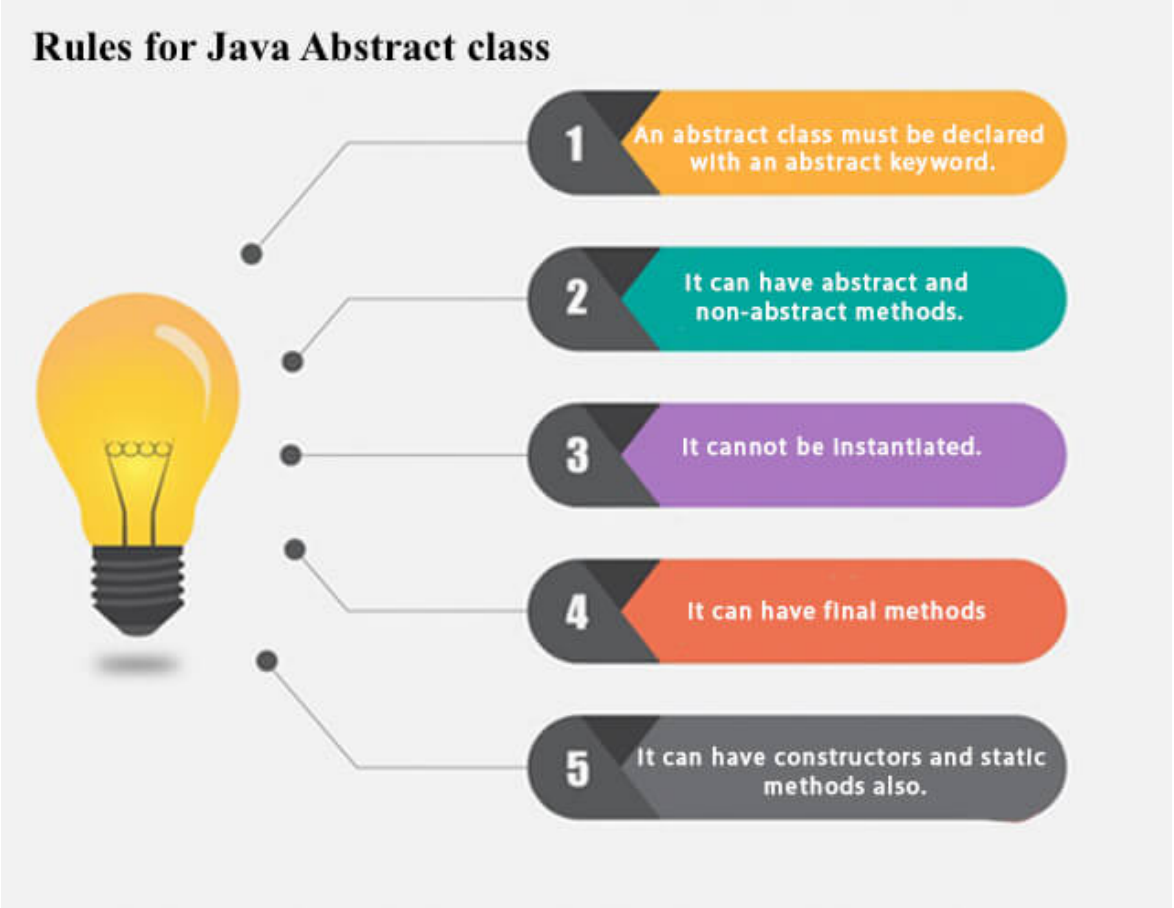
2. Pengenalan Konsep Abstract Class

Berdasarkan kamus, kata abstraksi adalah suatu hal yang berurusan dengan ide daripada aksi (action). Misalnya, ketika anda berbicara tentang sebuah email, detail rumit seperti apa yang terjadi setelah anda mengirim email, protokol yang digunakan server email anda disembunyikan dari pengguna. Dan untuk mengirimkan email anda hanya perlu mengetikkan isinya, sebutkan alamat penerimanya, dan klik kirim.

Demikian juga di dalam pemrograman berorientasi objek, abstraksi adalah proses menyembunyikan detail implementasi dari pengguna, hanya fungsionalitas yang akan diberikan kepada pengguna. Dengan kata lain, pengguna akan memiliki informasi tentang apa yang dilakukan objek daripada bagaimana melakukannya. Di Java, abstraksi dicapai dengan menggunakan **abstract class** dan **interface**.

Class yang berisi keyword **“abstract”** dalam deklarasinya dikenal sebagai **abstract class**.

- Abstract class mungkin berisi atau tidak berisi **abstract method**, yaitu method tanpa body. Contoh: **public void get();**
- Method yang dideklarasikan sebagai abstract dan tidak memiliki implementasi dikenal sebagai **abstract method**.
- Namun, jika suatu class memiliki setidaknya satu **abstract method**, maka class tersebut harus dinyatakan / dideklarasikan menggunakan keyword **"abstract"**.
- Jika sebuah class dideklarasikan sebagai **abstract**, kelas tersebut **tidak dapat** dibuat instance-nya (object-nya).
- Untuk menggunakan abstract class, anda harus mewarisinya (inherit) dari class lain, mendefiniskan "implementasinya" ke dalam abstract method di dalamnya.
- Jika anda mewarisi abstract class, anda harus menyediakan "implementasi" untuk semua metode abstrak di dalamnya.



Contoh **Abstract Class**:

```
abstract class A {}
```

Contoh **Abstract Method**:

```
abstract void printStatus(); //no method body and abstract
```

Dalam contoh di bawah ini, **Bike** adalah **abstract class** yang berisi **constructor**, satu **abstract method** dan satu **void method**. Implementasinya disediakan oleh class **Honda**.

```
abstract class Bike {
    Bike() {
        System.out.println("bike is created");
    }

    abstract void run();

    void changeGear() {
        System.out.println("gear changed");
    }
}

public class Honda extends Bike{
    @Override
    void run() {
        System.out.println("running safely");
    }

    public static void main(String args[]) {
        Bike obj = new Honda();
        obj.run();
        obj.changeGear();
    }
}
```

2. Pengenalan Konsep Interface

Interface di Java adalah cetak biru (blue print) sebuah class. Ia memiliki **static constant** dan **abstract method**. Interface di Java adalah mekanisme untuk mencapai abstraksi. Hanya ada deklarasi abstract method di sebuah **interface** (tidak ada body method nya). Ini digunakan untuk mencapai abstraksi dan pewarisan berganda (multiple inheritance) di Java.

Dengan kata lain, kita dapat mengatakan bahwa interface dapat memiliki abstract method dan abstract variable dan tidak memiliki **method body**. Interface pada Java juga mewakili **IS-A relationship** dan sama seperti abstract class, interface tidak bisa di-instantiate (dibuat objectnya).

Sebuah interface dideklarasikan dengan menggunakan keyword **interface**. Ini memberikan abstraksi total, berarti semua method dalam interface dideklarasikan dengan **body kosong**, dan semuanya bersifat publik, static, dan final secara default. Class yang mengimplementasikan interface harus mengimplementasikan **semua method** yang dideklarasikan dalam **interface**. Contoh penggunaan **interface** adalah sebagai berikut:

```
interface Animal {
   public void eat();
   public void travel();
}

public class Mamal implements Animal {
   @Override
   public void eat() {
      System.out.println("Mammal eats");
   }

   @Override
   public void travel() {
      System.out.println("Mammal travels");
   }

   public int noOfLegs() {
      return 0;
   }

   public static void main(String args[]) {
      Mamal m = new Mamal();
      m.eat();
      m.travel();
   }
}
```

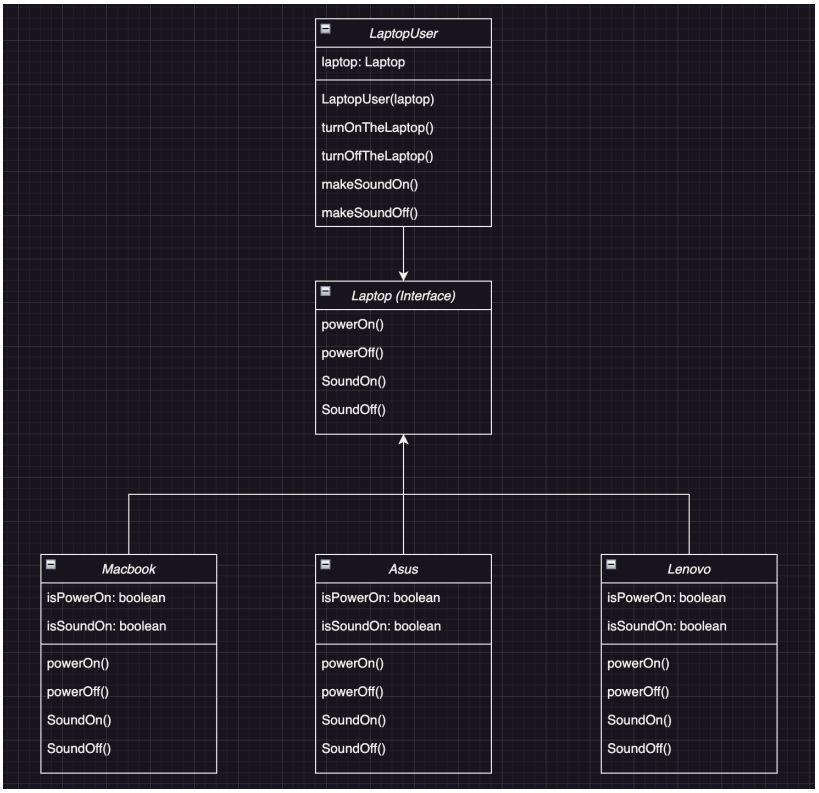
Contoh **multiple inheritance** dengan **interface**:

```
interface Printable{
    void print();
}

interface Showable{
    void show();
}

public class A7 implements Printable, Showable{
    @Override
    public void print() {
        System.out.println("Hello");
    }

    @Override
    public void show() {
        System.out.println("Welcome");
    }
```

```
    public static void main(String args[]){
        A7 obj = new A7();

        obj.print();
        obj.show();
    }
}
```

6. Latihan

Terdapat dua hal yang akan kita hubungkan dengan **interface**. Yakni manusia (laptop user) dan laptop. Class diagram nya adalah sebagai berikut. Implementasikan class diagram dibawah ini dengan pendekatan interface.



- Buat **interface Laptop** yang memiliki 4 **abstract method**.
- Buat class **LaptopUser** yang memiliki **1 constructor** dan **4 void method** dengan kode program seperti di bawah ini.

```
public class LaptopUser {

    private Laptop laptop;

    public LaptopUser(Laptop laptop) {
        this.laptop = laptop;
    }

    void turnOnTheLaptop(){
```

```
        this.laptop.powerOn();
    }

    void turnOffTheLaptop(){
        this.laptop.powerOff();
    }

    void makeSoundOn(){
        this.laptop.soundOn();
    }

    void makeSoundOff(){
        this.laptop.soundOff();
    }
}
```

- Buat class **Macbook**, **Asus** dan **Lenovo**.
- Berikut ini adalah contoh dari class **Macbook**.

```
public class Macbook implements Laptop {

    private boolean isPowerOn;
    private boolean isSoundOn;

    public Macbook() {}

    @Override
    public void powerOn() {
        isPowerOn = true;
        System.out.println("Laptop menyala...");
        System.out.println("Selamat datang di Macbook...");
    }

    @Override
    public void powerOff() {
        isPowerOn = false;
        System.out.println("Laptop dimatikan...");
    }

    @Override
    public void soundOn() {
        System.out.println("Mencoba menyalakan sound...");
        if (isPowerOn) {
            isSoundOn = true;
            System.out.println("Sound menyala...");
        } else {
            isSoundOn = false;
            System.out.println("Oops. Nyalakan dulu Laptopnya...");
        }
    }

    @Override
    public void soundOff() {
        System.out.println("Mencoba menyalakan sound...");
```

```
        if (isPowerOn) {
            isSoundOn = false;
            System.out.println("Sound dimatikan...");
        } else {
            isSoundOn = false;
            System.out.println("Nyalakan dulu Laptopnya...");
        }
    }

    public void getStatus() {
        if (isPowerOn) {
            System.out.println("Laptop menyala...");
        } else {
            System.out.println("Laptop mati...");
        }

        if (isSoundOn) {
            System.out.println("Sound menyala...");
        } else {
            System.out.println("Sound mati...");
        }
    }

}
```

- Buat program utama dari **interface** dan **class** di atas, yang jika dijalankan menghasilkan output sebagai berikut.

```
Laptop menyala...
Selamat datang di Macbook...
Mencoba menyalakan sound...
Sound menyala...
Laptop dimatikan...
Mencoba menyalakan sound...
Oops. Nyalakan dulu Laptopnya...
```

- Pahami dan analisis konsep dan alur dari kode program yang telah dibuat!

7. Tugas

1. **Praktekkan** poin praktikum dan **Latihan** yang ada di atas secara **INDIVIDU**.
2. Dokumentasikan seluruhnya **(screenshot kode program, output, dan penjelasan)** dalam bentuk Laporan Praktikum.
3. Kumpulkan Laporan Praktikum **(.pdf)** yang telah dibuat di dalam praktikum via **E-Learning** paling lambat sebelum jadwal praktikum minggu depan.