IPB University
Bogor Indonesia

KOM120C -- PEMROGRAMAN

Object Oriented Programming

OOP using Java

Tim Pengajar Pemrograman IPB University

Mengapa Java

Sederhana (Simple) - Bahasa pemrograman Java menggunakan sintaks mirip dengan C++, namun sintaks pada Java telah banyak diperbaiki terutama menghilangkan penggunaan pointer yang rumit.

Terdistribusi (Distributed) - Java dibuat untuk membuat aplikasi terdistribusi secara mudah dengan adanya libraries networking yang terintegrasi pada Java.

Interpreted - Program Java dijalankan menggunakan interpreter yaitu Java Virtual Machine (JVM). Hal ini menyebabkan source code Java yang telah dikompilasi menjadi Java bytecodes dapat dijalankan pada platform yang berbeda-beda.

Tetapi instruksinya panjang untuk melakukan suatu task tertentu.

Java

Free for download

Unit terkecil program Java adalah Class yang terdiri dari methods (C:procedure) dan instance (C: data)

```
public class Hello {
    public static void main(String[] args) {
        // menampilkan string ke layar
        System.out.println("Hello world!");
    }
}
```

Perlu diperhatikan

Program Java harus disimpan ke dalam file dengan nama *.java

Nama File seharusnya sama dengan nama class public nya.

Program yang berada pada satu folder dianggap sebagai satu package (kumpulan dari satu atau lebih class).

Buat komentar secukupnya untuk memperjelas kode program.

Bandingkan

```
class Counter
{
    private:
        int c;
    public:
        Counter() { c=0; }
        void set(int n)
           { c=(n<0)?0:n; }
        void inc() { c++; }
        void dec()
           { c=(c-1<0)?0:c-1; }
        void prin()
           { cout << c << endl; }
};
```

```
public class Counter
{
    private int c;
    public Counter() { c=0; }
    public void set(int n)
        { c=(n<0)?0:n; }
    public void inc() { c++; }
    public void dec()
        { c=(c-1<0)?0:c-1; }
    public void prin()
        { System.out.println(c); }
};
```

Java User Input

Class Scanner digunakan untuk membaca data dari standard input, dan terdapat di dalam package java.util.

Beberapa method yang dimiliki class Scanner:

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

Class Scanner

Contoh program menjumlahkan 2 bilangan bulat.

```
import java.util.Scanner;
public class Input {

    public static void main (String[] args) {
        int a, b, result;
        Scanner inp = new Scanner(System.in);
        a = inp.nextInt();
        b = inp.nextInt();
        result = a + b;
        System.out.println("Hasil penjumlahan: " + result);
    }
}
```

Standard Output

```
System.out.print(parameter);
```

```
System.out.println(parameter);
```

```
System.out.printf(parameter);
```

```
import java.io.*;

public class Tulis {
    public static void main (String[] args) {
        int a=5;
        System.out.print("Hasil: "+a);
        System.out.println(a+10);
        System.out.printf("%.2f\n", Math.PI);
    }
}
```

Array dalam Java

Sama dengan program dalam C

Mendeklarasikan variabel array:

```
int []usia;
```

atau

```
int usia[];
```

Membuat objek array (dalam Java disebut sebagai instantiation)

```
int usia[];
usia = new int[100];
```

atau bisa juga ditulis sekaligus menjadi

```
int usia[] = new int[100];
```

Membaca Data Bentuk Matrik

Input Data:

```
3 4
1 2 3 4
5 6 7 8
9 0 1 2
```

- Baca satu baris data menggunakan class **Reader** atau **Scanner**.
- Split (pisahkan) setiap nilai yang dipisahkan oleh spasi

Class Objek dan Class Driver

Class Objek dan Class Driver dapat diletakkan dalam satu class, atau terpisah

```
class Object
{
    // elements of class Object
}

public class myClass
{

    public static void main (String[] args)
    {
        //
    }

}
```

TERPISAH

```
public class myClass {

    // elements of class Object

    public static void main (String[] args)
    {
        //
    }

}
```

DALAM SATU CLASS

Latihan

Gunakan Java OOP

Deskripsi

Buat program mengelola bilangan pecahan a b/c.

Format Masukan

Beberapa baris operasi bilangan pecahan:

set a b c	inisialisasi bilangan pecahan a b/c
p	menampilkan bilangan pecahan sesederhana mungkin
add a b c	menambah bilangan pecahan yang ada dengan a b/c
mul a b c	mengalikan bilangan pecahan yang ada dengan a b/c
inc	postfix increment
dec	postfix decrement
end	akhir dari pengolahan

Format Keluaran

Beberapa baris bilangan pecahan sesederhana mungkin.

Contoh Input

```
set 4 2 8
p
add 0 6 8
inc
p
```

Contoh Output

```
4 1/4
5
```

IPB University
Bogor Indonesia

KOM120C -- PEMROGRAMAN

Object Oriented Programming

- Java Interface
- Java Collections Framework

Tim Pengajar Pemrograman IPB University

Interface

Interface dalam Java didefinisikan sebagai tipe abstrak yang digunakan untuk menentukan *behaviour* dari suatu class.

Java interface berisi *static constants* dan *abstract methods* (*pure virtual function* dalam C++).

Interface tidak memiliki *constructor*.

Sintaks:

```
interface <interface-identifier> {
    // declare constant fields
    // declare abstract methods
}
```

Class vs Interface

Class	Interface
Dengan class, kita DAPAT meng-instansiasi variabel dan membuat objek.	Interface, kita TIDAK DAPAT meng-instansiasi variabel atau membuat objek.
Class dapat berisi method yang kongkrit (memiliki implementasi).	Interface TIDAK berisi method yang kongkrit. Interface dapat berisi abstract method.
Access specifier anggota class adalah private, protected, dan public.	Access specifier anggota interface hanya public.

Implementasi Interface

Implementasi interface dalam Java menggunakan keyword **implements**

Contoh kasus: objek kendaraan (motor dan mobil) memiliki fungsi yang umum, yaitu (1) mengubah "gigi", dan (2) menambah/mengurangi kecepatan.

```
interface Kendaraan {
    // abstract methods.
    void ubahGigi(int a);
    void tambahKecepatan(int a);
    void kurangiKecepatan(int a);
}
```

Definisi class Mobil mirip dengan definisi class Motor.

```
class Motor implements Kendaraan {
    int kecepatan;
    int gigi;

    @Override
    public void ubahGigi(int g)
        { gigi = g;}

    @Override
    public void tambahKecepatan (int s)
        { kecepatan = kecepatan - s; }

    @Override
    public void kurangiKecepatan (int s)
        { kecepatan = kecepatan - s; }

    public void show() {
        System.out.println("kecepatan: " +
        kecepatan + " gigi: " + gigi); }
}
```

Java Collections Framework (JCF)

Mirip dengan STL (*Standard Template Library*) dalam C++.

std::set	java.util.HashSet
std::unordered_set	java.util.LinkedHashSet
std::vector	java.util.ArrayList
std::map	java.util.TreeMap
Kita dapat meneruskan (pass) fungsi sebagai parameter.	Tidak dapat.
C++ tidak mengenal interface.	Tergantung pada Interface.

Java Set

Definisi:

```
// Obj adalah tipe yang akan disimpan ke dalam Set
Set<Obj> set = new HashSet<Obj>();
```

```
import java.util.*;
public class ContohSet {
   public static void main(String[] args) {
      Set<String> hs = new HashSet<String>();

      hs.add("B");
      hs.add("A");
      hs.add("B");
      hs.add("D");
      hs.add("C");

      System.out.println(hash_Set);
   }
}
```

Output:

```
[A, B, C, D]
```

Set of Class

Contoh penggunaan Set untuk mengelola koleksi buku.

```
import java.util.*;

class Buku {
    int id;
    String judul,pengarang,penerbit;
    int jumlah;
    public Buku(int id, String judul, String pengarang,
                                          String penerbit, int jumlah)
    {
        this.id = id;
        this.judul = judul;
        this.pengarang = pengarang;
        this.penerbit = penerbit;
        this.jumlah = jumlah;
    }
}
```

Set of Class

Contoh penggunaan Set untuk mengelola koleksi buku.

```
public class Koleksi {
    public static void main(String[] args) {
        LinkedHashSet<Buku> kol=new LinkedHashSet<Buku>();

        Buku b1=new Buku(101,"Programming","Allan B Tucker","CSB",8);
        Buku b2=new Buku(102,"Data Communications","Forouzan","MGH",4);
        Buku b3=new Buku(103,"Operating System","Galvin","Wiley",6);

        kol.add(b1);
        kol.add(b2);
        kol.add(b3);

        for(Buku b:kol) {
            System.out.println(b.id+" "+b.judul+" "+b.jumlah);
        }
    }
}
```

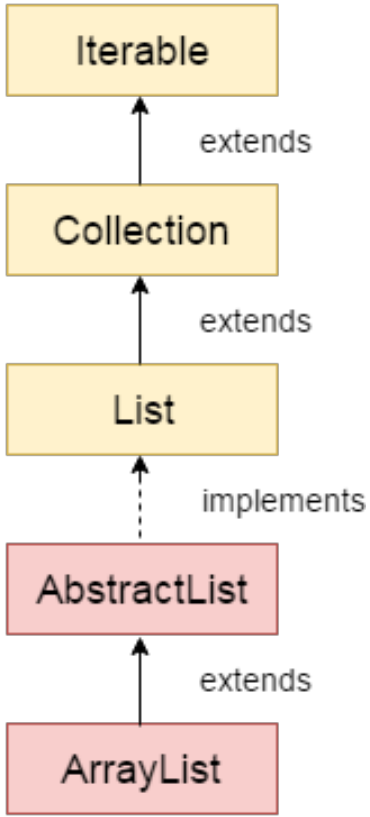
Java ArrayList

Dynamic array dalam Java

Seperti Vector dalam C++.

Tidak seperti Set, ArrayList dapat berisi elemen yang sama (tidak unik).

Lebih fleksibel dibanding array biasa. Mengapa?



Contoh ArrayList

```
import java.util.*;
public class ContohArrayList {
   public static void main(String args[]){
      ArrayList<String> list=new ArrayList<String>();
      list.add("Mangga");
      list.add("Apel");
      list.add("Pisang");
      list.add("Jambu");

      // Print
      System.out.println(list);

      Iterator it=list.iterator();
      while(itr.hasNext()) { // memeriksa iterator memiliki nilai
         System.out.println(itr.next());
      }
   }
}
```

Output:

```
[Mangga, Apel, Pisang, Jambu]
Mangga
Apel
Pisang
Jambu
```

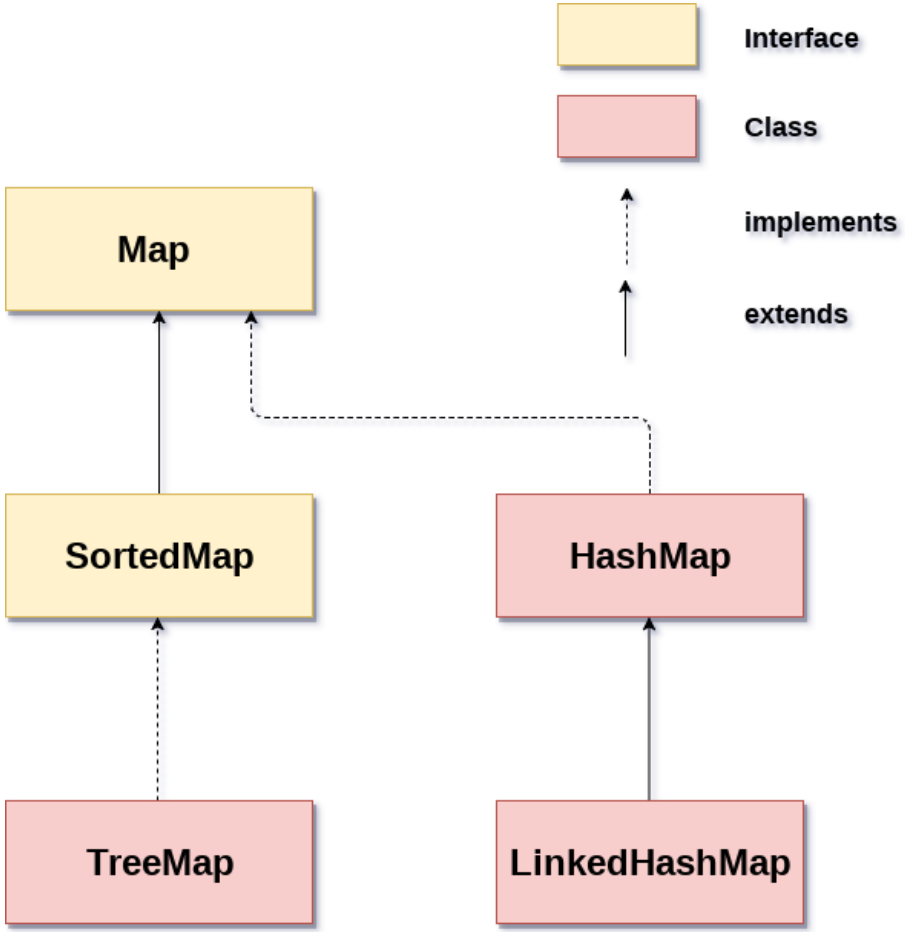
Java Map

The hierarchy of Java Map

Seperti Map dalam C++.

Mengandung key (tidak boleh duplikat) dan value.

Lebih fleksibel dibanding array biasa. Mengapa?



Contoh Java Map

```
import java.util.*;
public class ContohMap {
    public static void main(String args[]){
        Map<Integer,String> map=new HashMap<Integer,String>();
        map.put(100,"Amir");
        map.put(102,"Rudi");
        map.put(101,"Shanti");
        map.put(100,"Abdullah");

        for(Map.Entry m:map.entrySet()) {
            System.out.println(m.getKey()+" "+m.getValue());
        }

        // sort by value
        map.entrySet()
        .stream()
        .sorted(Map.Entry.comparingByValue())
        .forEach(System.out::println);
    }
}
```

Output:

```
100 Abdullah
101 Shanti
102 Rudi
100=Abdullah
102=Rudi
101=Shanti
```

Latihan

Gunakan Java OOP

Deskripsi

Buat program untuk membaca beberapa bilangan bulat, dan menghapus beberapa bilangan pada posisi tertentu.

Format Masukan

Bagian pertama berisi beberapa bilangan bulat yang diakhiri dengan nilai -9 (nilai ini tidak ikut dalam pengolahan, hanya sebagai akhir data masukan). Bagian kedua adalah beberapa bilangan terurut dari kecil ke besar yang menunjukkan elemen keberapa dari bilangan masukan yang dihapus. Input bagian ini juga diakhiri dengan -9 (tidak ikut diolah)

Format Keluaran

Baris pertama adalah dua bilangan yang menunjukkan banyaknya bilangan awal dan banyaknya bilangan setelah dihapus beberapa elemennya. Baris kedua adalah dua nilai rata-rata yang dituliskan dalam dua digit di belakang tanda desimal dari bilangan-bilangan awal dan bilangan-bilangan setelah dihapus. Jika data dalam array kosong, maka nilai rata-rata dituliskan -9.99. Output program diakhiri dengan newline..

Contoh Input

```
10 20 30 40 50 -9 1 3 4 -9
```

Contoh Output

```
5 2
30.00 35.00
```

IPB University
Bogor Indonesia

KOM120C -- PEMROGRAMAN

Object Oriented Programming

Pewarisan dalam Java

Tim Pengajar Pemrograman IPB University

Pewarisan

Konsep-konsep dasar OO pada Java mirip dengan yang telah dipelajari pada C++ dengan beberapa perbedaan pada implementasi.

Inheritance pada Java hanya bersifat **single-inheritance**. Sebagai gantinya, Java mendukung multiple interface-inheritance. Konsep interface akan dijelaskan di pertemuan yang lain.

```
<Subclass> extends <Superclass>{...}
```

Pewarisan

Dalam Java, semua class, termasuk class yang membangun Java API, adalah subclasses dari superclass Object.

Definisi subclass

```
<modifier> class <subclass> extends <superclass> {
   <attributeDeclaration>*
   <constructorDeclaration>*
   <methodDeclaration>*
}
```

Contoh:

```
public class Student extends Person {
            //beberapa kode di sini
}
```

Super

Memanggil constructor secara eksplisit dari superclass terdekat.

Pemanggil super() hanya dapat digunakan dalam definisi constructor.

Pemanggil super() harus dijadikan sebagai pernyataan atau instruksi pertama dalam constructor.

Dapat dipakai sebagai penunjuk anggota superclass, misalnya

```
public Student() {
    super.nama="Siapa Saja";
    super.usia=17;
}
```

is-a Relationship

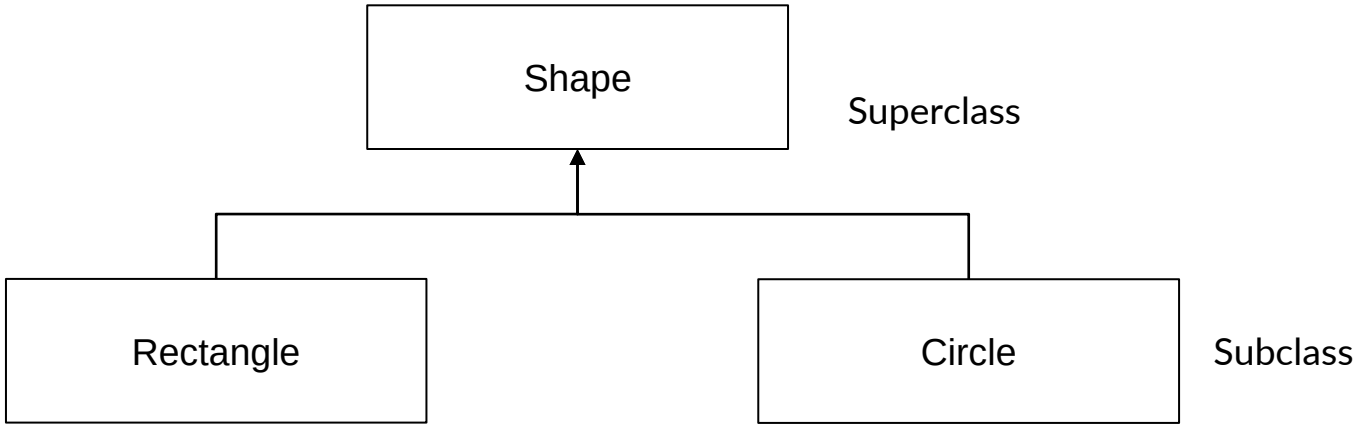
Dalam Java, pewarisan adalah bentuk hubungan **is-a**.

Artinya, kita menggunakan pewarisan hanya jika ada hubungan **is-a** antar 2 class.

Contoh:

- Car is a Vehicle
- Orange is a Fruit
- Surgeon is a Doctor
- Dog is an Animal

is-a Relationship



Contoh dan Latihan

```
class Shape{
  protected String color;
  protected double area;

  public Shape(String a){
    color = a;
  }
  public String getColor(){
    return color;
  }
  public double getArea(){
    return area;//error?}

  public void print(){
    System.out.println("Color:" + color);
  }
}
```

```
class Rectangle extends Shape{
  protected double width;
  protected double height;

  public Rectangle(String a, double b, double c){
    super(a);
    width = b;
    height = c;
  }

  public double getWidth(){return width;}
  public double getHeight(){return height;}

  public void print(){
    super.print();
    System.out.printf("W:%.2f H:%.2f\n",
      width, height);
  }
}
```

Latihan: Buatlah class Circle, turunan dari Shape, yang memiliki atribut radius. Sesuaikan implementasi fungsi print() pada class Circle.

Contoh

```
public class Driver
{
  public static void main(String[] args){
    Shape shp = new Shape("White");
    shp.print();

    Rectangle rect = new Rectangle("Black", 10, 15);
    rect.print();

    Circle circle = new Circle("Blue", 10);
    circle.print();
  }
}
```

Apakah keluaran dari program di atas? Konsep apakah yang digunakan pada contoh di atas?

Override

Subclass (kelas turunan) dapat meng-**override** method pada superclass dengan memberikan implementasi yang berbeda.

Method pada superclass akan memiliki signature dan tipe kembalian yang identik dengan method yang ada pada superclass.

Subclass akan menggunakan implementasi pada dirinya.

Superclass:

```
public class A{
  protected int x, y;
  ...
  public void calc(){return x + y;}
}
```

Subclass:

```
public class B extends A{
  protected int x, y;
  ...
  public void calc(){return x * y;}
}
```

Latihan

Tambahkan method **calculateArea()** pada class Shape, Rectangle, dan Circle.

```
public class Driver2{
  public static void main(String[] args){

    Rectangle rect = new Rectangle("Black", 10, 15);
    System.out.println(rect.getArea());

    Circle circle = new Circle("Blue", 10);
    System.out.println(circle.getArea());
  }
}
```

Output:

```
150.0
314.0
```

Apa implementasi method calculateArea() pada class Shape? Mengapa?

Keyword `final`

Keyword **final** pada Java merupakan suatu modifiers terhadap suatu class, method, atau atribut.

Pada class, final membuat bahwa definisi dari class telah lengkap dan class tersebut tidak dapat diwarisi.

```
public final MyClass{...}
```

Pada method, final membuat method tersebut tidak dapat di-*override*.

```
public final int calculate(...){...}
```

Pada atribut / variabel, final membuat atribut tersebut nilainya tidak dapat diubah setelah diinisialisasi. Inisialisasi wajib dilakukan.

```
public final String language = "Java";
```

Final

Upaya untuk menurunkan class, meng-override method, atau memberikan nilai pada atribut final akan menyebabkan kode program gagal dikompilasi.

```
class Shape{
  ...
  public final void print(){
    System.out.println("Color:" +
color);
  }
}
```

```
class Rectangle extends Shape{
  ...
  public void print(){
    super.print();
    System.out.printf("W:%.2f H:%.2f\n",
      width, height);
  }
}
```

```
Driver2.java:30: error: print() in Rectangle cannot override print() in Shape
  public void print(){
              ^
  overridden method is final
```

Latihan

Pak Agria memiliki tanah di N buah lokasi. Tanah tersebut berbentuk persegi panjang (panjang dan lebar) atau lingkaran (radius). Pak Agria memagari tanah miliknya. Akan tetapi, karena dana Pak Agria terbatas, ia hanya mampu memasang pagar di sebagian tanah miliknya. Dalam hal ini, Pak Agria hanya akan memasang pagar di tanah yang kelilingnya di atas rata-rata. Bantulah Pak Agria menghitung panjang pagar yang perlu ia sediakan.

Format Masukan:

- Baris pertama: N
- Digit pertama setiap baris adalah bentuk tanah (0 = persegi panjang, 1 = lingkaran).
- Digit-digit berikutnya adalah panjang dan lebar, atau radius bergantung pada bentuk tanah.

Contoh Masukan:

```
3
0 10.0 10.0
0 20.0 20.0
1 10.0
```

Contoh Keluaran:

```
142.83
```

Tantangan: bagaimana jika bentuk tanah pada masukan tidak tersedia?

IPB University
Bogor Indonesia

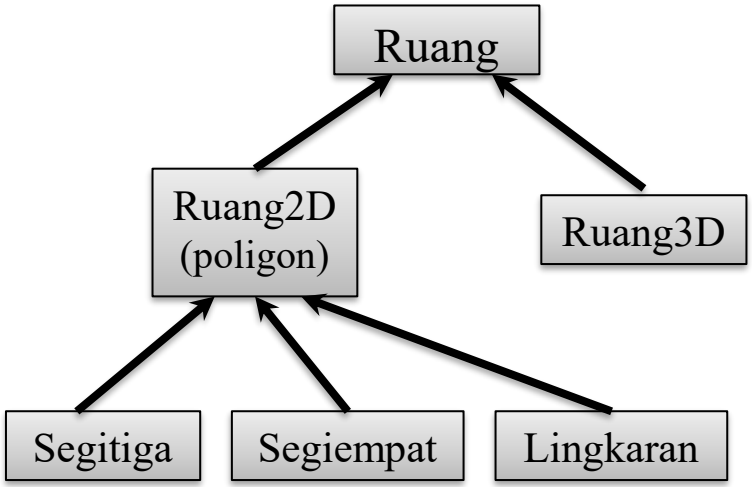
KOM120C -- PEMROGRAMAN

Object Oriented Programming

- Abstract Class
- Interface
- Multiple Inheritance

Tim Pengajar Pemrograman IPB University

Abstract Class



Class Ruang mempunyai method area() yang akan di-override oleh subclasses nya.

Ruang merupakan sifat umum dari suatu bidang dua dimensi (Segitiga, Segiempat, Lingkaran) dan tiga dimensi (Bola, Kubus).

Method dalam class Ruang tidak memiliki implementasi. Class jenis ini yang disebut dengan abstract class.

Umumnya abstract class muncul pada hirarki class pemrograman berbasis object paling atas, dan mendefinisikan keseluruhan aksi yang mungkin pada object dari seluruh subclasses dalam class.

Abstract Method

Method dalam abstract class yang tidak mempunyai implementasi dinamakan abstract method.

Only method signature, no body.

Pada C++, ditunjukkan dengan adanya method **virtual murni: virtual double area() = 0;**

Pada Java, dengan adanya deklarasi method **abstract,** Contoh:

```
public abstract class Poligon {
    public abstract double area();
}
```

Class Poligon

```
abstract class Poligon {
   protected double[] x;
   protected double[] y;
   protected int edge;

   public Poligon(int n) {
      x=new double[n]; y=new double[n]; edge=n; }

   public void set(double[] x, double[] y, int n) {
      for (int i=0; i<n; i++) {
            this.x[i]=x[i]; this.y[i]=y[i]; edge=n; }
   }

   public abstract double luas();
}
```

Interface

Interface dalam Java didefinisikan sebagai tipe abstrak untuk menentukan perilaku dari class.

Interfaces dapat memiliki abstract methods dan variables.

Interface tidak dapat dibuat obyek namun hanya dapat diimplementasi.

Sintaks:

```
interface <nameOfInterface> {
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Contoh

```
interface Car {
    void display();
}

public class Model implements Car {
    public void display() {
        System.out.println("Car model");
    }

    public static void main(String args[]) {
        Model obj = new Model();
        obj.display();
       }
}
```

Abstract Class vs Interface

Kapan kita menggunakan **abstract class** dan kapan kita menggunakan **interface:**

Abstract class:

- Ketika hubungan yang ingin ditunjukkan adalah **is-a**
- Perlu ada beberapa metode terimplementasi pada kelas parent
- Perlu menyimpan beberapa variabel/property yang ingin diwariskan
- Tidak perlu multiple inheritance. Java tidak mensupport multiple inheritance

Interface:

- Hubungan selain **is-a**
- Tidak perlu ada metode terimplementasi, semua metode abstract pada interface
- Tidak perlu menyimpan data/variabel apa-apa. Interface tidak bisa menampung variabel.
- Perlu implementasi di beberapa kelas berbeda. Java support multiple interface.

Abstract Class vs Interface

Kapan kita menggunakan **abstract class** dan kapan kita menggunakan **interface:**

Abstract class:

```
class Vehicle {
public String BrandName;
public String RegistrationNr;
abstract public void start() {...}
                ...
}
class Car extends Vehicle {}
class Boat extends Vehicle{}
```

Interface:

```
interface MovingObjects{
 abstract public void move();
 abstract public void accelerate();
}

class Car implements
MovingObjects;
class Bird implements
MovingObjects;
```

Multiple Interface

```
interface Interface1 { public void Method1(); }

interface Interface2 { public void Method2(); }

public class Multiple implements Interface1, Interface2 {
    public void Method1() {
        System.out.println("Method 1");
    }
    public void Method2() {
        System.out.println("Method 2");
    }

    public static void main(String[] args) {
        Multiple obj = new Multiple();
        obj.Method1();
        obj.Method2();
    }
}
```

Variabel dalam Interface

Harus final atau static variables.

```
interface Rectangle {
    int height = 0;
    int width = 0;

    public int getHeight();
    public int getWidth();
    public void setHeight(int h);
    public void setWidth(int w);
}

public class Shape implements Rectangle {
    public int getHeight() { return height; }
    public int getWidth() { return width; }
    public void setHeight(int h) { height = h; }
    public void setWidth(int w) { }
    // ...
}
```

Multiple Inheritance

Java tidak mendukung multiple inheritance ➔ tidak dapat extends lebih dari 1 class.

Class dalam Java dapat implements satu atau lebih interface ➔ membantu Java mengimplementasikan konsep multiple inheritance.

Gunakan extends interface.

Contoh

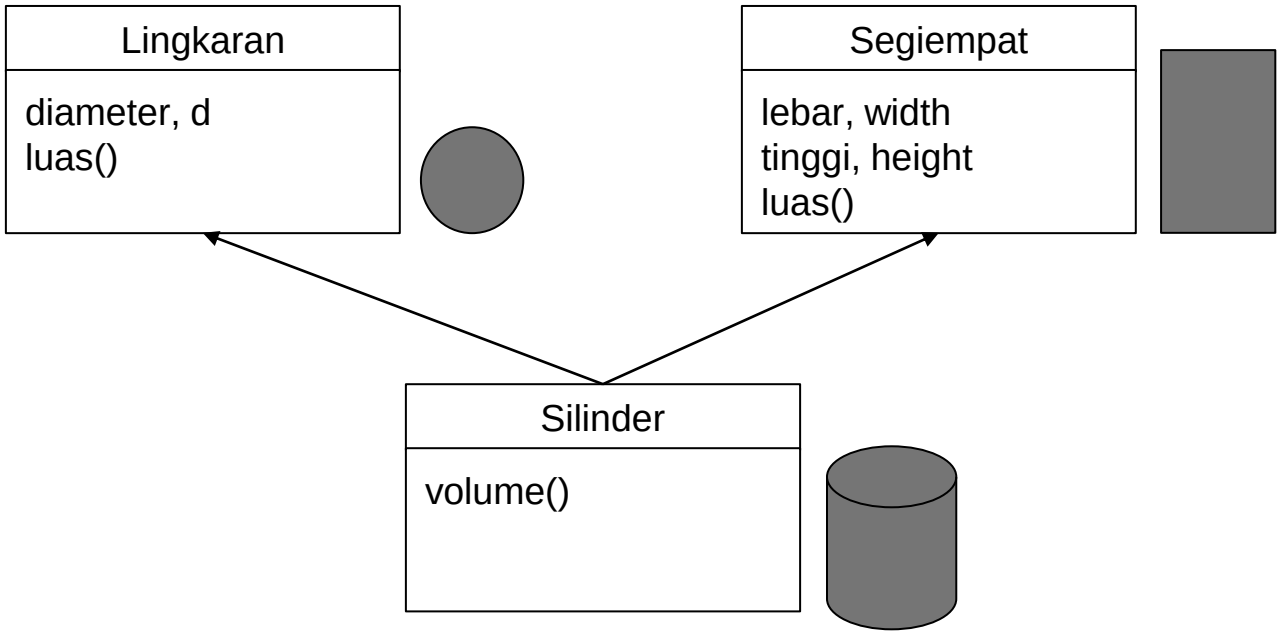
```
interface Event { public void start(); }
interface Sports { public void play(); }
interface Badminton extends Sports, Event {  public void show(); }

public class Thomascup implement Badminton {
   public static void main(String[] args) {
      Thomascup obj = new Thomascup() {
         public void start() { System.out.println("Start Event"); }
         public void play() { System.out.println("Play Sports."); }
         public void show() { System.out.println("Show Badminton."); }
      };

      obj.start();
      obj.play();
      obj.show();
   }
}
```

Latihan Kelas

Buat program Java untuk mengimplementasikan struktur berikut:



IPB University
Bogor Indonesia

KOM120C -- PEMROGRAMAN

Object Oriented Programming

- Polymorphisme pada Java

Tim Pengajar Pemrograman IPB University

Polymorphisme

Ada banyak jenis *polymorphism*(**poly** = banyak, **morph** = bentuk) dalam konteks pemrograman. Secara umum merujuk pada fenomena/kemampuan sepotong kode (variabel, fungsi, object dll) untuk mempunyai beberapa bentuk/fungsi yang berbeda, tergantung konteksnya.

Contoh bentuk *polymorphism* pada OOP: method & operator overloading pada C++

Disini, kita akan secara spesifik merujuk pada *polymorphism* yang terkait dengan method pada pewarisan obyek (*object inheritance*) dan *interface implementation* pada Java

Polymorphisme

Polymorphism dalam konteks ini berarti adanya kemampuan **obyek** untuk menjalankan sebuah methode sesuai dengan konteksnya: pemanggilan fungsi secara polimorfik. Perhatikan objek *obj* di bawah ini. Kedua pemanggilan metode f(); di bawah secara polimorfik akan memiliki bentuk yang berbeda

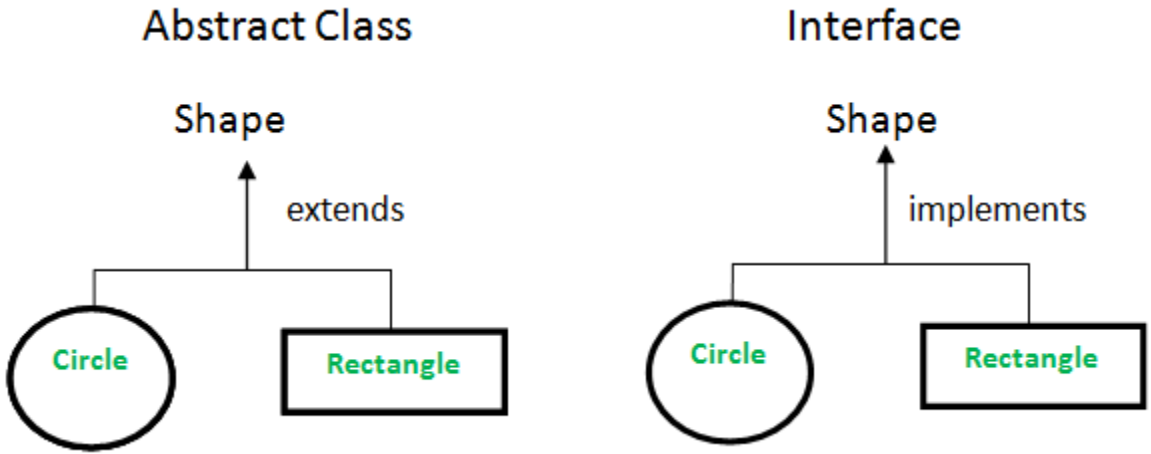
```
X obj;
obj = new Y();
obj.f()
…

obj = new Z();
obj.f()
```

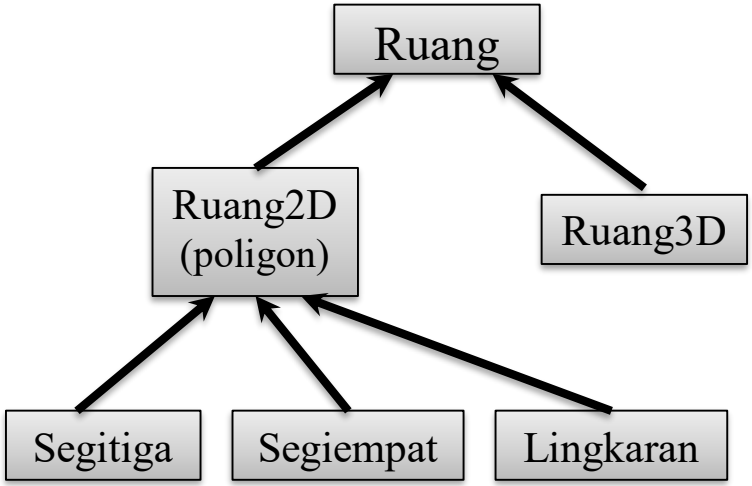
Polymorphisme

Polymorphisme pada Java memiliki dua cara yang berbeda:

- Menggunakan kelas dan metode abstrak
- Menggunakan interface

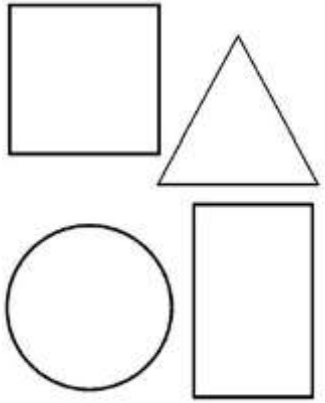


Review Abstract Class



Class Ruang mempunyai method area() yang akan di-override oleh subclasses nya.

Ruang merupakan sifat umum dari suatu bidang dua dimensi (Segitiga, Segiempat, Lingkaran) dan tiga dimensi (Bola, Kubus).



Method dalam class Ruang tidak memiliki implementasi. Class jenis ini yang disebut dengan abstract class.

Review Abstract Method

Method dalam abstract class yang tidak mempunyai implementasi dinamakan abstract method.

Only method signature, no body.

Pada C++, ditunjukkan dengan adanya method **virtual murni: virtual double area() = 0;**

Pada Java, dengan adanya deklarasi method **abstract,** Contoh:

```
public abstract class Shape {
    public abstract double area();
}
```

Polymorphism dengan Abstract Class

Sebuah **abstract class X** memiliki sebuah metode abstract **f()**. X kemudian diwariskan pada dua kelas: **Y** dan **Z**.

Jika kedua kelas turunan, **Y** dan **Z** meng-override metode **f**(), maka kita dapat mengamati adanya polimorfisme dengan membuat objek bertipe **X** namun dengan **diisi** dengan objek **Y** atau **Z** dan kemudian memanggil fungsi **f().**

Contoh Polimorfisme dengan Abstract Class

```
abstract class Shape
{
        public abstract double area();
}

class Persegi extends Shape {
        double panjang;
        double lebar;
        public Persegi(double p, double l) {
                panjang = p;
                lebar = l;
        }
        public double area (){
                return panjang * lebar;
}}
```

Contoh Polimorfisme dengan Abstract Class

```
class Lingkaran extends Shape {
        double jari;
        public static final double PI = 3.1425;
        public Lingkaran(double r)     {
                jari = r;
        }
        public double area()     {
                return PI * jari * jari;
        }
}
```

Contoh Polimorfisme dengan Abstract Class

```
public class Main{
        public static void main(String[] args) {
        Shape s;
        s = new Persegi(3, 4);
        System.out.printf("Luas persegi = %.2f\n",
s.area());
        …
        s = new Lingkaran(10);
        System.out.printf("Luas lingkaran = %.2f\n",
s.area());
        }
}
Output:
Luas persegi = 12.00
Luas lingkaran = 314.25
```

Review Interface

Interface dalam Java didefinisikan sebagai tipe abstrak untuk menentukan perilaku dari class.

Interfaces dapat memiliki abstract methods dan variables.

Interface tidak dapat dibuat obyek namun hanya dapat diimplementasi.

Sintaks:

```
interface <nameOfInterface> {
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Polymorphism dengan Interface

Dua buah kelas **A** dan **B** sama-sama mengimplementasikan sebuah *interface* **X**, yang memiliki sebuah metode **f()**.

Jika kita buat sebuah variabel bertipe *interface* **X** lalu mengisinya dengan **A** atau **B**, kemudian memanggil metode **f()** pada variabel tersebut, maka kita akan mengamati fenomena polimorfisme pada objek tersebut.

Contoh Polimorfisme dengan Interface

```
interface Shape
{
        public double area();
}

class Persegi implements Shape {
        double panjang;
        double lebar;
        public Persegi(double p, double l) {
                panjang = p;
                lebar = l;
        }
        public double area (){
                return panjang * lebar;
}}
```

Contoh Polimorfisme dengan Interface

```
class Lingkaran implements Shape {
        double jari;
        public static final double PI = 3.1425;
        public Lingkaran(double r)     {
                jari = r;
        }
        public double area()     {
                return PI * jari * jari;
        }
}
```

Contoh Polimorfisme dengan Interface

```
public class Main{
        public static void main(String[] args) {
        Shape s;
        s = new Persegi(3, 4);
        System.out.printf("Luas persegi = %.2f\n",
s.area());
        …
        s = new Lingkaran(10);
        System.out.printf("Luas lingkaran = %.2f\n",
s.area());
        }
}
Output:
Luas persegi = 12.00
Luas lingkaran = 314.25
```

Polymorphisme pada kumpulan data

Salah satu pola umum yang biasa digunakan dalam penggunaan polymorphism adalah menyimpan sekumpulan objek yang berbeda-beda, namun merupakan turunan dari kelas abstract yang sama, atau yang mengimplementasikan interface yang sama dalam sebuah **collection**, lalu mengisinya dengan objek-objek dari kelas yang berbeda tersebut.

```
ArrayList<Shape> list = new ArrayList<Shape>();
list.add(new Lingkaran(10))
list.add(new Persegi(5))
…
for(int i=0; i<n; i++) sum += list[i].area();
```

Latihan Kelas

Buat program Java dengan menggunakan polymorphism dengan abstract class atau dengan interface untuk menyelesaikan permasalahan berikut:

Diberikan masukan berupa nilai *n*, diikuti *n* buah baris, masing-masing berisi dua kemungkinan:

- baris diawali angka 1 dan kemudian terdapat dua buah bilangan desimal **a** dan **b**
- baris diawali dengan angka 2, kemudian terdapat 4 buah bilangan bulat, **p q r s** yang melambangkan dua buah pecahan **a = p/q** dan **b = r/s**

Keluarkan n buah baris yang berisi pecahan hasil penjumlahan **a + b**. Jika masukan pada sebuah baris adalah pecahan desimal, maka keluarkan jumlah dalam bentuk desimal pula (dengan 2 digit di belakang koma). Jika masukan adalah pecahan rasional, keluarkan dalam bentuk pecahan rasional pula (dalam bentuk paling sederhana).

Latihan Kelas

Contoh masukan:

```
3
1 0.3 0.25
2 1 2 1 4
2 1 3 1 1
```

Contoh keluaran:

```
0.55
3 4
4 3
```

IPB University
Bogor Indonesia

KOM120C -- PEMROGRAMAN

Object Oriented Programming

Inner Class

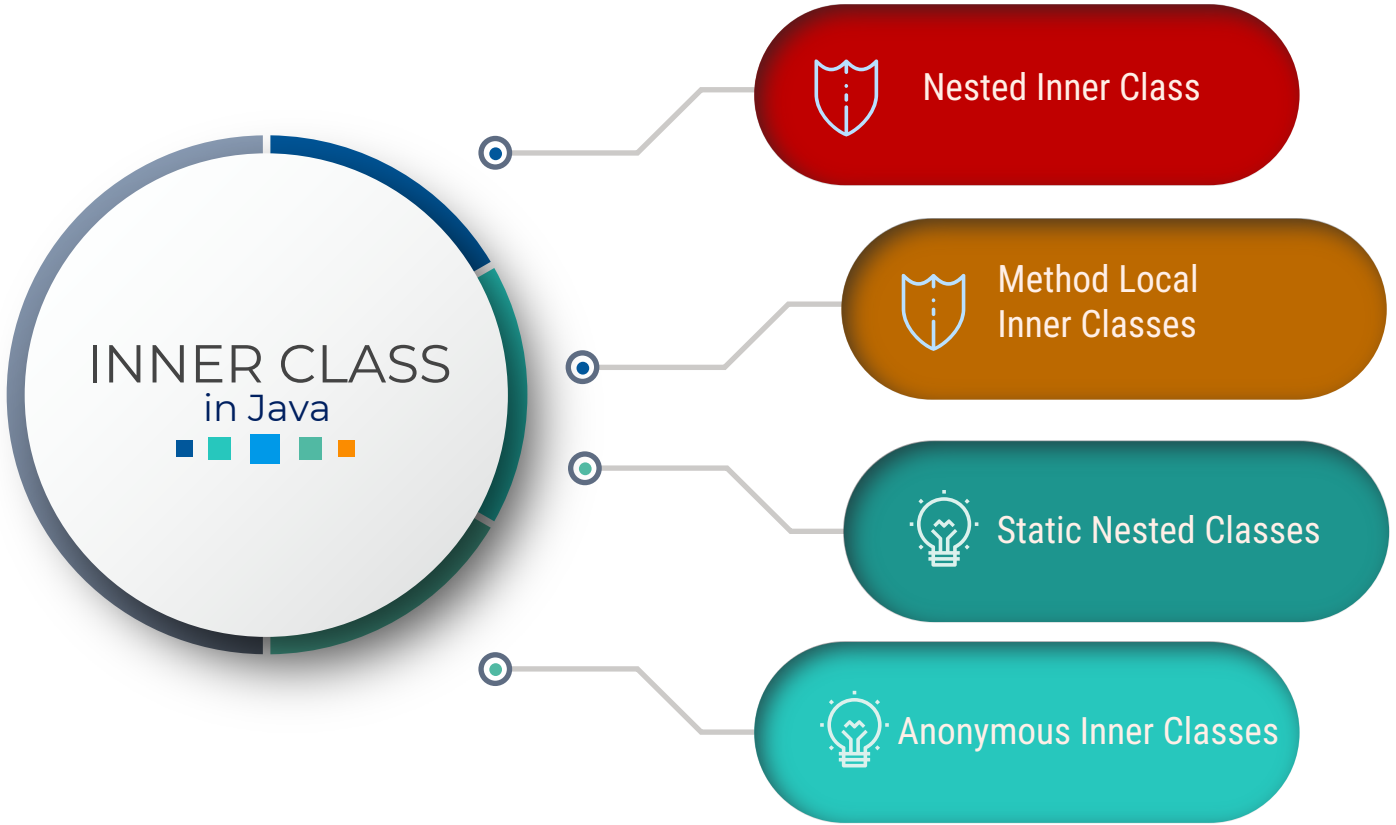
Tim Pengajar Pemrograman IPB University

Inner Class

Nested Class

Class yang dibuat di dalam suatu class atau interface.

Inner class:
- Membuat program menjadi clean and readable
- Private method dapat diakses


INNER CLASS
in Java
Nested Inner Class
Method Local
Inner Classes
Static Nested Classes
Anonymous Inner Classes

[1] Nested Inner Class

Dapat mengakses private dari outer class. Berlaku juga untuk nested inner interface.

```
class Outer {
    class Inner {
        public void show() { System.out.println("Nested class method"); }
    }
}

public class Driver {
    public static void main(String[] args) {
        Outer.Inner obj = new Outer().new Inner();
        obj.show();
    }
}
```

Output:

```
Nested class method
```

[1] Nested Inner Class :: Interface

```
class Outer {
    interface Inner { void show(); }
}

class Testing implements Outer.Inner {
    public void show() { System.out.println("Method of interface"); }
}

public class Main {
    public static void main(String[] args) {
    Outer.Inner obj;
    Testing t = new Testing();
    obj=t;
    obj.show();
    }
}
```

Output:

```
Method of interface
```

[2] Method Local Inner Classes

Inner class yang dideklarasikan di dalam method dari outer class.

```
class Outer {
    void outerMethod() {
        System.out.println("Outer Method");
        class Inner {
            void innerMethod() { System.out.println("Inner Method"); }
        }
        Inner y = new Inner();
        y.innerMethod();
    }
}

public class Main {
    public static void main(String[] args) {
        Outer obj = new Outer();
        obj.outerMethod();
    }
}
```

Output:

```
Outer Method
Inner Method
```

[2] Method Local Inner Classes

Inner class tidak dapat mengakses variabel lokal yang bukan final.

```
class Outer {
   void outerMethod() {
      final int x=98;
      System.out.println("Outer Method");
      class Inner {
         void innerMethod() {System.out.println("x = "+x); }
      }
      Inner y = new Inner();
      y.innerMethod();
   }
}
public class MethodLocalVariableDemo {
   public static void main(String[] args){
      Outer x = new Outer();
      x.outerMethod();
   }
}
```

Output:

```
Outer Method
x = 98
```

[3] Static Nested Classes

Sebenarnya secara teknis bukan inner class. Lebih mirip static member dari outer class.

```
class Outer {
    private static void outerMethod() {
        System.out.println("Outer Method");
    }
    static class Inner {
        public static void display() {
            System.out.println("Inner Class Method");
            outerMethod();
        }
    }
}
public class Main {
    public static void main(String args[]) {
        Outer.Inner.display();
    }
}
```

Output:

```
Inner Class Method
Outer Method
```

[4] Anonymous Inner Classes

Mendeklarasikan inner class tanpa nama.

```
class Demo {
    void show() { System.out.println("class Demo"); }
}

public class Main {
    static Demo d = new Demo() {
        void show() {
            super.show();
            System.out.println("class Main");
        }
    };

    public static void main(String[] args) {
        d.show();
    }
}
```

Output:

```
class Demo
class Main
```

[4] Anonymous Inner Classes :: Interface

Inner class tanpa nama mengimplementasikan interface.

```
interface Hello {
   void show();
}

public class Main {
   static Hello h = new Hello() {
      public void show() { System.out.println("Class Tanpa Nama"); }
   };

   public static void main(String[] args) {
      h.show();
   }
}
```

Output:

```
Class Tanpa Nama
```

Latihan Kelas

Silakan menggunakan konsep-konsep OOP yang pernah dipelajari

Suatu kalkulator akan memiliki nilai *currentValue* yang awalnya bernilai 0. Setelah itu, nilai tersebut akan dimodifikasi dengan operasi aritmatika yang diberikan terus menerus hingga kalkulator dimatikan. Pada soal ini, kalian diminta untuk membuat suatu interface yang mendefinisikan sifat suatu kalkulator yang dapat melakukan beberapa operasi matematika. Detail interface yang perlu kalian buat dapat dilihat pada tabel berikut:

- AritmatikaDasar - berisi fungsi-fungsi:
  - double tambah(double a, double b): menjumlahkan a dan b.
  - double kurang(double a, double b): mengurangi a dengan b.
  - double kali(double a, double, b): mengalikan a dan b;
  - double bagi(double a, double b): membagi a dengan b
- AritmatikaLanjut - berisi fungsi-fungsi:
  - double akarKuadrat(double a): mengembalikan nilai akar kuadrat dari a.
  - double pangkat(double a, double b): mengembalikan nilai a pangkat b
- KalkulatorSaintifik - merupakan kalkulator yang dapat melakukan operasi AritmatikaDasar dan AritmatikaLanjut. Juga memiliki fungsi void clear(): mengembalikan nilai currentValue pada kalkulator menjadi 0.

Latihan Kelas

Silakan menggunakan konsep-konsep OOP yang pernah dipelajari

Format Masukan

Setiap baris masukan merupakan salah satu dari tujuh kemungkinan berikut. Masukan akan diakhiri dengan simbol ~.

- `+ X` : menambah *currentValue* sebanyak X.
- `- X` : mengurangi *currentValue* sebanyak X.
- `* X` : mengalikan *currentValue* dengan X.
- `/ X` : membagi *currentValue* dengan X.
- `^ X` : memangkatkan *currentValue* sebanyak X.
- `C` : mengembalikan nilai *currentValue* menjadi 0.
- `~` : mematikan kalkulator, program berhenti

Format Keluaran

Nilai-nilai *currentValue* setiap kali suatu operasi selesai dilakukan. Cetak dengan dua angka di belakang koma dan akhiri dengan newline.

Contoh Masukan

```
+ 10
- 5
* 2
/ 5
^ 2
# 2
C
+ 5
~
```

Contoh Keluaran

```
10.00
5.00
10.00
2.00
4.00
2.00
0.00
5.00
```