Contents

[Run Program 4](#_Toc167610784)

[Data Type 4](#_Toc167610785)

[Ternary 5](#_Toc167610786)

[Class and Object 6](#_Toc167610787)

[Access instance from class 6](#_Toc167610788)

[Method 7](#_Toc167610789)

[Method Overloading 8](#_Toc167610790)

[Method Overriding 9](#_Toc167610791)

[Looping (enchance for/foreach) 10](#_Toc167610792)

[Array 10](#_Toc167610793)

[Multidimensional Array 11](#_Toc167610794)

[Array of Object 11](#_Toc167610795)

[Immutable String & Mutable String 12](#_Toc167610796)

[StringBuffer 13](#_Toc167610797)

[Static Variable 13](#_Toc167610798)

[Static block 15](#_Toc167610799)

[Call Static Block without create object 16](#_Toc167610800)

[Static Methods 17](#_Toc167610801)

[Encapsulation(private) 18](#_Toc167610802)

[Private 18](#_Toc167610803)

[Constructors 19](#_Toc167610804)

[Super() dan this() method 20](#_Toc167610805)

[Inheritance 22](#_Toc167610806)

[Multilevel Inheritance 23](#_Toc167610807)

[Multiple Inheritance 24](#_Toc167610808)

[Packages 25](#_Toc167610809)

[Access Modifier di dalam Package 26](#_Toc167610810)

[Public 26](#_Toc167610811)

[Final Keyword 28](#_Toc167610812)

[Final Variable 28](#_Toc167610813)

[Final Method 28](#_Toc167610814)

[Final Class 28](#_Toc167610815)

[TypeCasting 29](#_Toc167610816)

[Abstract Keyword 29](#_Toc167610817)

[Inner Class 30](#_Toc167610818)

[Inner Class menggunakan Static 31](#_Toc167610819)

[Annonymous Inner Class 32](#_Toc167610820)

[Abstract & Annonymous inner class 32](#_Toc167610821)

[Interface 33](#_Toc167610822)

[Enum 37](#_Toc167610823)

[Enum as Class 37](#_Toc167610824)

[Annotation 38](#_Toc167610825)

[Functional Interface Annotation 39](#_Toc167610826)

[Exception Handling 41](#_Toc167610827)

[Try catch 41](#_Toc167610828)

[Multiple catch 42](#_Toc167610829)

[Throw keyword 42](#_Toc167610830)

[Custom Exception 43](#_Toc167610831)

[Ducking Exception using throws 44](#_Toc167610832)

[Keyword Finally 45](#_Toc167610833)

[User Input 45](#_Toc167610834)

[Using BufferedReader 45](#_Toc167610835)

[Using Scanner 46](#_Toc167610836)

[Menutup Resource Input dengan try dan finally 46](#_Toc167610837)

[Thread 47](#_Toc167610838)

[Alternatif inisiasi Thread 49](#_Toc167610839)

[Dengan Annonymous Class 50](#_Toc167610840)

[Dengan Lambda Expression 50](#_Toc167610841)

[Race Condition 51](#_Toc167610842)

[Collection 52](#_Toc167610843)

[ArrayList 52](#_Toc167610844)

[Set 53](#_Toc167610845)

[Iterator 54](#_Toc167610846)

[Map 55](#_Toc167610847)

[Sorting Collection 55](#_Toc167610848)

[Custom sort by last digit, using comparator 56](#_Toc167610849)

[Sort List of class 57](#_Toc167610850)

[forEach di List 59](#_Toc167610851)

[Stream 59](#_Toc167610852)

# Run Program

Buat file bernama Hello.java

class Hello

{

    public static void main(String a[])

    {

        System.out.println("hello");

    }

}

Run di terminal

>javac Hello.java

dan

> java Hello

javac akan mengkompile java code kita menjadi binary yg extension nya .class

kemudian kita dapat melihat hasilnya dengan command java<nama class>

# Data Type

class Hello

{

    public static void main(String a[])

    {

        int num1 = 3;

        int num2 = 2;

        int result = num1 +num2;

        byte by = 127;

        short sh = 558;

        long lg = 5854l;

        float marks = 7.5f;

        double marks2 = 7.5;

        char k = 'k';

        String sentence = "hello world";

        boolean flag1 = true;

        boolean flag2 = false;

        //literals

        int num3 = 10\_000\_000;

        char c = 'a';

        c++;

    }

}

Output:

5

127

5854

=================

7.5

7.5

=================

k

hello world

=================

true

false

=================

10000000

B

# Ternary

class Hello

{

    public static void main(String a[])

    {

        int n = 5;

        int result = 0;

        result = n%2==0 ? 10 : 20;

        System.out.println(result);

    }

}

# Class and Object

class Calculator

{

    int a; //variabel

    public int add(int n1, int n2) // method return int

    {

        int r = n1 + n2;

        return r;

    }

}

public class Hello

{

    public static void main(String a[])

    {

        int num1=4;

        int num2=5;

new Calculator(); // annonymus object, cannot reuse

        Calculator calc = new Calculator(); //new object / reference

        int result = calc.add(num1, num2);

        System.out.println(result);

    }

}

## Access instance from class

class Calculator

{

   int a = 8;

    public int add(int n1, int n2)

   {

        return n1 + n2;

   }

}

public class Hello

{

    public static void main(String a[])

    {

        Calculator obj = new Calculator();

        System.out.println(obj.a); // access instance / variable of class Calculator

    }

}

# Method

class Computer

{

    public void playMusic() // void method have no return

    {

        System.out.println("Music Playing");

    }

    public String getMeAPen(int cost) //string method

    {

        return "Pen";

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Computer com = new Computer();

        com.playMusic();

        String str = com.getMeAPen(10);

        System.out.println(str);

    }

}

## Method Overloading

class Calculator

{

   public int add(int n1, int n2)

   {

        return n1 + n2;

   }

   public int add(int n1, int n2, int n3)

   {

        return n1 + n2 + n3;

   }

   public double add(double n1, int n2)

   {

        return n1+n2;

   }

}

public class Hello

{

    public static void main(String a[])

    {

        Calculator obj = new Calculator();

        int r1 = obj.add(3,4);

        int r2 = obj.add(3,4, 5);

        System.out.println(r1);

        System.out.println(r2);

    }

}

Perhatikan untuk method double add(), double add() memiliki 2 param sama dengan add(int n1, int n2), tapi tipe data param nya berbeda, yg satu int dan int, yg satu double dan int

## Method Overriding

class A

{

    public void show()

    {

        System.out.println("Show A");

    }

}

class B extends A

{

    public void show()

    {

        System.out.println("Show B");

    }

}

public class Hello

{

    public static void main(String a[])

    {

        B obj = new B();

        obj.show(); // Show B

    }

}

# Looping (enchance for/foreach)

public static void main(String a[])

    {

        int nums[] = new int[4];

        nums[0] = 7;

        nums[1] = 3;

        nums[2] = 2;

        nums[3] = 9;

        for(int value: nums)

        {

            System.out.println(value);

        }

    }

# Array

public class Hello

{

    public static void main(String a[])

    {

        int nums[] = {5,6,7};

        nums[2] = 8;

        int dnums[] = new int[4]; // new empty dynamic array of size 4

        dnums[1] = 12;

        System.out.println(nums[1]); //6

        System.out.println(nums[2]); //8

        System.out.println(dnums[0]); //0

        System.out.println(dnums[1]); //12

        //looping array

        for(int i = 0; i<3 ; i++){

            System.out.println(nums[i]);

        }

    }

}

## Multidimensional Array

public class Hello

{

    public static void main(String a[])

    {

        int nums [][] = new int [3][4]; // 3 rows, 4 colums

        for(int i=0; i<3; i++)

        {

            for(int j=0; j<4; j++)

            {

                nums[i][j] = (int)(Math.random() \* 10); // assigned random number into array

                System.out.print(nums[i][j] + " ");

            }

                System.out.println();

        }

        // Loop Array using enhance loop / foreach

        for(int n[]: nums) // assign n as array with value of nums

        {

            for(int m: n) // assign m with value of n

            {

                System.out.print(m + " ");

            }

            System.out.println();

        }

    }

}

## Array of Object

class Student

{

    int rollno;

    String name;

    int marks;

}

public class Hello

{

    public static void main(String a[])

    {

        Student s1 = new Student();

        s1.rollno = 1;

        s1.name = "Navia";

        s1.marks = 88;

        Student s2 = new Student();

        s2.rollno = 2;

        s2.name = "Candance";

        s2.marks = 78;

        Student s3 = new Student();

        s3.rollno = 3;

        s3.name = "John";

        s3.marks = 68;

        Student students[] = new Student[3];// Define 1 Array of Object with 3 length

        students[0] = s1;

        students[1] = s2;

        students[2] = s3;

        for (int i=0; i < students.length; i++)

        {

            System.out.println(students[i].name + " : " + students[i].marks );

        }

 //atau menggunakan foreach/enchance foreach

        for(Student objValue : students)

        {

            System.out.println(objValue.name + " : " + objValue.marks );

        }

    }

}

# Immutable String & Mutable String

Immutable = tidak bisa di ubah

Mutable = bisa diubah

By default String merupakan Immutable

Jika kita tetap ingin mengubah string, bisa menggunakan StringBuffer dan StringBuilder

## StringBuffer

public static void main(String a[])

    {

        StringBuffer sb =new StringBuffer("Navin");

        sb.append(" ready");

        System.out.println(sb);

    }

# Static Variable

Menjadikan instance/variable dalam scoop class, bukan lagi menjadi scoop object

Contoh scoop object

class Mobile

{

    String brand;

    int price;

    String name;

    public void show()

    {

        System.out.println(brand + " : " + price + " : " + name);

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Mobile obj1 = new Mobile();

        obj1.brand = "Apple";

        obj1.price = 1500;

        obj1.name = "Apple Phone";

        Mobile obj2 = new Mobile();

        obj2.brand = "Samsung";

        obj2.price = 1100;

        obj2.name = "Samsung Phone";

        obj1.name = "My Apple Phone";

       obj1.show(); // Apple : 1500 : My Apple Phone

        obj2.show(); // Samsung : 1100 : Samsung Phone

    }

}

Pada contoh diatas kita mengubah name dari obj1 menjadi My Apple Phone, dan yg berubah hanya name pada obj1

Contoh scoop class menggunakan static (static String name):

class Mobile

{

    String brand;

    int price;

    static String name;

    public void show()

    {

        System.out.println(brand + " : " + price + " : " + name);

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Mobile obj1 = new Mobile();

        obj1.brand = "Apple";

        obj1.price = 1500;

        obj1.name = "Apple Phone";

        Mobile obj2 = new Mobile();

        obj2.brand = "Samsung";

        obj2.price = 1100;

        obj2.name = "Samsung Phone";

        obj1.name = "My Apple Phone";

        obj1.show(); // Apple : 1500 : My Apple Phone

        obj2.show(); // Samsung : 1100 : My Apple Phone

    }

}

Disini meski kita hanya mengganti name di obj1, name di obj2 juga ikut berubah, karena name telah menjadi static variabel(class scoope). Obj1 dan obj2 merupakan objek dari class yg sama (Mobile)

Penulisan static yg benar untuk object yg ingin mengakses nya adalah <Nama class>.instance

Contoh yg benar:

class Mobile

{

    String brand;

    int price;

    static String name;

    public void show()

    {

        System.out.println(brand + " : " + price + " : " + name);

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Mobile obj1 = new Mobile();

        obj1.brand = "Apple";

        obj1.price = 1500;

        Mobile.name = "Apple Phone";

        Mobile obj2 = new Mobile();

        obj2.brand = "Samsung";

        obj2.price = 1100;

        Mobile.name = "Samsung Phone";

        Mobile.name = "My Apple Phone";

        obj1.show(); // Apple : 1500 : My Apple Phone

        obj2.show(); // Samsung : 1100 : My Apple Phone

    }

}

# Static block

class Mobile

{

    String brand;

    int price;

    static String name;

    static {

        name = "phone";

        System.out.println("this is static block");

    }

    public Mobile(){ //constructor

        brand = "";

        price = 200;

        System.out.println("in constructor");

    }

}

public class Hello

{

    public static void main(String a[])

    {

       Mobile obj1 = new Mobile();

      Mobile obj2 = new Mobile();

    }

}

Output:

this is static block

in constructor

in constructor

Static akan di jalankan saat object memanggil class dan hanya di jalankan 1x, constructor akan di panggil setelahnya dan akan di panggil ulang setiap membuat object baru

## Call Static Block without create object

class Mobile

{

    String brand;

    int price;

    static String name;

    static {

        name = "phone";

        System.out.println("this is static block");

    }

    public Mobile(){ //constructor

        brand = "";

        price = 200;

        System.out.println("in constructor");

    }

}

public class Hello

{

    public static void main(String a[]) throws ClassNotFoundException

    {

        Class.forName("Mobile");

    }

}

# Static Methods

class Mobile

{

    String brand;

    int price;

    static String name;

    public static void show1(Mobile obj)

    {

        System.out.println("this is static methods");

        System.out.println(obj.brand + " : " + name); // apple : null

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Mobile obj1 = new Mobile();

        obj1.brand = "apple";

        Mobile.show1(obj1);

    }

}

Kita hanya bisa memanggil variabel static di dalam method static, agar bisa mengkases variabel non static, dapat di lakukan dengan kirim parameter

# Encapsulation(private)

## Private

Hanya bisa di akses di class yg sama, atau menggunakan method

class Human

{

    private int age;

    private String name;

    public int getAge()

    {

        return this.age; // same class, jadi bisa akses private

    }

    public String getName()

    {

        return this.name;

    }

    public void setAge(int a)

    {

        this.age = a;

    }

    public void setName(String b)

    {

        this.name = b;

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Human obj = new Human();

        // obj.age = 30; // error krn akses private langsung

        obj.setAge(30);

        obj.setName("Navin");

        System.out.println(obj.getName() + " : " + obj.getAge());

    }

}

# Constructors

class Human

{

    private int age;

    private String name;

    public Human(){ //default constructor

        this.age = 12;

        this.name = "cancdance";

    }

    public Human(int a, String n){ //parameter constructor

        this.age = a;

        this.name = n;

    }

    public int getAge()

    {

        return this.age; // same class, jadi bisa akses private

    }

    public String getName()

    {

        return this.name;

    }

    public void setAge(int a)

    {

        this.age = a;

    }

    public void setName(String b)

    {

        this.name = b;

    }

}

public class Hello

{

    public static void main(String a[])

    {

        Human obj = new Human();

        Human obj2 = new Human(17, "diluc");

        // obj.age = 30; // error krn akses private langsung

        System.out.println(obj.getName() + " : " + obj.getAge());

        System.out.println(obj2.getName() + " : " + obj2.getAge());

    }

}

# Super() dan this() method

Pada super()

class A

{

    public A(){

        super();

        System.out.println(" this in A");

    }

    public A(int a){

        super();

        System.out.println(" this in A int " + a);

    }

}

class B extends A

{

    public B(){

        super();

        System.out.println(" this is B");

    }

    public B(int b){

        super();

        System.out.println(" this is B int");

    }

}

public class Hello

{

    public static void main(String a[])

    {

        B obj = new B(5);

    }

}

Method super() ada di dalam constructor, meski kita tidak menulisnya. Method super akan memanggil constructor dari super class nya

Pada contoh diatas Class B extends Class A, artinya B adalah SubClass, A adalah SuperClass(parent)

Saat kita membuat new object dari B, akan di jalankan terlebih dahulu constructor di super class nya (A) sesuai dengan parameter nya, baru menjalankan constructor di sub class nya(B) sesuai dengan param/tidak

Pada this()

class A

{

    public A(){

        super();

        System.out.println(" this in A");

    }

    public A(int a){

        super();

        System.out.println(" this in A int " + a);

    }

}

class B extends A

{

    public B(){

        super();

        System.out.println(" this is B");

    }

    public B(int b){

        this(); // menjalankan constructor default pada class yg sama(class B)

        System.out.println(" this is B int");

    }

}

public class Hello

{

    public static void main(String a[])

    {

        B obj = new B(5);

    }

}

Outputnya:

this in A

this is B

this is B int

# Inheritance

Hello.java

public class Hello

{

    public static void main(String a[])

    {

        AdvCalc obj = new AdvCalc();

        int r1 = obj.add(5,5); //sub class can call method on parent

        int r2 = obj.sub(10,3); //sub class can call method on parent

        int r3 = obj.multi(10,3);

        int r4 = obj.div(30,3);

        System.out.println(r1 + " " + r2 + " " + r3 + " " + r4);

    }

}

Clac.java

public class Calc

{

    public int add(int n1, int n2)

    {

        return n1 + n2;

    }

    public int sub(int n1, int n2)

    {

        return n1 - n2;

    }

}

AdvCalc.java

public class AdvCalc extends Calc

{

    public int multi(int n1, int n2)

    {

        return n1\*n2;

    }

    public int div(int n1, int n2)

    {

        return n1/n2;

    }

}

## Multilevel Inheritance

Hello.java

public class Hello

{

    public static void main(String a[])

    {

        VeryAdvCalc obj = new VeryAdvCalc();

        int r1 = obj.add(5,5);

        int r2 = obj.sub(10,3);

        int r3 = obj.multi(10,3);

        int r4 = obj.div(30,3);

        double r5 = obj.power(3,2);

        System.out.println(r1 + " " + r2 + " " + r3 + " " + r4 + " " + r5);

    }

}

Calc.java

public class Calc

{

    public int add(int n1, int n2)

    {

        return n1 + n2;

    }

    public int sub(int n1, int n2)

    {

        return n1 - n2;

    }

}

AdvCalc.java

public class AdvCalc extends Calc

{

    public int multi(int n1, int n2)

    {

        return n1\*n2;

    }

    public int div(int n1, int n2)

    {

        return n1/n2;

    }

}

VeryAdvCalc.java

public class VeryAdvCalc extends AdvCalc

{

    public double power(int n1, int n2)

    {

        return Math.pow(n1, n2);

    }

}

Disini VeryAdvCalc mewarisi AdvCalc yg mewarisi Calc sehingga kerika kita membuat object VeryAdvCalc, kita masih bisa menggunakan method yg ada di AdvCalc dan Calc

Calc > AdvCalc > VeryAdvCalc

## Multiple Inheritance

Pada java tidak bias multiple inheritance, contoh

Class A tidak dapat extends Class B dan C sekaligus

Class A extends B,C (x)

Class A extends B extends C (x)

# Packages

Tempat untuk menaruh class di rekomendasikan di dalam packages. Packages merupakan kumpulan class

package tools;

public class Calc

{

    public int add(int n1, int n2)

    {

        return n1 + n2;

    }

    public int sub(int n1, int n2)

    {

        return n1 - n2;

    }

}

Bila menggunakan Vscode dan extension java nya, maka akan muncul suggestion(icon lampu) untuk membuat folder tools di dalam project dan file Clac.java akan di pindahkan kesana.

Hello.java

import tools.AdvCalc;

import tools.Calc;

public class Hello

{

    public static void main(String a[])

    {

        Calc obj = new Calc();

        AdvCalc obj2 = new AdvCalc();

    }

}

Jika ingin memanggil class yg ada di dalam project, gunakan import <nama\_folder>.<nama\_class>

Bisa juga kita tulis import tools.\*, tanda bintang (\*) disini berarti kita menggunakan semua *file* yg ada di dalam folder tools

## Access Modifier di dalam Package

### Public

Jika kita ingin mengakses suatu variable yg class nya ada di dalam package, harus di dalam public

<root>/Hello.java

import others.\*;

public class Hello

{

    public static void main(String a[])

    {

        A obj = new A();

        System.out.println(obj.marks);

    }

}

others/A.java

package others;

public class A

{

    public int marks = 10;

}

\*bila kita mengakses dari folder/package yg sama, maka disebut akses default, maka kita bisa langsung akses tanpa menambahkan public

<root>/B.java

public class B {

    int nilai = 20;

}

<root>/Hello.java

public class Hello

{

    public static void main(String a[])

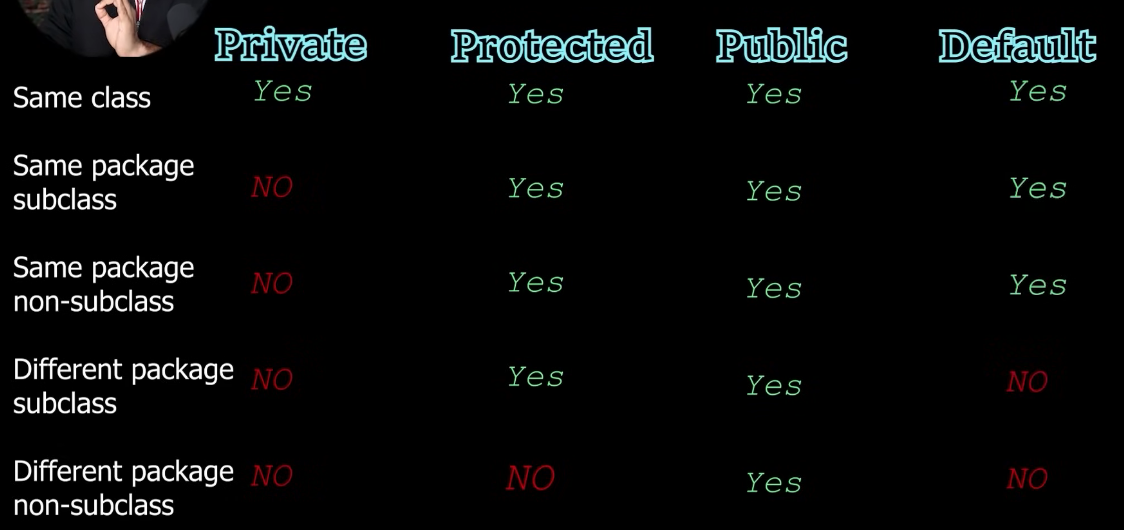
    {

        B obj = new B();

        System.out.println(obj.nilai);

    }

}



Kesimpulan

Private hanya di class yg sama

Protected hanya di class yg sama, package yg sama, dan package yg lain tpi subclass nya(child)

Public bisa di semua tmpt

Default hanya bisa di package yg sama

Polymorphism

Artinya many behavior, dibagi 2

1. Compile Time, behavior di tentukan ketika Compile, contoh overloading:

Add(int a, int b) Add(int a, int b, int c), method mana yg akan di pakai akan ditentukan ketika di compile

1. Run Time, behavior di tentukan ketika run, contoh overriding, lihat contoh di (sub menu sebelumnya). Penentuan method mana yg di pakai terjadi saaat run time

# Final Keyword

## Final Variable

public class Hello

{

    public static void main(String a[])

    {

        final int num = 8;

        // num =9; disini error krn num sudah final

        System.out.println(num);

    }

}

Secara tidak langsung kita define constant di variable num

## Final Method

class Calc

{

    public final void show()

    {

        System.out.println("Calc Show");

    }

    public void add(int a, int b)

    {

        System.out.println(a + b);

    }

}

class AdvCalc extends Calc

{

    public void show() //disini extends error krn kita meng extends method yg sdh final

    {

        System.out.println("By Jhon");

    }

}

## Final Class

final class Calc

{

    public void show()

    {

        System.out.println("Calc Show");

    }

    public void add(int a, int b)

    {

        System.out.println(a + b);

    }

}

class AdvCalc extends Calc //disini extends error krn kita meng extends clas Clac yg sdh final

{

}

# TypeCasting

Typecasting:

Mengganti type data

public class Hello

{

    public static void main(String a[])

    {

        double num = 5.8;

        int i = (int)num;

        System.out.println(i); // hasilnya 5

    }

}

# Abstract Keyword

Abstract Method adalah method yg hanya kita define, tanpa ada isi code/blm di implement. Method abstract hanya bias di define di class abstract.

Yang bertugas meng implement method ini adalah class yg mengextend abstract class

Tidak bisa create object dari abstract class, jika ingin mengakses, extend ke class lain

abstract class Car

{

    abstract public void drive();

    public void playMusic()

    {

        System.out.println("Play Music");

    }

}

class WagonR extends Car

{

    public void drive()

    {

        System.out.println("Drive a car");

    }

}

public class Hello

{

    public static void main(String a[])

    {

       WagonR obj = new WagonR();

       obj.drive();

       obj.playMusic();

    }

}

# Inner Class

Didalam class ada class lain

class A

{

    int age;

    public void show()

    {

        System.out.println("show A");

    }

    class B

    {

        public void config()

        {

            System.out.println("config");

        }

    }

}

public class Hello

{

    public static void main(String a[])

    {

       A objA = new A();

       objA.show();

       A.B objB = objA.new B();

       objB.config();

    }

}

Pada contoh diatas Class B didefine di dalam Class A, cara membuat objeck nya A.B ojbB = objA.new B() dimana kita menggunakan object dari class A (objA) untuk membuat object Class B

## Inner Class menggunakan Static

class A

{

    int age;

    public void show()

    {

        System.out.println("show A");

    }

    static class B

    {

        public void config()

        {

            System.out.println("config");

        }

    }

}

public class Hello

{

    public static void main(String a[])

    {

       A objA = new A();

       objA.show();

       A.B objB = new A.B();

       objB.config();

    }

}

Disini untuk membuat object class B kita bisa langsung menggunakan new A.B(). Kita hanya bisa menggunakan static untuk inner class(Class B), dan tidak bisa menggunakan static untuk outter class (Class A)

## Annonymous Inner Class

class A

{

    public void show()

    {

        System.out.println("show A");

    }

}

public class Hello

{

    public static void main(String a[])

    {

       A obj = new A()  // annonymous inner class

       {

            public void show()

            {

                System.out.println("new Show ");

            }

       };

       obj.show(); // new Show

    }

}

Perhatikan di annonymous class kita tidak mendefine nama class nya, sehingga disebut annonymous inner class

## Abstract & Annonymous inner class

abstract class A

{

    public abstract void show();

}

public class Hello

{

    public static void main(String a[])

    {

       A obj = new A()  // annonymous inner class

       {

            public void show()

            {

                System.out.println("new Show ");

            }

       };

       obj.show(); // new Show

    }

}

# Interface

Pada interface kita hanya mendefine method & variabel apa saja yg akan di gunakan pada class. Kita menggunakan keyword implemets untuk menggunakan interface. Tidak bisa langsung membuat object dari interface

interface A // only show the design

{

    int age = 25; // by default variable in interface = final and static

    String area = "Jakarta";

    void show(); // by default method in interface = public abstract

    void config();

}

class B implements A

{

    public void show()

    {

        System.out.println("show");

    }

    public void config()

    {

        System.out.println("config");

    }

}

public class Hello

{

    public static void main(String a[])

    {

      A obj;

      obj = new B();

      obj.show();

      obj.config();

      System.out.println(A.area);

    }

}

Class dapat meng-implements lebih dari 1 interface

interface A // only show the design

{

    int age = 25; // by default variable in interface = final and static

    String area = "Jakarta";

    void show(); // by default method in interface = public abstract

    void config();

}

interface X

{

    void run();

}

class B implements A,X

{

    public void show()

    {

        System.out.println("show");

    }

    public void config()

    {

        System.out.println("config");

    }

    public void run()

    {

        System.out.println("running");

    }

}

Class -> Class = extends

Class-> interface = implements

Interface -> interface = extends

interface A // only show the design

{

    int age = 25; // by default variable in interface = final and static

    String area = "Jakarta";

    void show(); // by default method in interface = public abstract

    void config();

}

interface X

{

    void run();

}

interface Y extends X

{

}

class B implements A,Y

{

Full example:

interface A // only show the design

{

    int age = 25; // by default variable in interface = final and static

    String area = "Jakarta";

    void show(); // by default method in interface = public abstract

    void config();

}

interface X

{

    void run();

}

interface Y extends X

{

}

class B implements A,Y

{

    public void show()

    {

        System.out.println("show");

    }

    public void config()

    {

        System.out.println("config");

    }

    public void run()

    {

        System.out.println("running");

    }

}

public class Hello

{

    public static void main(String a[])

    {

      A obj;

      obj = new B();

      obj.show();

      obj.config();

      System.out.println(A.area);

      X obj1;

      obj1 = new B();

      obj1.run();

    }

}

# Enum

enum Status{

    Running, Failed, Pending, Success;

}

public class Hello

{

    public static void main(String a[])

    {

      Status s = Status.Running;

      Status[] allStatus = Status.values();

      System.out.println(s); // Running

      for(Status v: allStatus)

      {

        System.out.println(v);

      }

    }

}

## Enum as Class

Sebagai class kita bisa membuat method dan variable di dalam blok kode Enum

enum Laptop{

    Asus(200), Lenovo(210), Macbook(300), Acer(220), XPS;

    private int price;

    private Laptop() { // constructor, untuk XPS krn tdk ada param, akan memakai ini

        price = 250;

    }

    private Laptop(int price) { //constructor dgn param

        this.price = price;

    }

    public int getPrice() {

        return price;

    }

    public void setPrice(int price) {

        this.price = price;

    }

}

public class Hello

{

    public static void main(String a[])

    {

      Laptop lap = Laptop.Macbook; //use enum Macbook

      System.out.println(lap + " : " + lap.getPrice()); //Macbook : 300

      for(Laptop val : Laptop.values()){//print all enum and price

        System.out.println(val + " : " + val.getPrice());

      }

    }

}

# Annotation

class A

{

    public void show()

    {

        System.out.println("in A Show");

    }

}

class B extends A

{

    @Override //annotation

    public void shows()

    {

        System.out.println("in B show");

    }

}

public class Hello

{

    public static void main(String a[])

    {

      B obj = new B();

      obj.show();

    }

}

Pada contoh diatas kita menggunakan annotation @override agar compiler membantu kita untuk mengecek, ada tidak method “shows” di class A. Diatas kita sengaja memberi nama yg berbeda, agar saat di VS Code ada warning yg menyatakan tidak ada method “shows”. Annotation membantu kita mencegah terjadi nya bugs.

Contoh lain nya, annotation dalam level Class

@Deprecated

class A

{

    public void show()

    {

        System.out.println("in A Show");

    }

}

@Deprecated, artinya memberi tanda pada class bahwa kita bisa menggunakan class ini, tetapi sebaiknya jangan digunakan karena akan segera di hapus saat ada update

## Functional Interface Annotation

Interface yg hanya memiliki 1 method

@FunctionalInterface

interface A

{

    void show();

    void read();

}

Bia kita paksa membuat 2 method maka akan muncul error/warning

Functional Interface with Lambda Expression

@FunctionalInterface

interface A

{

    void show();

}

public class Hello

{

    public static void main(String a[])

    {

      A obj = new A()

      {

        public void show()

        {

            System.out.println("in show");

        }

      };

      obj.show();

    }

}

Pada contoh diatas, kita mengetahui Functional Interface hanya memiliki 1 method, dan jika kita ingin secara anonymous class, kita tidak perlu menuliskan ulang “public void show()”

public static void main(String a[])

    {

      A obj = ()-> //lambda expression

      {

            System.out.println("in show");

      };

      // bila hanya 1 baris code

      A obj = ()-> System.out.println("in show");

      obj.show();

    }

Dengan lambda expression, kita dapat menghemat penulisan

 // bila hanya 1 baris code

      A obj = ()-> System.out.println("in show");

      obj.show();

Contoh dengan parameter

@FunctionalInterface

interface A

{

    void show(int i, int j) ;

}

public class Hello

{

    public static void main(String a[])

    {

      // dengan param

      A obj = (i, j)-> System.out.println("in show " + i + j);

// bila pada interface hanya pakai 1 param kita bisa tulis:

A obj = I -> System.out.println("in show " + i );

      obj.show(5, 8 );

    }

}

Dengan return value

@FunctionalInterface

interface A

{

    int add(int i, int j) ;

}

public class Hello

{

    public static void main(String a[])

    {

      A obj = (i, j)->  i+j; // i+j disini sama dengan return i+j

      obj.add(5, 8 );

    }

}

# Exception Handling

## Try catch

public static void main(String a[])

    {

      int i =0;

      int j = 0;

      try {

        j = 16/i;

      } catch (Exception e) {

        System.out.println("some thing went wrong");

      }

      System.out.println(j);

    }

## Multiple catch

public class Hello

{

    public static void main(String a[])

    {

      int i =2;

      int j = 0;

      int nums[] = new int[5];// new array size of 5

      String str = null;

      try {

        j = 18/i;

        System.out.println(str.length());

        // System.out.println(nums[5]); //print index 5

      } catch (ArithmeticException e) {

        System.out.println("Cannot devided by 0. ");

      } catch(ArrayIndexOutOfBoundsException e) {

        System.out.println("Out of index limit");

      } catch(Exception e) { // parent exception, taruh di paling bawah

        System.out.println("Some thing went wrong. " + e);

      }

      System.out.println(j);

    }

}

## Throw keyword

public class Hello

{

    public static void main(String a[])

    {

      int i =20;

      int j = 0;

      try {

        j = 18/i; // hasilnya 0

        if (j==0)

          throw new ArithmeticException();

      } catch (ArithmeticException e) {

        j = 18 / 1; // handle exception

        System.out.println("This is default output ");

      } catch(Exception e) {

        System.out.println("Some thing went wrong. " + e);

      }

      System.out.println(j);

    }

}

Pada contoh diatas kita try 18/i atau 18/20, karena int, maka hasilnya 0, if hasil nya 0 kita throw ArithmaticException. Bisa di bilang throw merupakan cara agar kita bisa memanggil isi blok catch meski tidak error.

Hasil output code diatas:

This is default output

18

## Custom Exception

class NavinException extends Exception

{

  public NavinException(String string)

  {

    super(string);

  }

}

public class Hello

{

    public static void main(String a[])

    {

      int i =20;

      int j = 0;

      try {

        j = 18/i; // hasilnya 0

        if (j==0)

          throw new NavinException("i dont want to print zero");

      } catch (NavinException e) {

        j = 18 / 1; // handle exception

        System.out.println("This is default output " + e);

      } catch(Exception e) {

        System.out.println("Some thing went wrong. " + e);

      }

      System.out.println(j);

    }

}

Disini kita membuat Exception sendiri bernama NavinException, pertama kita buat dulu class NavinException dengan method NavinException juga, jika ingin mereturn param string gunakan super()

Outputnya:

This is default output NavinException: i dont want to print zero

18

## Ducking Exception using throws

class A

{

  public void show() throws ClassNotFoundException

  {

    Class.forName("Calc");

  }

}

public class Hello

{

    public static void main(String a[])

    {

      A obj = new A();

      try {

        obj.show();

      } catch (ClassNotFoundException e) {

        System.out.println(e);

      }

    }

}

Pada contoh di atas kita mencoba memanggil method show() yg ada di class A. Method show berusaha memanggil class bernama Calc, karena tidak ada class Calc, maka akan error. Perhatikan pada method show() kita throws Excpetion bernama ‘ClassNotFoundException’ artinya jika terjadi error pada method ini, yg akan me-try catch adalah class yg memanggil nya, yaitu class main. Oleh karena nya kita membuat try catch di bagian class main

## Keyword Finally

public class Hello

{

    public static void main(String a[])

    {

      int i =0;

      int j =18;

      try

      {

        j = j/i;

        System.out.println("Hasil j = " + j);

      }

      catch(Exception e)

      {

        System.out.println("Something went wrong");

      }

      finally

      {// block ini akan tetap di panggil, meski masuk/tdk msk catch

        System.out.println("Last statement");

      }

    }

}

Blok data finally akan tetap tanpa memedulikan code kita error(msk catch) atau tidak.

# User Input

## Using BufferedReader

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Hello

{

    public static void main(String a[]) throws IOException

    {

      System.out.println("Enter a number");

      InputStreamReader in = new InputStreamReader(System.in);

      BufferedReader bf = new BufferedReader(in);

      int num = Integer.parseInt(bf.readLine());

      System.out.println(num);

      bf.close();

    }

}

## Using Scanner

import java.util.Scanner;

public class Hello

{

    public static void main(String a[])

    {

      System.out.println("Enter a number");

      Scanner sc = new Scanner(System.in);

      int num = sc.nextInt();

      System.out.println(num);

    }

}

## Menutup Resource Input dengan try dan finally

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Hello

{

    public static void main(String a[]) throws NumberFormatException, IOException

    {

      int num = 0;

      BufferedReader br = null;

      BufferedReader br2 = null;

      try

      {

        // InputStreamReader in = new InputStreamReader(System.in);

        // br = new BufferedReader(in);

        // atau

        br2 = new BufferedReader(new InputStreamReader(System.in));

        num = Integer.parseInt(br2.readLine());

        System.out.println("Hasil  = " + num);

      }

      finally

      {

        br2.close();

      }

    }

}

Alternatif

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Hello

{

    public static void main(String a[]) throws NumberFormatException, IOException

    {

      int num = 0;

      try(BufferedReader br = new BufferedReader(new InputStreamReader(System.in)))

      {

        num = Integer.parseInt(br.readLine());

        System.out.println("Hasil  = " + num);

      }

    }

}

Penulisan lebih singkat. Kita mendeklarasikan BufferedReader di dalam try, jadi bisa otomatis langsung di close tanpa membuat blok data finally

# Thread

Memungkin kan kita menjalankan proses secara pararel

Di dalam class Thread, harus ada method bernama run()

Cara memanggil threads <obj>.start();

class A extends Thread

{

  public void run()

  {

    for(int i = 0; i <10; i++)

    {

      System.out.println("hi");

    }

  }

}

class B extends Thread

{

  public void run()

  {

    for(int i = 0; i <10; i++)

    {

      System.out.println("hello");

    }

  }

}

public class Hello

{

    public static void main(String a[])

    {

      A obj1 = new A();

      B obj2 = new B();

obj2.setPriority(Thread.MAX\_PRIORITY);//0 -10

      obj1.start();

      obj2.start();

    }

}

Bila kita run akan tampak seperti print hi dlu baru hello, karena pc kita cepat memproses nya, di belakang layar, kedua nya jalan bersamaan

Sleep, digunakan untuk memberi jeda

  public void run()

  {

    for(int i = 0; i <10; i++)

    {

      System.out.println("hi");

      try {

        Thread.sleep(10); //pause 10ms

      } catch (InterruptedException e) {

        e.printStackTrace();

      }

    }

  }

## Alternatif inisiasi Thread

Kita dapat menggunakan thread dengan meng-implements Runnanle, sehingga kalai class nya mau kita extends masih bisa.

Untuk start() nya kita ganti dengan

Runnable obj1 = new A();

      Runnable obj2 = new B();

      Thread t1 = new Thread(obj1);

      Thread t2 = new Thread(obj2);

      t1.start();

      t2.start();

contoh lengkap:

class A implements Runnable

{

  public void run()

  {

    for(int i = 0; i <10; i++)

    {

      System.out.println("hi");

      try {

        Thread.sleep(10); //pause 10ms

      } catch (InterruptedException e) {

        e.printStackTrace();

      }

    }

  }

}

class B implements Runnable

{

  public void run()

  {

    for(int i = 0; i <10; i++)

    {

      System.out.println("hello");

    }

  }

}

public class Hello

{

    public static void main(String a[])

    {

      Runnable obj1 = new A();

      Runnable obj2 = new B();

      Thread t1 = new Thread(obj1);

      Thread t2 = new Thread(obj2);

      t1.start();

      t2.start();

    }

}

## Dengan Annonymous Class

Runnable obj1 = new Runnable() {

        public void run()

        {

          for(int i = 0; i <10; i++)

          {

            System.out.println("hi");

            try {

              Thread.sleep(10); //pause 10ms

            } catch (InterruptedException e) {

              e.printStackTrace();

            }

          }

        }

      };

## Dengan Lambda Expression

Runnable obj1 = () ->

        {

          for(int i = 0; i <10; i++)

          {

            System.out.println("hi");

            try {

              Thread.sleep(10); //pause 10ms

            } catch (InterruptedException e) {

              e.printStackTrace();

            }

          }

        };

## Race Condition

Misal kita punya variable count, variable count digunakan bersama oleh 2 threads yg berbeda. Maka akan ada kemungkinan hasilhnya tidak sesuai harapan, oleh karena itu kita bisa menggunaan .join() agar hasil komputasi thread1 dan thread2 di gabungkan. Bila menggunakan join() jgn lupa throw exception InterruptedException

class Counter

{

  int count;

  public void increment()

  {

    count++;

  }

}

public class Hello

{

    public static void main(String a[]) throws InterruptedException

    {

      Counter c = new Counter();

      Runnable obj1 = () ->

      {

        for(int i = 1; i <1000; i++)

        {

          c.increment();

        }

      };

        Runnable obj2 = () ->

        {

          for(int i = 1; i <1000; i++)

          {

            c.increment();

          }

        };

      Thread t1 = new Thread(obj1);

      Thread t2 = new Thread(obj2);

      t1.start();

      t2.start();

      t1.join();

      t2.join();

      System.out.println("akhir = " + c.count);

    }

}

\*Menggunakan 1variable yg sama di dalam 2 thread yg berbeda tidak di anjurkan karena hasilnya tidak akan konsisten

Kita dapat menggunakan method sysnchronized agar method hanya dijalankan 1 x di suatu waktu. Bila thread1 menggunakan increment(), Thread2 harus menunggu

public synchronized void increment()

  {

    count++;

  }

# Collection

## ArrayList

import java.util.ArrayList;

import java.util.Collection;

public class Hello

{

    public static void main(String a[])

    {

      Collection<Integer> nums = new ArrayList<Integer>();

      nums.add(6);

      nums.add(5);

      nums.add(2);

      for(int n: nums)

      {

        System.out.println(n);

      }

    }

}

Array List tidak ada index nya. Jika ingin melakukan operasi dengan index, sebaiknya menggunakan List

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

public class Hello

{

    public static void main(String a[])

    {

      List<Integer> nums = new ArrayList<Integer>();

      nums.add(6);

      nums.add(5);

      nums.add(2);

      System.out.println(nums.get(1)); //index ke 1

Cara lain insert ArrayList, gunakan asList

List<Integer> nums  = Arrays.asList(5,7,9,8,1);

## Set

Support unique values, nilai yg sama tidak ditampilkan dan tidak di sort (ditampilkan secara acak)

import java.util.Collection;

import java.util.HashSet;

import java.util.Set;

public class Hello

{

    public static void main(String a[])

    {

      Set<Integer> nums = new HashSet<Integer>();

      nums.add(6);

      nums.add(5);

      nums.add(2);

      nums.add(5);

      for(int n: nums)

      {

        System.out.println(n);

      }

    }

}

Outputnya:

2

5

6

Jika ingin hasilnya di sort gunakan TreeSet

import java.util.Collection;

import java.util.HashSet;

import java.util.Set;

import java.util.TreeSet;

public class Hello

{

    public static void main(String a[])

    {

      Set<Integer> nums = new TreeSet<Integer>();

      nums.add(6);

      nums.add(5);

      nums.add(2);

      nums.add(5);

      for(int n: nums)

      {

        System.out.println(n);

      }

    }

}

## Iterator

Alternatif looping collection

import java.util.Collection;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

import java.util.TreeSet;

public class Hello

{

    public static void main(String a[])

    {

      Collection<Integer> nums = new TreeSet<Integer>();

      nums.add(6);

      nums.add(5);

      nums.add(2);

      nums.add(5);

      Iterator<Integer> val = nums.iterator();

      while(val.hasNext())

        System.out.println(val.next());

    }

}

## Map

Map adalah kumpuan dari key beserta value nya

import java.util.HashMap;

import java.util.Map;

public class Hello

{

    public static void main(String a[])

    {

      Map<String, Integer> students = new HashMap<>();

      //<type key, type value>

      students.put("Navin", 56); //insert into Map

      students.put("Kiran", 50);

      students.put("Kaveh", 67);

      students.put("Cyno", 70);

      students.put("Kiran", 80); //key Kiran akan di overwrite menjadi 80

      System.out.println(students); // {Navin=56, Kiran=80, Cyno=70, Kaveh=67}

      System.out.println(students.get("Kaveh")); //67

      System.out.println(students.keySet()); // get key [Navin, Kiran, Cyno, Kaveh]

      //print all

      for(String key: students.keySet())

      {

        System.out.println(key + " : " + students.get(key));

      }

    }

}

## Sorting Collection

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class Hello

{

    public static void main(String a[])

    {

      List<Integer> nums  =new ArrayList<>();

      nums.add(53);

      nums.add(31);

      nums.add(72);

      nums.add(99);

      //default sort

      Collections.sort(nums);

      System.out.println(nums);

    }

}

### Custom sort by last digit, using comparator

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class Hello

{

    public static void main(String a[])

    {

      Comparator<Integer> com = new Comparator<Integer>()

      {

          public int compare(Integer i, Integer j)

          {

            if(i%10 > j%10)

              return 1; //swap

            else

              return -1; //dont'swap

          }

      };

      List<Integer> nums  =new ArrayList<>();

      nums.add(53);

      nums.add(31);

      nums.add(72);

      nums.add(99);

      //custom sort by last digit, using comparator

      Collections.sort(nums, com);

      System.out.println(nums); //[31, 72, 53, 99]

    }

}

### Sort List of class

Sort by student age menggunakan Comparator

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Student

{

    int age;

    String name;

    public Student(int age, String name) {

      this.age = age;

      this.name = name;

    }

    public String toString() {

      return "Student [age=" + age + ", name=" + name + "]";

    }

}

public class Hello

{

    public static void main(String a[])

    {

      Comparator<Student> com = new Comparator<Student>()

      {

          public int compare(Student i, Student j)

          {

            if(i.age > j.age)

              return 1; //swap

            else

              return -1; //dont'swap

          }

      };

//Alternatif with Lambda Expression

      Comparator<Student> com = (i, j) ->i.age > j.age ? 1 : -1;

      List<Student> studs  =new ArrayList<>();

      studs.add(new Student(17, "Navin"));

      studs.add(new Student(15, "Cyno"));

      studs.add(new Student(23, "Karin"));

      studs.add(new Student(20, "Paul"));

      Collections.sort(studs, com);

      for(Student s: studs)

        System.out.println(s);

    }

}

Alternatif, implements Comparable di class yg mau di sort dan tambahkan menthod compareTo()

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Student implements Comparable<Student>

{

    int age;

    String name;

    public Student(int age, String name) {

      this.age = age;

      this.name = name;

    }

    public String toString() {

      return "Student [age=" + age + ", name=" + name + "]";

    }

    public int compareTo(Student that){

      if(this.age > that.age)

          return 1; //swap

        else

          return -1; //dont'swap

    }

}

public class Hello

{

    public static void main(String a[])

    {

      List<Student> studs  =new ArrayList<>();

      studs.add(new Student(17, "Navin"));

      studs.add(new Student(15, "Cyno"));

      studs.add(new Student(23, "Karin"));

      studs.add(new Student(20, "Paul"));

      Collections.sort(studs);

      for(Student s: studs)

        System.out.println(s);

    }

}

## forEach di List

 public static void main(String a[])

    {

      List<Integer> nums  = Arrays.asList(5,7,9,8,2);

      int sum = 0;

      nums.forEach(n -> System.out.println(n));

## Stream

Melakukan operasi untuk Collection/List

Contoh hanya menampilkan bilangan genap dan di kali 2 dan di sum hasilnya

import java.util.Arrays;

import java.util.List;

import java.util.stream.Stream;

public class Hello

{

public static void main(String a[])

    {

      List<Integer> nums  = Arrays.asList(5,7,9,8,2);

      Stream<Integer>s1 = nums.stream();

      Stream<Integer>s2 = s1.filter(n->n%2 ==0); //only even number

      Stream<Integer>s3 = s2.map(n -> n\*2); // double the value

      int result = s3.reduce(0, (c,e)-> c+e); // hasil sum dari s3, 0 adalah initial value

      //penulisan lain, yg lebih singkat

      int result2 = nums.stream()

                        .filter(n->n%2 ==0)

                        .map(n -> n\*2)

                        .reduce(0, (c,e)-> c+e);

      System.out.println(result2);

## Sorted Stream

List<Integer> nums  = Arrays.asList(5,7,9,8,2);

Stream<Integer> sortedValues = nums.stream()

                                    .filter(n -> n%2 ==0)

                                    .sorted();

sortedValues.forEach(n -> System.out.println(n));