|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LAPORAN PRAKTIKUM  PEMROGRAMAN OBJEK (OOP)     |  |  |  |  | | --- | --- | --- | --- | | NAMA PRAKTIKAN | NPM MAHASISWA | TANGGAL KUMPUL | TANDA TANGAN  PRAKTIKAN | | ASY’ARI FANDI AHMAD | 201869040009 | 31 Desember 2019 |  |      |  |  |  |  | | --- | --- | --- | --- | | NAMA PENILAI | TANGGAK KOREKSI | NILAI | TANDA TANGAN  DOSEN | | M. IMRON ROSADI, S.kom., M.kom. |  |  |  |     FAKULTAS TEKNIK INFORMATIKA  UNIVERSITAS YUDHARTA  PASURUAN  2019 |

# KATA PENGANTAR

Segala puji bagi Allah SWT yang telah memberikan kesehatan, kekuatan, kesempatan serta limpahan rahmat dan karunianya sehingga penulis dapat menyelesaikan laporan praktikum pada mata kuliah Pemrograman Objek (OOP) ini dengan baik. Penulisan laporan ini dibuat agar dapat memenuhi sebagi salah satu syarat untuk mengikuti ujian semester. Dengan sepenuh hati penulis menyadari bahwa tersusunnya tugas ini berkat bantuan dan bimbingan dari semua pihak baik berupa material, spiritual, maupun informasi. Oleh karena itu dalam kesempatan ini penulis tidak lupa mengucapkan terima kasih sebanyak- banyaknya kepada :

1. Kepada Tuhan Yang Maha Esa yaitu Allah Swt, karena dengan anugerahnya penulis dapat menyelesaikan laporan praktikum ini dengan baik.
2. Orangtua saya yang telah memberikan semangat dan dukungan sehingga laporan praktikum ini dapat terselesaikan dengan baik serta mendapatkan nilai yang diinginkan
3. M. Imron Rosadi, S.Kom., M.kom. , selaku dosen Pemrograman objek (OOP).
4. Semua pihak yang telah membantu dalam penyusunan laporan ini. Akhirnya penulis mohon maaf.

Apabila ada kekurangan atau kesalahan dalam penyusunan laporan praktikum ini. Semoga laporan

praktikum ini dapat bermanfaat bagi penulis, maupun pembacanya

Pasuruan,31 Desember 2019

Penulis

Asy’ari Fandi Ahmad

# DAFTAR ISI

[KATA PENGANTAR 2](#_Toc28712076)

[DAFTAR ISI 3](#_Toc28712077)

[BAB I 4](#_Toc28712078)

[PENDAHULUAN 4](#_Toc28712079)

[1.1 Latar Belakang 4](#_Toc28712080)

[1.2 Tujuan 4](#_Toc28712081)

[1.3 Manfaat 5](#_Toc28712082)

[BAB II 6](#_Toc28712083)

[PEMBAHASAN 6](#_Toc28712084)

[2.1 Analisa Hasil Percobaan Java OOP 6](#_Toc28712085)

[**2.2 Analisa Hasil Percobaan** **Java Class & Object** 7](#_Toc28712086)

[2.3 Analisa Hasil Percobaan Java Class Attributes 8](#_Toc28712087)

[**2.4 Analisa Hasil Percobaan** **Java Method** 11](#_Toc28712088)

[2.5 Analisa Hasil Percobaan Java Constructor 15](#_Toc28712089)

[2.6 Analisa Hasil Percobaan Java Modifier 16](#_Toc28712090)

[6.1 Access Modifiers 17](#_Toc28712091)

[6.2 Non-Access Modifiers 19](#_Toc28712092)

[2.7 Analisa Hasil Percobaan Java Enkapsulasi 20](#_Toc28712093)

[2.8 Analisa Hasil Percobaan Java Paket & API 23](#_Toc28712094)

[2.9 Analisa Hasil Percobaan Java Inheritance/ Pewarisan (Subclass dan Superclass) 25](#_Toc28712095)

[2.10 Analisa Hasil Percobaan Java Polimorfisme 34](#_Toc28712096)

[2.11 Analisa Hasil Percobaan Java Inner Class 35](#_Toc28712097)

[2.12 Analisa Hasil Percobaan Java Abstrak 37](#_Toc28712098)

[2.13 Analisa Hasil Percobaan Java Interface 39](#_Toc28712099)

[2.14 Analisa Hasil Percobaan Java Enums 41](#_Toc28712100)

[2.15 Analisa Hasil Percobaan Java User Input (Scanner) 43](#_Toc28712101)

[2.16 Analisa Hasil Percobaan Java Date and Time 45](#_Toc28712102)

[2.17 Analisa Hasil Percobaan Java ArrayList 49](#_Toc28712103)

[2.18 Analisa Hasil Percobaan Java HashMap 54](#_Toc28712104)

[2.19 Analisa Hasil Percobaan Java Wrapper Classes 59](#_Toc28712105)

[2.19 Analisa Hasil Percobaan Java Wrapper Classes 61](#_Toc28712106)

[Uji Diri Anda Dengan Latihan 64](#_Toc28712107)

[KESIMPULAN 65](#_Toc28712108)

[SARAN 65](#_Toc28712109)

**BAB I**

**PENDAHULUAN**

## 1.1 Latar Belakang

Pemrograman Berorientasi Objek (Object Oriented Programming atau OOP) merupakan [paradigma pemrograman](http://id.wikipedia.org/wiki/Paradigma_pemrograman" \o "Paradigma pemrograman) yang berorientasikan kepada objek. Objek adalah struktur data yang terdiri dari [bidang data](http://translate.googleusercontent.com/translate_c?hl=id&prev=/search%3Fq%3DObject%2BOriented%2BProgramming%26hl%3Did%26biw%3D1362%26bih%3D505%26prmd%3Divnsb&rurl=translate.google.com&sl=en&u=http://en.wikipedia.org/wiki/Field_%28computer_science%29&usg=ALkJrhjRS16JOPdmNPay5L2gdgLgv70i-w" \o "Lapangan (ilmu komputer)) dan [metode](http://translate.googleusercontent.com/translate_c?hl=id&prev=/search%3Fq%3DObject%2BOriented%2BProgramming%26hl%3Did%26biw%3D1362%26bih%3D505%26prmd%3Divnsb&rurl=translate.google.com&sl=en&u=http://en.wikipedia.org/wiki/Method_%28computer_science%29&usg=ALkJrhig1pvf5S551Zxlic7sRTgx28FeFQ) bersama dengan interaksi mereka untuk merancang aplikasi dan program komputer. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika [pemrograman terstruktur](http://id.wikipedia.org/wiki/Pemrograman_terstruktur" \o "Pemrograman terstruktur). Setiap objek dapat menerima [pesan](http://id.wikipedia.org/wiki/Pesan" \o "Pesan), memproses data, dan mengirim pesan ke objek lainnya. Pada jaman sekarang, banyak bahasa pemrograman yang mendukung OOP.

OOP adalah paradigma pemrograman yang cukup dominan saat ini, karena mampu memberikan solusi kaidah pemrograman modern. Meskipun demikian, bukan berarti bahwa pemrograman prosedural sudah tidak layak lagi. OOP diciptakan karena dirasakan masih adanya keterbatasan pada bahasa pemrograman tradisional. Konsep dari OOP sendiri adalah semua pemecahan masalah dibagi ke dalam objek. Dalam OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek. Proses perancangan atau desain dalam suatu pemrograman merupakan proses yang tidak terpisah dari proses yang mendahului, yaitu analisis dan proses yang mengikutinya. Pembahasan mengenai orientasi objek tidak akan terlepas dari konsep objek seperti inheritance atau penurunan, encapsulation atau pembungkusan, dan polymorphism atau kebanyakrupaan. Konsep-konsep ini merupakan fundamental dalam orientasi objek yang perlu sekali dipahami serta digunakan dengan baik, dan menghindari penggunaannya yang tidak tepat.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam [teknik piranti lunak](http://id.wikipedia.org/w/index.php?title=Teknik_piranti_lunak&action=edit&redlink=1" \o "Teknik piranti lunak (halaman belum tersedia)) skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

## 1.2 Tujuan

* Memenuhi tugas mata kuliah pemrograman objek.
* Memahami tentang pemrograman berbasis oop.
* Supaya pembaca ataupun user dapat memahami pemrograman Java lebih luas dengan adanya laporan praktikum ini.

## 1.3 Manfaat

Memang kelihatannya konsep ini sangat rumit, tetapi ada beberapa alasan mengapa pendekatan objek sangat kuat diantaranya:

* Konsep ini menyederhanakan kompleksitas dengan memungkinkan kita untuk mendefinisikan sebuah sistem besar dan kompleks menggunakan set yang lebih kecil dari objek yang saling terkait.
* Dengan merancang aplikasi menggunakan objek, berarti kita telah menerapkan permodelan yang mendekati kehidupan nyata. Hal ini memungkinkan desain program kita menjadi lebih alamiah , yang memungkinkan kita untuk bekerja lebih intuitif.
* Konsep ini memberi kita sebuah kosa kata/ vocabulary, sehingga kita dapat lebih efektif mendiskusikan sistem atau aplikasi dengan rekan kerja kita (jauh lebih mudah untuk membahas fungsi dalam hal metode obyek daripada salah satu dari ratusan fungsi yang berada dalam modul kode umum).
* Class memiliki sifat mereka sendiri (properti), metode, dan event terintegrasi atau dikenal dengan istilah di-encapsulation, sehingga mereka umumnya mandiri. Ini membuat kode kita lebih teratur dan lebih mudah untuk di-maintain.
* Adanya enkapsulasi berarti, penggunaan kembali kode menjadi mudah, karena kita cukup menyalin kelas template dan perubahan akan secara otomatis tercermin dalam setiap kelas yang mewarisi dari kelas dasar yang telah kita ubah. Hal ini bermanfaat ketika berbagi/ sharing kode dengan orang lain, atau ketika menggunakan kembali kode dari proyek-proyek lama, sehingga berimplikasi pada penghematan biaya dan waktu.

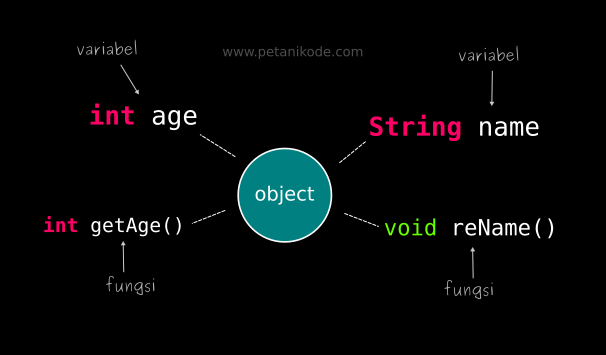
**BAB II**

**PEMBAHASAN**

## 2.1 Analisa Hasil Percobaan Java OOP

OOP (*Object Oriented Programming*) atau dalam bahasa indonesia dikenal dengan pemrograman berorientasikan objek (PBO) merupakan sebuah paradigma atau teknik pemrograman yang berorientesikan Objek.

Pada OOP, Fungsi dan variabel **dibungkus** dalam sebuah **objek** atau *class* yang dapat saling brinteraksi, sehingga membentuk sebuah program.



Variabel dalam objek akan menyimpan data dari objek. Sedangkan fungsi akan menentukan operasinya.

Contoh objek dalam dunia nyata: Mobil, Burung, Drone, Meja, Pohon, dll.

------------------OBJEK

Drone

------------------Variabel/Atribut

energi = 100;

ketinggian = 200;

kecepatan = 29;

------------------Fungsi

terbang();

matikanMesin();

turun();

maju();

mundur();

belok();

------------------

Semua objek di dunia nyata yang memiliki sifat dan tingkah laku, bisa kita representasikan dalam kode.

**2.2 Analisa Hasil Percobaan** **Java Class & Object**

Java adalah bahasa pemrograman berorientasi objek.

Segala sesuatu di Java dikaitkan dengan kelas dan objek, bersama dengan atribut dan metodenya. Sebagai contoh: dalam kehidupan nyata, mobil adalah objek. Mobil memiliki **atribut** , seperti berat dan warna, dan **metode** , seperti drive dan rem.

Kelas seperti konstruktor objek, atau "cetak biru" untuk membuat objek.

* **Create a Class**

Untuk membuat kelas, gunakan class kata kunci:

* **MyClass.java**

Buat kelas bernama " MyClass " dengan variabel x:

public class MyClass { int x = 5; }

Ingat dari [bab Java Syntax](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=auto&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_syntax.asp&xid=17259,15700021,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhgasDqQzD-ci9E6skRLW0YIAQy1-Q) bahwa suatu kelas harus selalu dimulai dengan huruf besar pertama, dan bahwa nama file java harus cocok dengan nama kelas.

* **Create an Object**

Di Java, objek dibuat dari kelas. Kami telah membuat kelas bernama MyClass , jadi sekarang kita bisa menggunakan ini untuk membuat objek.

Untuk membuat objek MyClass , tentukan nama kelas, diikuti oleh nama objek, dan gunakan kata kunci new :

Contoh :

Buat objek yang disebut " myObj " dan cetak nilai x:

public class MyClass {

int x = 5;

public static void main(String[] args) {

MyClass myObj1 = new MyClass();

MyClass myObj2 = new MyClass();

System.out.println(myObj1.x);

System.out.println(myObj2.x);

}

}

5  
5

* **Using Multiple Class**

Anda juga bisa membuat objek kelas dan mengaksesnya di kelas lain. Ini sering digunakan untuk organisasi kelas yang lebih baik (satu kelas memiliki semua atribut dan metode, sedangkan kelas lainnya memegang metode main() (kode yang akan dieksekusi)).

Ingat bahwa nama file java harus sesuai dengan nama kelas. Dalam contoh ini, kami telah membuat dua file di direktori / folder yang sama:

* MyClass.java
* OtherClass.java

#### **MyClass.java**

public class MyClass {

int x = 5;

}

#### **OtherClass.java**

class OtherClass {

public static void main(String[] args) {

MyClass **myObj** = new MyClass();

System.out.println(myObj.x);

}

}

Ketika kedua file telah dikompilasi:

C:\Users\ Your Name >javac MyClass.java  
C:\Users\ Your Name >javac OtherClass.java

Jalankan file OtherClass.java:

C:\Users\ Your Name >java OtherClass

Outputnya adalah:

5

## 2.3 Analisa Hasil Percobaan Java Class Attributes

Pada bab sebelumnya, kami menggunakan istilah "variabel" untuk x dalam contoh (seperti yang ditunjukkan di bawah). Ini sebenarnya adalah **atribut** dari kelas. Atau Anda bisa mengatakan bahwa atribut kelas adalah variabel dalam kelas:

Contoh

Create a class called "MyClass" with two attributes: x and y:

public class MyClass {

int x = 5;

int y = 3;

}

* **Accessing Attributes**

Anda bisa mengakses atribut dengan membuat objek kelas, dan dengan menggunakan sintaks dot ( . ):

Contoh berikut akan membuat objek kelas MyClass , dengan nama myObj . Kami menggunakan atribut x pada objek untuk mencetak nilainya:

Contoh

Buat objek yang disebut " myObj " dan cetak nilai x :

public class MyClass {

int x = 5;

public static void main(String[] args) {

MyClass myObj = new MyClass();

System.out.println(myObj.x);

}

}

Outputnya adalah:

**5**

* + - * **Modify Attribut**

Anda juga dapat mengubah nilai atribut:

Contoh

Tetapkan nilai x hingga 40:

public class MyClass {

int x;

public static void main(String[] args) {

MyClass myObj = new MyClass();

myObj.x = 40;

System.out.println(myObj.x);

}

}

Outputnya adalah:

40

Atau timpa nilai yang ada:

Contoh

Ubah nilai x menjadi 25:

public class MyClass {

int x = 10;

public static void main(String[] args) {

MyClass myObj = new MyClass();

myObj.x = 25; // x is now 25

System.out.println(myObj.x);

}

}

Outputnya adalah:

25

Jika Anda tidak ingin kemampuan untuk menimpa nilai yang ada, nyatakan atribut sebagai final :

Contoh

public class MyClass {

**final** int x = 10;

public static void main(String[] args) {

MyClass myObj = new MyClass();

myObj.x = 25; // akan menghasilkan kesalahan: tidak dapat menetapkan nilai ke final variable

System.out.println(myObj.x);

}

}

Outputnya adalah:

MyClass.java:6: error: cannot assign a value to final variable x  
    myObj.x = 25;  
         ^  
1 error

Kata kunci final berguna ketika Anda ingin variabel selalu menyimpan nilai yang sama, seperti PI (3.14159 ...).

* **Multiple Objects**

Jika Anda membuat beberapa objek dari satu kelas, Anda bisa mengubah nilai atribut di satu objek, tanpa memengaruhi nilai atribut di yang lain:

Contoh

Ubah nilai x menjadi 25 di myObj2 , dan biarkan x di myObj1 tidak berubah:

public class MyClass {

int x = 5;

public static void main(String[] args) {

MyClass myObj1 = new MyClass(); // Object 1

MyClass myObj2 = new MyClass(); // Object 2

myObj2.x = 25;

System.out.println(myObj1.x); // Outputs 5

System.out.println(myObj2.x); // Outputs 25

}

}

Outputnya adalah:

5

25

* **Multiple Attributes**

Anda dapat menentukan atribut sebanyak yang Anda inginkan:

Contoh

public class Person {

String fname = "John";

String lname = "Doe";

int age = 24;

public static void main(String[] args) {

Person myObj = new Person();

System.out.println("Name: " + myObj.fname + " " + myObj.lname);

System.out.println("Age: " + myObj.age);

}

}

Outputnya adalah:

Name: John Doe  
Age: 24

**2.4 Analisa Hasil Percobaan** **Java Method**

Anda belajar dari bab [Metode Java](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=en&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_methods.asp&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhii2CNnWae5W2hv74FqedaDSyReQA) bahwa metode dideklarasikan dalam kelas, dan bahwa mereka digunakan untuk melakukan tindakan tertentu:

**Contoh**

Buat metode bernama myMethod() di MyClass:

public class MyClass {

static void myMethod() {

System.out.println("Hello World!");

}

}

myMethod() mencetak teks (aksi), ketika **dipanggil** . Untuk memanggil metode, tulis nama metode diikuti oleh dua tanda kurung **()** dan tanda titik koma **;**

**Contoh**

Di dalam main , panggil myMethod() :

public class MyClass {

static void myMethod() {

System.out.println("Hello World!");

}

public static void main(String[] args) {

myMethod();

}

}

Outputnya adalah:

"Hello World!"

* **Statis vs Non-Statis**

Anda akan sering melihat program Java yang memiliki atribut dan metode static atau public .

Dalam contoh di atas, kami membuat metode static , yang berarti dapat diakses tanpa membuat objek

Contoh untuk menunjukkan perbedaan antara **metode** static dan public :

public class MyClass {

// Static method

static void myStaticMethod() {

System.out.println("Static methods can be called without creating objects");

}

// Public method

public void myPublicMethod() {

System.out.println("Public methods must be called by creating objects");

}

// Main method

public static void main(String[] args) {

myStaticMethod(); // Call the static method

// myPublicMethod(); This would compile an error

MyClass myObj = new MyClass(); // Create an object of MyClass

myObj.myPublicMethod(); // Call the public method on the object

}

}

Outputnya adalah:

Static methods can be called without creating objects  
Public methods must be called by creating objects

* **Metode Akses Dengan Objek (Access Methods With an Object)**

Contoh

Buat objek Mobil bernama myCar . Panggil metode fullThrottle() dan speed() pada objek myCar , dan jalankan program:

// Membuat Kelas Car

public class Car {

// Membuat a fullThrottle() method

public void fullThrottle() {

System.out.println("The car is going as fast as it can!");

}

// Membuat speed() method dan menambahkan parameters

public void speed(int maxSpeed) {

System.out.println("Max speed is: " + maxSpeed);

}

// Inside main, call the methods on the myCar object

public static void main(String[] args) {

Car myCar = new Car(); // Create a myCar object

myCar.fullThrottle(); // Call the fullThrottle() method

myCar.speed(200); // Call the speed() method

}

}

Outputnya adalah:

The car is going as fast as it can!  
Max speed is: 200

**Contoh dijelaskan**

1. Kami membuat kelas Car khusus dengan kata kunci class .
2. Kami membuat metode fullThrottle() dan speed() di kelas Car .
3. Metode fullThrottle() dan metode speed() akan menampilkan beberapa teks, ketika mereka dipanggil.
4. Metode speed() menerima parameter int disebut maxSpeed - kita akan menggunakannya dalam **8)** .
5. Untuk menggunakan kelas Car dan metodenya, kita perlu membuat **objek** dari Kelas Car .
6. Lalu, buka metode main() , yang Anda tahu sekarang adalah metode Java bawaan yang menjalankan program Anda (kode apa pun di dalam main dieksekusi).
7. Dengan menggunakan kata kunci new kami membuat objek Car dengan nama myCar .
8. Kemudian, kita memanggil metode fullThrottle() dan speed() pada objek myCar , dan menjalankan program menggunakan nama objek ( myCar ), diikuti oleh titik ( . ), Diikuti oleh nama metode ( fullThrottle(); dan speed(200); ). Perhatikan bahwa kita menambahkan parameter int **200** di dalam metode speed() .

**Ingat bahwa..**

Titik ( . ) Digunakan untuk mengakses atribut dan metode objek.

Untuk memanggil metode di Jawa, tulis nama metode diikuti dengan seperangkat tanda kurung **()** , diikuti dengan tanda titik koma ( ; ).

Kelas harus memiliki nama file yang cocok ( Car dan **Car.java** ).

* **Using Multiple Classes**

Seperti yang kami sebutkan di [bab Kelas](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=en&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_classes.asp&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhhhhDlFHobgvukqihx3pqxwsKZuog) , itu adalah praktik yang baik untuk membuat objek kelas dan mengaksesnya di kelas lain.

Ingat bahwa nama file java harus sesuai dengan nama kelas. Dalam contoh ini, kami telah membuat dua file di direktori yang sama:

* Car.java
* Other Class.java

**Car.java**

public class Car {

public void fullThrottle() {

System.out.println("The car is going as fast as it can!");

}

public void speed(int maxSpeed) {

System.out.println("Max speed is: " + maxSpeed);

}

}

**OtherClass.java**

class OtherClass {

public static void main(String[] args) {

Car myCar = new Car(); // Create a myCar object

myCar.fullThrottle(); // Call the fullThrottle() method

myCar.speed(200); // Call the speed() method

}

}

Ketika kedua file telah dikompilasi:

C:\Users\ Your Name >javac Car.java  
C:\Users\ Your Name >javac OtherClass.java

Jalankan file OtherClass.java:

C:\Users\ Your Name >java OtherClass

Outputnya adalah:

The car is going as fast as it can!  
Max speed is: 200

## 2.5 Analisa Hasil Percobaan Java Constructor

Konstruktor di Java adalah **metode khusus** yang digunakan untuk menginisialisasi objek. Konstruktor dipanggil ketika objek kelas dibuat. Ini dapat digunakan untuk mengatur nilai awal untuk atribut objek:

Contoh

public class MyClass {

int x; // membuat attribute class

// membuat kelas konstruktor untuk kelas MyClass

public MyClass() {

x = 5; // mengisi nilai dari atribut x dalam MyClass

}

public static void main(String[] args) {

MyClass myObj = new MyClass(); // membuat objek dalam kelas myclass(bertujuan untuk menmanggil kelas konstruktor)

System.out.println(myObj.x); // Menampilkan hasil nilai x

}

}

Outputnya adalah:

5

Perhatikan bahwa nama konstruktor harus **cocok dengan nama kelas** , dan tidak boleh memiliki **tipe kembali** (seperti void ).

Perhatikan juga bahwa konstruktor dipanggil saat objek dibuat.

Semua kelas memiliki konstruktor secara default: jika Anda tidak membuat konstruktor kelas sendiri, Java membuat satu untuk Anda. Namun, maka Anda tidak dapat menetapkan nilai awal untuk atribut objek.

* **Parameter konstruktor**

Konstruktor juga dapat mengambil parameter, yang digunakan untuk menginisialisasi atribut.

Contoh berikut menambahkan parameter int y ke konstruktor. Di dalam konstruktor kita atur x ke y (x = y). Ketika kita memanggil konstruktor, kita meneruskan parameter ke konstruktor (5), yang akan menetapkan nilai x ke 5:

public class MyClass {

int x;

public MyClass(int y) {

x = y;

}

public static void main(String[] args) {

MyClass myObj = new MyClass(5);

System.out.println(myObj.x);

}

}

Outputnya adalah:

5

Contoh Java Constructor Parameter

public class Constructor\_Parameters {

long modelNim;

String modelName;

String modelAlamat;

String modelJurusan;

long modelAngkatan;

public Constructor\_Parameters(long nim, String name, String Alamat, String Jurusan, long Angkatan) {

modelNim = nim;

modelName = name;

modelAlamat = Alamat;

modelJurusan = Jurusan;

modelAngkatan = Angkatan;

}

public static void main(String[] args) {

Constructor\_Parameters myProfile = new Constructor\_Parameters(201869040009L,"Asy'ari fandi ahmad ","Gesing - Randupitu - Gempol","Teknik Informatika", 2018L);

System.out.println(myProfile.modelNim + " " + myProfile.modelName + " " + myProfile.modelAlamat + " " + myProfile.modelJurusan + " " + myProfile.modelAngkatan );

}

}

Outputnya adalah:

201869040009 Asy'ari fandi ahmad Gesing - Randupitu - Gempol Teknik Informatika 2018

## 2.6 Analisa Hasil Percobaan Java Modifier

Sekarang, Anda cukup akrab dengan kata kunci public yang muncul di hampir semua contoh kami:

**public** class MyClass

Kata kunci public adalah **pengubah akses** , artinya digunakan untuk mengatur tingkat akses untuk kelas, atribut, metode, dan konstruktor.

Kami membagi pengubah menjadi dua kelompok:

* **Access Modifiers** - mengontrol level akses
* **Non-Access Modifiers** - tidak mengontrol level akses, tetapi menyediakan fungsionalitas lain.

**6.1 Access Modifiers**

Untuk **kelas** , Anda dapat menggunakan public atau default :

* **Public**

Kelas dapat diakses oleh kelas lain

public class Access\_Modifiers\_public {

public String name = "Asy'ari fandi ahmad";

public String lname = "(Asy'ari)";

public String email = "asyarifandiahmad03@gmail.com";

public int umur = 20;

public static void main(String[] args) {

Access\_Modifiers\_public myObj = new Access\_Modifiers\_public();

System.out.println("Name : " + myObj.name + " " + myObj.lname);

System.out.println("Email : " + myObj.email);

System.out.println("Umur : " + myObj.umur);

}

}

Outputnya adalah:

Name : Asy'ari fandi ahmad (Asy'ari)

Email : asyarifandiahmad03@gmail.com

Umur : 20

* **Default**

Kelas hanya dapat diakses oleh kelas-kelas dalam paket yang sama. Ini digunakan ketika Anda tidak menentukan pengubah. Anda akan mempelajari lebih lanjut tentang paket di bab Paket

public class Access\_Modifiers\_default {

String name = "Asy'ari fandi ahmad";

String name\_p = "(Asy'ari)";

String email = "asyarifandiahmad03@gmail.com";

int umur = 20;

public static void main(String[] args) {

Access\_Modifiers\_default myObj = new Access\_Modifiers\_default();

System.out.println("Name : " + myObj.name + " " + myObj.name\_p);

System.out.println("Email : " + myObj.email);

System.out.println("Umur : " + myObj.umur);

}

}

Outputnya adalah:

Name : Asy'ari fandi ahmad (Asy'ari)

Email : asyarifandiahmad03@gmail.com

Umur : 20

* **Private**

Kode hanya dapat diakses di dalam kelas yang dideklarasikan

//kode dibawah ini dalah sebuah variable yang akan dipanggil dalam 1 class

//kode private ini merupakan kode yang antara variable dan pemanggilnya dijadikan 1 class

public class Access\_Modifiers\_private {

private String nama = "Asy'ari fandi ahmad";

private String nama\_p = "Asy'ari";

private String email = "asyarifandiahmad03@gmail.com";

private int umur = 20;

//kode dibawah ini untuk memanggil variable yang akan ditampilkan dalam output

//kode ini juga dapat digunakan baik di class private maupun di class public

public static void main(String[] args) {

Access\_Modifiers\_private myObj = new Access\_Modifiers\_private();

System.out.println("Nama : " + myObj.nama );

System.out.println ("Nama Panggilan : " + myObj.nama\_p);

System.out.println("Email : " + myObj.email);

System.out.println("Umur : " + myObj.umur);

}

}

Outputnya adalah:

Nama : Asy'ari fandi ahmad

Nama Panggilan : Asy'ari

Email : asyarifandiahmad03@gmail.com

Umur : 20

* **Protected**

Kode ini dapat diakses dalam paket dan subclass yang sama. Anda akan belajar lebih banyak tentang subclass dan superclasses di bab Inheritance

public class Access\_Modifiers\_protected {

protected String fname = "Asy'ari fandi ahmad";

protected String lname = "Asy'ari";

protected String email = "asyarifandiahmad03@gmail.com";

protected int age = 20;

}

class Student extends Access\_Modifiers\_protected {

private int graduationYear = 2018;

public static void main(String[] args) {

Student myObj = new Student();

System.out.println("Name : " + myObj.fname + " " + myObj.lname);

System.out.println("Email : " + myObj.email);

System.out.println("Age : " + myObj.age);

System.out.println("Graduation Year : " + myObj.graduationYear);

}

}

Outputnya adalah:

Name : Asy'ari fandi ahmad Asy'ari

Email : asyarifandiahmad03@gmail.com

Age : 20

Graduation Year : 2018

**6.2 Non-Access Modifiers**

Untuk kelas, Anda dapat menggunakan final atau abstrak:

* **Final**

Kelas tidak dapat diwarisi oleh kelas lain (Anda akan belajar lebih banyak tentang warisan dalam bab Warisan)

//class ini sudah bersifat final tidak bisa diubah nilai variabelnya

public class Non\_Access\_Modifiers\_final {

final int x = 10;

final double PI = 3.14;

public static void main(String[] args) {

Non\_Access\_Modifiers\_final myObj = new Non\_Access\_Modifiers\_final();

myObj.x = 50; // will generate an error: cannot assign a value to a final variable

myObj.PI = 25; // will generate an error: cannot assign a value to a final variable

System.out.println(myObj.x);

}

}

Outputnya adalah:

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - cannot assign a value to final variable x at Non\_Access\_Modifiers.Non\_Access\_Modifiers\_final.main(Non\_Access\_Modifiers\_final.java:18)

C:\Users\asy'ari\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1

* **Abstract**

Kelas tidak dapat digunakan untuk membuat objek (Untuk mengakses kelas abstrak, itu harus diwarisi dari kelas lain. Anda akan belajar lebih banyak tentang warisan dan abstraksi di bab Warisan dan Abstraksi)

//kelas yang mewarisi main class

abstract class Non\_Accesss\_Modifiers\_Abstract {

public String fname = "Asy'ari fandi ahmad";

public int age = 20;

public abstract void study(); // abstract method

}

class Student extends Non\_Accesss\_Modifiers\_Abstract {

public int graduationYear = 2018;

public void study() { // the body of the abstract method is provided here

System.out.println("Belajar Sepanjang Hari");

}

}

//kelas ini tidak bisa membuat objek sendiri. tapi hanya bisa memanggil dari kelas lain(pewarisan)

class MyClass{

public static void main(String[] args) {

// create an object of the Student class (which inherits attributes and methods from Person)

Student myObj = new Student();

System.out.println("Name : " + myObj.fname);

System.out.println("Age : " + myObj.age);

System.out.println("Lulusan Tahun : " + myObj.graduationYear);

myObj.study(); // call abstract method

}

}

Outputnya adalah:

Name : Asy'ari fandi ahmad

Age : 20

Lulusan Tahun : 2018

Belajar Sepanjang Hari

## 2.7 Analisa Hasil Percobaan Java Enkapsulasi

Arti **Enkapsulasi** , adalah untuk memastikan bahwa data "sensitif" disembunyikan dari pengguna. Untuk mencapai ini, Anda harus:

* mendeklarasikan variabel / atribut kelas sebagai private
* memberikan metode **get** dan **set** publik untuk mengakses dan memperbarui nilai variabel private
* **Dapatkan dan Atur**

Anda telah belajar dari bab sebelumnya bahwa variabel private hanya dapat diakses di dalam kelas yang sama (kelas luar tidak memiliki akses ke sana). Namun, dimungkinkan untuk mengaksesnya jika kami menyediakan metode **get** dan **set** publik.

Metode get mengembalikan nilai variabel, dan metode **set** menetapkan nilai.

Sintaks untuk keduanya adalah bahwa mereka mulai dengan get atau set, diikuti dengan nama variabel, dengan huruf pertama dalam huruf besar:

**Contoh**

public class Encapsulation {

private String name;

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String newName) {

this.name = newName;

}

public static void main(String[] args) {

Encapsulation myObj = new Encapsulation ();

myObj.name = "Asy'ari";

System.out.println(myObj.name);

}

}

Outputnya adalah:

Asy'ari

**Dijelaskan**

Metode get mengembalikan nilai name variabel. metodeset mengambil parameter ( newName ) dan menetapkannya ke variabel name . Kata kunci ini digunakan untuk merujuk ke objek saat ini.

Namun, karena variabel name dinyatakan sebagai private , kami **tidak dapat** mengaksesnya dari luar kelas ini:

Contoh

public class myclass {

public static void main(String[] args) {

Encapsulation myObj = new Encapsulation();

myObj.setName("John"); // Set the value of the name variable to "John"

System.out.println(myObj.getName());

}

}

Jika variabel dinyatakan sebagai public , kami akan mengharapkan output seperti berikut ini:

John

Namun, saat kami mencoba mengakses variabel private , kami mendapatkan kesalahan:

MyClass.java:4: error: name has private access in Person  
    myObj.name = "John";  
         ^  
MyClass.java:5: error: name has private access in Person  
    System.out.println(myObj.name);  
                  ^  
2 errors

Sebagai gantinya, kami menggunakan metode getName() dan setName() untuk mengakses dan memperbarui variabel:

Contoh

public class myclass {

public static void main(String[] args) {

Encapsulation myObj = new Encapsulation();

myObj.setName("John"); // Set the value of the name variable to "John"

System.out.println(myObj.getName());

}

}

Outputnya adalah:

John

**Mengapa Enkapsulasi?**

* Kontrol atribut dan metode kelas yang lebih baik
  + Atribut kelas dapat dibuat **hanya-baca** (jika Anda hanya menggunakan metode get ), atau **hanya-tulis** (jika Anda hanya menggunakan metode yang set )
  + Fleksibel: programmer dapat mengubah satu bagian kode tanpa mempengaruhi bagian lainnya
* Peningkatan keamanan data

## 2.8 Analisa Hasil Percobaan Java Paket & API

Paket di Java digunakan untuk mengelompokkan kelas yang terkait. Anggap saja sebagai **folder dalam direktori file** . Kami menggunakan paket untuk menghindari konflik nama, dan untuk menulis kode yang dapat dikelola dengan lebih baik. Paket dibagi menjadi dua kategori:

* Paket Built-in (paket dari Java API)
* Paket Buatan Pengguna (buat paket Anda sendiri)
* Paket Bawaan

Java API adalah pustaka kelas yang sudah ditulis sebelumnya, yang bebas digunakan, termasuk dalam Java Development Environment. Perpustakaan berisi komponen untuk mengelola input, pemrograman basis data, dan banyak lagi lainnya.

Perpustakaan dibagi menjadi beberapa **paket** dan **kelas** . Berarti Anda dapat mengimpor satu kelas (beserta metode dan atributnya), atau seluruh paket yang berisi semua kelas yang termasuk dalam paket yang ditentukan.

Untuk menggunakan kelas atau paket dari perpustakaan, Anda perlu menggunakan kata kunci import :

* **Sintaksis**

import package . name . Class ; // Import a single class import package . name .\*; // Import the whole package

* **Impor Kelas**

Jika Anda menemukan kelas yang ingin Anda gunakan, misalnya, kelas Scanner , **yang digunakan untuk mendapatkan input pengguna** , tulis kode berikut:

Contoh

import java.util.Scanner;

Pada contoh di atas, java.util adalah sebuah paket, sementara Scanner adalah kelas dari paket java.util .

Untuk menggunakan kelas Scanner , buat objek kelas dan gunakan salah satu metode yang tersedia yang ditemukan dalam dokumentasi kelas Scanner . Dalam contoh kita, kita akan menggunakan metode nextLine() , yang digunakan untuk membaca baris lengkap:

Contoh

Menggunakan kelas Scanner untuk mendapatkan input pengguna:

import java.util.Scanner;

class MyClass {

public static void main(String[] args) {

Scanner myObj = new Scanner(System.in);

System.out.println("Enter username"); String userName = myObj.nextLine();

System.out.println("Username is: " + userName);

}

}

Outputnya adalah:

Enter username

jabrik

Username is: jabrik

* **Impor Paket**

Ada banyak paket untuk dipilih. Pada contoh sebelumnya, kami menggunakan kelas Scanner dari paket java.util . Paket ini juga berisi fasilitas tanggal dan waktu, generator nomor acak, dan kelas utilitas lainnya.

Untuk mengimpor seluruh paket, akhiri kalimat dengan tanda bintang ( \* ). Contoh berikut akan mengimpor SEMUA kelas dalam paket java.util :

Contoh

import java.util.\*;

* **Paket yang ditentukan pengguna**

Untuk membuat paket Anda sendiri, Anda perlu memahami bahwa Java menggunakan direktori sistem file untuk menyimpannya. Sama seperti folder di komputer Anda:

Contoh

└── root

└── mypack

└── MyPackageClass.java

Untuk membuat paket, gunakan kata kunci package :

MyPackageClass.java

package mypack;

class MyPackageClass {

public static void main(String[] args) {

System.out.println("This is my package!");

}

}

Simpan file sebagai **MyPackageClass.java** , dan kompilasi:

C:\Users\ Your Name >javac MyPackageClass.java

Kemudian kompilasi paket:

C:\Users\ Your Name >javac -d . MyPackageClass.java

Ini memaksa kompiler untuk membuat paket "mypack".

Kata kunci -d menentukan tujuan tempat penyimpanan file kelas. Anda dapat menggunakan nama direktori apa saja, seperti c: / user (windows), atau, jika Anda ingin menyimpan paket dalam direktori yang sama, Anda dapat menggunakan tanda titik " . ", Seperti dalam contoh di atas.

**Catatan:**

Nama paket harus ditulis dalam huruf kecil untuk menghindari konflik dengan nama kelas.

Ketika kami mengompilasi paket dalam contoh di atas, folder baru dibuat, disebut "mypack".

Untuk menjalankan file **MyPackageClass.java** , tulis berikut ini:

C:\Users\ Your Name >java mypack.MyPackageClass

Outputnya adalah:

This is my package!

## 2.9 Analisa Hasil Percobaan Java Inheritance/ Pewarisan (Subclass dan Superclass)

Di Java, dimungkinkan untuk mewarisi atribut dan metode dari satu kelas ke kelas lain. Kami mengelompokkan "konsep pewarisan" ke dalam dua kategori:

* **subclass** (child) - kelas yang mewarisi dari kelas lain
* **superclass** (parent) - kelas yang diwarisi dari

Untuk mewarisi dari kelas, gunakan kata kunci extends .

Pada contoh di bawah ini, kelas Car (subkelas) mewarisi atribut dan metode dari kelas Vehicle (superclass):

Contoh

class Vehicle {

protected String brand = "Ford"; // Vehicle attribute

public void honk() { // Vehicle method System.out.println("Tuut, tuut!");

}

}

class Car extends Vehicle {

private String modelName = "Mustang"; // Car attribute

public static void main(String[] args) { // Create a myCar object

Car myCar = new Car(); // Call the honk() method (from the Vehicle class) on the myCar object

myCar.honk(); // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class System.out.println(myCar.brand + " " + myCar.modelName);

}

}

Outputnya adalah:

Tuut, tuut!

Ford Mustang

Apakah Anda memperhatikan pengubah yang protected di Vehicle?

Kami menetapkan atribut **Brand** di **Vehicle** ke protected [access modifier](https://www.w3schools.com/java/java_modifiers.asp). Jika disetel ke private , kelas Mobil tidak akan dapat mengaksesnya.

Implementasi Inheritance menggunakan konsep pewarisan subclass (kelas yang mewarisi dari kelas) :

package Inheritance;

import java.util.Scanner;

public class Bangun\_datar{

public static void main(String[] args) {

boolean ulang=true;

int menu;

Lingkaran L = new Lingkaran();

segitiga S = new segitiga();

Persegi B = new Persegi();

Persegi\_Panjang P = new Persegi\_Panjang();

Belah\_Ketupat BK = new Belah\_Ketupat();

Jajar\_Genjang JG = new Jajar\_Genjang();

Layang\_Layang LL = new Layang\_Layang();

Trapesium T = new Trapesium();

while ( ulang ) {

System.out.println("-<= Menu Bangun Datar =>-");

System.out.println("1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi");

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

Scanner input = new Scanner(System.in);

System.out.print("Pilih Menu : ");

menu = input.nextInt();

switch(menu) {

case 1 : L.Lingkaran();break;

case 2 : S.segitiga();break;

case 3 : B.Persegi();break;

case 4 : P.Persegi\_Panjang();break;

case 5 : BK.Belah\_Ketupat();break;

case 6 : JG.Jajar\_Genjang ();break;

case 7 : LL.Layang\_Layang ();break;

case 8 : T.Trapesium ();break;

case 9 : System.exit(0);

default:

System.out.println("Pilih salah satu nomor dari 1-9!");

}

}

System.out.println("");

}

}

* Class Belah Ketupat

package Inheritance;

import java.util.Scanner;

public class Belah\_Ketupat extends Bangun\_datar {

public void Belah\_Ketupat(){

double ab,bc,cd,ad,d1,d2,lbk,kbk,phi = 3.14;

Scanner b = new Scanner(System.in);

System.out.println("BELAH KETUPAT");

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

System.out.print("Masukkan A : ");

ab = b.nextDouble();

System.out.print("Masukkan B : ");

bc = b.nextDouble();

System.out.print("Masukkan C : ");

cd = b.nextDouble();

System.out.print("Masukkan D : ");

ad = b.nextDouble();

System.out.print("Masukkan diameter 1 : ");

d1 = b.nextDouble();

System.out.print("Masukkan diameter 2 : ");

d2 = b.nextDouble();

lbk = 2/d1\*d2;

kbk = ab+bc+cd+ad;

System.out.println("Luas Lingkaran adalah : "+lbk);

System.out.println("Keliling Lingkaran adalah : "+kbk);

}

}

* Class Jajar Genjang

package Inheritance;

import java.util.Scanner;

public class Jajar\_Genjang extends Bangun\_datar{

public void Jajar\_Genjang(){

double ab,bc,cd,ad,t,lj,kj;

Scanner c = new Scanner(System.in);

System.out.println("JAJAR GENJANG");

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

System.out.print("Masukkan A : ");

ab = c.nextDouble();

System.out.print("Masukkan B : ");

bc = c.nextDouble();

System.out.print("Masukkan C : ");

cd = c.nextDouble();

System.out.print("Masukkan D : ");

ad = c.nextDouble();

System.out.print("Masukkan T : ");

t = c.nextDouble();

lj = ab\*t;

kj = ab+bc+cd+ad;

System.out.println("Luas Bujur Sangkar adalah : "+lj);

System.out.println("Keliling Bujur Sangkar adalah : "+kj);

}

}

* Class Layang Layang

package Inheritance;

import java.util.Scanner;

public class Layang\_Layang extends Bangun\_datar {

public void Layang\_Layang(){

double ab,bc,cd,ad,d1,d2,ll,kl,phi = 3.14;

Scanner b = new Scanner(System.in);

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

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

System.out.print("Masukkan A : ");

ab = b.nextDouble();

System.out.print("Masukkan B : ");

bc = b.nextDouble();

System.out.print("Masukkan C : ");

cd = b.nextDouble();

System.out.print("Masukkan D : ");

ad = b.nextDouble();

System.out.print("Masukkan diameter 1 : ");

d1 = b.nextDouble();

System.out.print("Masukkan diameter 2 : ");

d2 = b.nextDouble();

ll = 2/d1\*d2;

kl = ab+bc+cd+ad;

System.out.println("Luas Lingkaran adalah : "+ll);

System.out.println("Keliling Lingkaran adalah : "+kl);

}

}

* Class Lingkaran

package Inheritance;

import java.util.Scanner;

public class Lingkaran extends Bangun\_datar {

public void Lingkaran(){

double r,d,ll,kl,phi = 3.14;

Scanner b = new Scanner(System.in);

System.out.println("LINGKARAN");

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

System.out.print("Masukkan jari-jari : ");

r = b.nextDouble();

System.out.print("Masukkan diameter : ");

d = b.nextDouble();

ll = phi\*r\*r;

kl = phi\*d;

System.out.println("Luas Lingkaran adalah : "+ll);

System.out.println("Keliling Lingkaran adalah : "+kl);

}

}

* Class Persegi

package Inheritance;

import java.util.Scanner;

public class Persegi extends Bangun\_datar{

public void Persegi(){

double s,lp,kp;

Scanner c = new Scanner(System.in);

System.out.println("BUJUR SANGKAR");

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

System.out.print("Masukkan sisi : ");

s = c.nextDouble();

lp = s\*s;

kp = 4\*s;

System.out.println("Luas Bujur Sangkar adalah : "+lp);

System.out.println("Keliling Bujur Sangkar adalah : "+kp);

}

}

* Class Persegi Tiga

package Inheritance;

import java.util.Scanner;

public class Persegi\_Panjang extends Bangun\_datar{

public void Persegi\_Panjang(){

double p,l,lpp,kpp;

Scanner d = new Scanner(System.in);

System.out.println("PERSEGI PANJANG");

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

System.out.print("Masukkan Panjang : ");

p = d.nextDouble();

System.out.print("Masukkan Lebar : ");

l = d.nextDouble();

lpp = p\*l;

kpp = 2\*(p+l);

System.out.println("Luas Persegi Panjang adalah : "+lpp);

System.out.println("Keliling Persegi Panjang adalah : "+kpp);

}

}

* Class Trapesium

package Inheritance;

import java.util.Scanner;

public class Trapesium extends Bangun\_datar{

public void Trapesium(){

double ab,bc,cd,ad,t,lt,kt;

Scanner c = new Scanner(System.in);

System.out.println("TRAPESIUM");

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

System.out.print("Masukkan A : ");

ab = c.nextDouble();

System.out.print("Masukkan B : ");

bc = c.nextDouble();

System.out.print("Masukkan C : ");

cd = c.nextDouble();

System.out.print("Masukkan D : ");

ad = c.nextDouble();

System.out.print("Masukkan T : ");

t = c.nextDouble();

lt = 2/(ab+bc)\*t;

kt = ab+bc+cd+ad;

System.out.println("Luas Bujur Sangkar adalah : "+lt);

System.out.println("Keliling Bujur Sangkar adalah : "+kt);

}

}

* Class Segitiga

package Inheritance;

import java.util.Scanner;

public class segitiga extends Bangun\_datar{

public void segitiga(){

double ab,bc,ac,t,ls,ks;

Scanner c = new Scanner(System.in);

System.out.println("SEGITIGA");

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

System.out.print("Masukkan A : ");

ab = c.nextDouble();

System.out.print("Masukkan B : ");

bc = c.nextDouble();

System.out.print("Masukkan C : ");

ac = c.nextDouble();

System.out.print("Masukkan T : ");

t = c.nextDouble();

ls = ab+bc+ac;

ks = 2/ab\*t;

System.out.println("Luas Bujur Sangkar adalah : "+ls);

System.out.println("Keliling Bujur Sangkar adalah : "+ks);

}

}

Outputnya adalah:

run:

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 1

LINGKARAN

=========

Masukkan jari-jari : 14

Masukkan diameter : 7

Luas Lingkaran adalah : 615.44

Keliling Lingkaran adalah : 21.98

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 2

SEGITIGA

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

Masukkan A : 23

Masukkan B : 45

Masukkan C : 2

Masukkan T : 32

Luas Bujur Sangkar adalah : 70.0

Keliling Bujur Sangkar adalah : 2.782608695652174

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 3

BUJUR SANGKAR

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

Masukkan sisi : 4

Luas Bujur Sangkar adalah : 16.0

Keliling Bujur Sangkar adalah : 16.0

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 4

PERSEGI PANJANG

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

Masukkan Panjang : 4

Masukkan Lebar : 8

Luas Persegi Panjang adalah : 32.0

Keliling Persegi Panjang adalah : 24.0

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 5

BELAH KETUPAT

=========

Masukkan A : 4

Masukkan B : 6

Masukkan C : 8

Masukkan D : 23

Masukkan diameter 1 : 17

Masukkan diameter 2 : 17

Luas Lingkaran adalah : 2.0

Keliling Lingkaran adalah : 41.0

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 6

JAJAR GENJANG

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

Masukkan A : 12

Masukkan B : 34

Masukkan C : 42

Masukkan D : 24

Masukkan T : 56

Luas Bujur Sangkar adalah : 672.0

Keliling Bujur Sangkar adalah : 112.0

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 7

LANYANG - LAYANG

=========

Masukkan A : 34

Masukkan B : 12

Masukkan C : 23

Masukkan D : 35

Masukkan diameter 1 : 21

Masukkan diameter 2 : 21

Luas Lingkaran adalah : 2.0

Keliling Lingkaran adalah : 104.0

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 8

TRAPESIUM

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

Masukkan A : 12

Masukkan B : 56

Masukkan C : 34

Masukkan D : 43

Masukkan T : 23

Luas Bujur Sangkar adalah : 0.6764705882352942

Keliling Bujur Sangkar adalah : 145.0

-<= Menu Bangun Datar =>-

1.Lingkaran 2. Segitiga 3. Persegi 4. Persegi panjang 5. Belah ketupat 6. Jajar genjang 7. Layang - layang 8. Trapesium 9.Keluar Aplikasi

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

Pilih Menu : 9

BUILD SUCCESSFUL (total time: 2 minutes 46 seconds)

**Mengapa Dan Kapan Menggunakan "Inheritance"?**

Berguna untuk penggunaan kembali kode: menggunakan kembali atribut dan metode dari kelas yang ada saat Anda membuat kelas baru.

* **Kata Kunci terakhir**

Jika Anda tidak ingin kelas lain mewarisi dari suatu kelas, gunakan kata kunci final :

Jika Anda mencoba mengakses kelas final , Java akan menghasilkan kesalahan:

final class Vehicle { ... } class Car extends Vehicle { ... }

Outputnya akan seperti ini:

Car.java:8: error: cannot inherit from final Vehicle  
class Car extends Vehicle {   
                  ^   
1 error)

## 2.10 Analisa Hasil Percobaan Java Polimorfisme

Polimorfisme berarti "banyak bentuk", dan itu terjadi ketika kita memiliki banyak kelas yang terkait satu sama lain melalui pewarisan.

Seperti yang kami sebutkan di bab sebelumnya; **Inheritance** memungkinkan kita mewarisi atribut dan metode dari kelas lain. **Polimorfisme** menggunakan metode-metode itu untuk melakukan tugas yang berbeda. Ini memungkinkan kami untuk melakukan satu tindakan dengan berbagai cara.

Sebagai contoh, pikirkan superclass bernama Animal yang memiliki metode yang disebut animalSound() . Subkelas Hewan dapat berupa Babi, Kucing, Anjing, Burung - Dan mereka juga memiliki implementasi sendiri dari suara binatang (babi oinks, dan kucing mengeong, dll.):

Contoh

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

Ingat dari [bab Inheritance](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=auto&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_inheritance.asp&xid=17259,15700021,15700186,15700190,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhg_vq-Wu25VX64gyDlZc3_tut88aA) bahwa kami menggunakan kata kunci extends untuk mewarisi dari suatu kelas.

Sekarang kita dapat membuat objek Pig dan Dog dan memanggil metode animalSound() pada keduanya:

Contoh

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

class MyMainClass {

public static void main(String[] args) {

Animal myAnimal = new Animal(); // Create a Animal object

Animal myPig = new Pig(); // Create a Pig object

Animal myDog = new Dog(); // Create a Dog object myAnimal.animalSound();

myPig.animalSound();

myDog.animalSound();

}

}

Outputnya adalah:

The animal makes a sound

The pig says: wee wee

The dog says: bow wow

**Mengapa dan Kapan Menggunakan "Inheritance" dan "Polimorfisme"?**

Berguna untuk penggunaan kembali kode: menggunakan kembali atribut dan metode dari kelas yang ada saat Anda membuat kelas baru.

## 2.11 Analisa Hasil Percobaan Java Inner Class

Di Java, mungkin juga untuk kelas sarang (kelas didalam kelas). Tujuan dari kelas bersarang adalah untuk mengelompokkan kelas-kelas yang termasuk bersama, yang membuat kode Anda lebih mudah dibaca dan dipelihara.

Untuk mengakses kelas dalam, buat objek dari kelas luar, dan kemudian buat objek dari kelas dalam:

Contoh

class OuterClass {

int x = 10;

class InnerClass {

int y = 5; } }

public class MyMainClass {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass(); System.out.println(myInner.y + myOuter.x);

}

}

Outputnya adalah:

15 (5 + 10)

* **Kelas Batin Pribadi**

Tidak seperti kelas "reguler", kelas dalam bisa bersifat private atau protected . Jika Anda tidak ingin objek luar mengakses kelas dalam, nyatakan kelas sebagai private :

Contoh

class OuterClass {

int x = 10;

**private** class InnerClass {

int y = 5;

}

}

public class MyMainClass {

public static void main(String[] args) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.y + myOuter.x);

}

}

Jika Anda mencoba mengakses kelas dalam pribadi dari kelas luar (MyMainClass), kesalahan terjadi:

MyMainClass.java:12: error:

OuterClass.InnerClass has private access in OuterClass  
    OuterClass.InnerClass myInner = myOuter.new InnerClass();  
              ^

* **Kelas Batin Statis**

Kelas dalam juga bisa menjadi static , yang berarti bahwa Anda dapat mengaksesnya tanpa membuat objek dari kelas luar:

Contoh

class OuterClass {

int x = 10;

static class InnerClass {

int y = 5;

}

}

public class MyMainClass {

public static void main(String[] args) {

OuterClass.InnerClass myInner = new OuterClass.InnerClass(); System.out.println(myInner.y);

}

}

Outputnya adalah:

5

**Catatan:** sama seperti atribut dan metode static kelas dalam static tidak memiliki akses ke anggota kelas luar.

* **Akses Kelas Luar Dari Kelas Dalam**

Satu keuntungan dari kelas dalam, adalah mereka dapat mengakses atribut dan metode dari kelas luar:

Contoh

class OuterClass {

int x = 10;

class InnerClass {

public int myInnerMethod() {

return x;

}

}

}

public class MyMainClass {

public static void main(String args[]) {

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass(); System.out.println(myInner.myInnerMethod());

}

}

Outputnya adalah:

10

## 2.12 Analisa Hasil Percobaan Java Abstrak

**Kelas dan Metode Abstrak Java**

**Abstraksi** data adalah proses menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna. Abstraksi dapat dicapai dengan **kelas** atau antarmuka **abstrak** (yang akan Anda pelajari lebih lanjut di bab selanjutnya).

Kata kunci abstract adalah pengubah non-akses, digunakan untuk kelas dan metode:

* **Kelas abstrak:** adalah kelas terbatas yang tidak dapat digunakan untuk membuat objek (untuk mengaksesnya, itu harus diwarisi dari kelas lain).
* **Metode abstrak:** hanya dapat digunakan dalam kelas abstrak, dan tidak memiliki tubuh. Tubuh disediakan oleh subclass (diwarisi dari).

Kelas abstrak dapat memiliki metode abstrak dan reguler:

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

}

}

Dari contoh di atas, tidak mungkin untuk membuat objek dari kelas Hewan:

Animal myObj = new Animal(); // will generate an error

Untuk mengakses kelas abstrak, itu harus diwarisi dari kelas lain. Mari kita konversi kelas Animal yang kita gunakan di bab **Polimorfisme** menjadi kelas abstrak:

Ingat dari bab **Warisan** bahwa kami menggunakan kata kunci extends untuk mewarisi dari suatu kelas.

Contoh

// Abstract class

abstract class Animal {

// Abstract method (does not have a body)

public abstract void animalSound(); // Regular method

public void sleep() {

System.out.println("Zzz");

}

}

// Subclass (inherit from Animal)

class Pig extends Animal {

public void animalSound() { //The body of animalSound() is provided here

System.out.println("The pig says: wee wee");

}

}

class MyMainClass {

public static void main(String[] args) {

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound();

myPig.sleep();

}

}

Outputnya adalah:

The pig says: wee wee

Zzz

#### **Mengapa dan Kapan Menggunakan Kelas dan Metode Abstrak?**

Untuk mencapai keamanan - sembunyikan detail tertentu dan hanya tampilkan detail penting suatu objek.

**Catatan:** Abstraksi juga dapat dicapai dengan Antarmuka , yang akan Anda pelajari lebih lanjut di bab selanjutnya.

## 2.13 Analisa Hasil Percobaan Java Interface

interface adalah " **kelas abstrak** " yang sepenuhnya digunakan untuk mengelompokkan metode terkait dengan benda kosong:

Contoh

// interface interface Animal {

public void animalSound(); //metode antarmuka (tidak memiliki tubuh)

public void run(); //metode antarmuka (tidak memiliki tubuh)

}

Untuk mengakses metode antarmuka, antarmuka harus "diimplementasikan" (agar seperti diwariskan) oleh kelas lain dengan kata kunci implements (alih-alih extends ). Isi metode antarmuka disediakan oleh kelas "implement":

Contoh

public interface interface\_teman {

interface teman {

public void TemanSound(); // interface method (does not have a body)

public void hudi(); // interface method (does not have a body)

}

// yudi "mengimplementasikan" interface teman

class yudi implements teman {

public void TemanSound() {

// TemanSound () disediakan di sini

System.out.println("yudi :Kemana hudi?");

}

public void hudi() {

// hudi () disediakan di sini

System.out.println("hudi :dia sepak bola di lapangan dekat kantor desa ");

}

}

class MyMainClass {

public static void main(String[] args) {

yudi myyudi = new yudi(); // Buat objek yudi

myyudi.TemanSound();

myyudi.hudi();

}

}

}

Outputnya adalah:

yudi :Kemana hudi?

hudi :dia sepak bola di lapangan dekat kantor desa

Mengapa Dan Kapan Menggunakan Antarmuka?

1) Untuk mencapai keamanan - sembunyikan detail tertentu dan hanya tampilkan detail penting dari suatu objek (antarmuka).

2) Java tidak mendukung "multiple inheritance" (sebuah kelas hanya dapat diwarisi dari satu superclass). Namun, ini dapat dicapai dengan antarmuka, karena kelas dapat **mengimplementasikan** banyak antarmuka. Untuk mengimplementasikan beberapa antarmuka, pisahkan dengan koma (lihat contoh di bawah).

* Multiple Interfaces

Untuk mengimplementasikan beberapa Interface, pisahkan dengan koma:

Contoh

package FirstInterface;

interface FirstInterface {

public void myMethod(); // interface method

}

interface SecondInterface {

public void myOtherMethod(); // interface method

}

class DemoClass implements FirstInterface, SecondInterface {

public void myMethod() {

System.out.println("Bagaimana kabarmu ?");

}

public void myOtherMethod() {

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

}

}

class MyMainClass {

public static void main(String[] args) {

DemoClass myObj = new DemoClass();

myObj.myMethod();

myObj.myOtherMethod();

}

}

Outputnya adalah:

Bagaimana kabarmu ?

baik - baik lur

Catatan tentang Interfaces:

* Seperti **kelas abstrak** , antarmuka **tidak** dapat digunakan untuk membuat objek (dalam contoh di atas, tidak mungkin membuat objek "Hewan" di MyMainClass)
* Metode antarmuka tidak memiliki tubuh - tubuh disediakan oleh kelas "implement"
* Pada implementasi antarmuka, Anda harus mengganti semua metodenya
* Metode antarmuka secara default abstract dan public
* Atribut antarmuka secara default public , static dan final
* Antarmuka tidak dapat berisi konstruktor (karena tidak dapat digunakan untuk membuat objek)

## 2.14 Analisa Hasil Percobaan Java Enums

Enum adalah "kelas" khusus yang mewakili sekelompok **konstanta** (variabel yang tidak dapat diubah, seperti variabel final ).

Untuk membuat enum , gunakan kata kunci enum (bukan kelas atau antarmuka), dan pisahkan konstanta dengan koma. Perhatikan bahwa mereka harus dalam huruf besar:

Contoh

enum Level {

LOW,

MEDIUM,

HIGH

}

public class MyClass {

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

System.out.println(myVar);

}

}

Outputnya adalah:

MEDIUM

Cacatan : Anda dapat mengakses konstanta enum dengan sintaks **dot** :

Level myVar = Level.MEDIUM;

* **Enum di dalam Kelas**

Anda juga dapat memiliki enum di dalam kelas:

Contoh

public class MyClass {

enum Level {

LOW,

MEDIUM,

HIGH

}

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

System.out.println(myVar);

}

}

Outputnya adalah:

MEDIUM

* **Enum dalam Pernyataan Switch**

Enum sering digunakan dalam pernyataan switch untuk memeriksa nilai yang sesuai:

Contoh

Enum Level {

LOW,

MEDIUM,

HIGH

}

public class MyClass {

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

switch(myVar) {

case LOW:

System.out.println("Low level");

break;

case MEDIUM:

System.out.println("Medium level");

break;

case HIGH:

System.out.println("High level");

break;

}

}

}

Outputnya adalah:

Medium level

* **Loop Through Enum**

Tipe enum memiliki metode values() , yang mengembalikan array dari semua konstanta enum. Metode ini berguna ketika Anda ingin mengulangi konstanta enum:

Contoh

for (Level myVar : Level.values()) {

System.out.println(myVar);

}

Outputnya adalah:

LOW  
MEDIUM  
HIGH

**Perbedaan antara Enums dan Kelas**

enum can, seperti halnya class , memiliki atribut dan metode. Satu-satunya perbedaan adalah bahwa konstanta enum bersifat public , static , dan final (tidak dapat diubah - tidak dapat diganti).

enum tidak dapat digunakan untuk membuat objek, dan itu tidak dapat memperluas kelas lain (tetapi dapat mengimplementasikan antarmuka).

**Mengapa dan Kapan Menggunakan Enum?**

Gunakan enum ketika Anda memiliki nilai yang Anda tahu tidak akan berubah, seperti hari bulan, hari, warna, setumpuk kartu, dll.

## 2.15 Analisa Hasil Percobaan Java User Input (Scanner)

Kelas Scanner digunakan untuk mendapatkan input pengguna, dan ditemukan dalam paket java.util . Untuk menggunakan kelas Scanner , buat objek kelas dan gunakan salah satu metode yang tersedia yang ditemukan dalam dokumentasi kelas Scanner . Dalam contoh kita, kita akan menggunakan metode nextLine() , yang digunakan untuk membaca Strings:

Contoh

import java.util.Scanner; // import the Scanner class

class MyClass {

public static void main(String[] args) {

Scanner myObj = new Scanner(System.in);

String userName;

// Enter username and press Enter

System.out.println("Enter username");

userName = myObj.nextLine();

System.out.println("Username is: " + userName);

}

}

Outputnya adalah:

Enter username

* **Jenis Input**

Pada contoh di atas, kami menggunakan metode nextLine() , yang digunakan untuk membaca Strings. Untuk membaca jenis lain, lihat tabel di bawah ini:

|  |  |
| --- | --- |
| **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 |

Dalam contoh di bawah ini, kami menggunakan metode berbeda untuk membaca data dari berbagai jenis:

Contoh

import java.util.Scanner;

class MyClass {

public static void main(String[] args) {

Scanner myObj = new Scanner(System.in);

System.out.println("Enter name, age and salary:");

// String input

String name = myObj.nextLine();

// Numerical input

int age = myObj.nextInt();

double salary = myObj.nextDouble();

// Output input by user

System.out.println("Name: " + name);

System.out.println("Age: " + age);

System.out.println("Salary: " + salary);

}

}

Outputnya adalah:

Enter name, age and salary:

Name : Jeson

Age : 2018

Salary : 85.000

## 2.16 Analisa Hasil Percobaan Java Date and Time

* **Tanggal java**

Java tidak memiliki kelas Date java.time , tetapi kami dapat mengimpor paket java.time untuk bekerja dengan API tanggal dan waktu. Paket ini mencakup banyak kelas tanggal dan waktu. Sebagai contoh:

|  |  |
| --- | --- |
| **Class** | **Description** |
| LocalDate | Represents a date (year, month, day (yyyy-MM-dd)) |
| LocalTime | Represents a time (hour, minute, second and milliseconds (HH-mm-ss-zzz)) |
| LocalDateTime | Represents both a date and a time (yyyy-MM-dd-HH-mm-ss.zzz) |
| DateTimeFormatter | Formatter for displaying and parsing date-time objects |

* **Tampilkan Tanggal Saat Ini**

Untuk menampilkan tanggal saat ini, impor kelas java.time.LocalDate , dan gunakan metode now() :

Contoh

import java.time.LocalDate; // import the LocalDate class

public class MyClass {

public static void main(String[] args) {

LocalDate myObj = LocalDate.now(); // Create a date object

System.out.println(myObj); // Display the current date

}

}

Outputnya adalah:

2019-12-14

* Tampilkan Waktu Saat Ini

Untuk menampilkan waktu saat ini (jam, menit, detik, dan milidetik), impor kelas java.time.LocalTime , dan gunakan metode now() :

Contoh

import java.time.LocalTime; // import the LocalTime class

public class MyClass {

public static void main(String[] args) {

LocalTime myObj = LocalTime.now();

System.out.println(myObj);

}

}

Outputnya adalah:

12:13:03.503724

* **Tampilkan Tanggal dan Waktu Saat Ini**

Untuk menampilkan tanggal dan waktu saat ini, impor kelas java.time.LocalDateTime , dan gunakan metode now() :

Contoh

import java.time.LocalDateTime; // import the LocalDateTime class

public class MyClass {

public static void main(String[] args) {

LocalDateTime myObj = LocalDateTime.now();

System.out.println(myObj);

}

}

Outputnya adalah:

2019-12-14T12:13:03.504598

* **Memformat Tanggal dan Waktu**

"T" pada contoh di atas digunakan untuk memisahkan tanggal dari waktu. Anda bisa menggunakan kelas DateTimeFormatter dengan metode ofPattern() dalam paket yang sama untuk memformat atau mem-parsing objek tanggal-waktu. Contoh berikut akan menghapus "T" dan milidetik dari tanggal-waktu:

Contoh

import java.time.LocalDateTime; // Import the LocalDateTime class

import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class MyClass {

public static void main(String[] args) {

LocalDateTime myDateObj = LocalDateTime.now();

System.out.println("Before formatting: " + myDateObj);

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

String formattedDate = myDateObj.format(myFormatObj);

System.out.println("After formatting: " + formattedDate);

}

}

Outputnya adalah:

Before Formatting: 2019-12-14T12:13:03.506064   
After Formatting: 14-12-2019 12:13:03

Metode ofPattern() menerima semua jenis nilai, jika Anda ingin menampilkan tanggal dan waktu dalam format yang berbeda. Sebagai contoh:

* *yyyy-MM-dd (*1988-09-29*)*

Contoh :

import java.time.LocalDateTime; // Import the LocalDateTime class

import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class MyClass {

public static void main(String[] args) {

LocalDateTime myDateObj = LocalDateTime.now();

System.out.println("Before formatting: " + myDateObj);

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

String formattedDate = myDateObj.format(myFormatObj);

System.out.println("After formatting: " + formattedDate);

}

}

Result:

Before Formatting: 2019-12-15T00:08:33.241010   
After Formatting: 15-12-2019 00:08:33

* *dd/MM/yyyy (*29/09/1988*)*

Contoh :

import java.time.LocalDateTime; // Import the LocalDateTime class

import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class MyClass {

public static void main(String[] args) {

LocalDateTime myDateObj = LocalDateTime.now();

System.out.println("Before Formatting: " + myDateObj);

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

String formattedDate = myDateObj.format(myFormatObj);

System.out.println("After Formatting: " + formattedDate);

}

}

Outputnya adalah:

Before Formatting: 2019-12-15T00:08:37.161684   
After Formatting: 15/12/2019 00:08:37

* *dd-MMM-yyyy (*29-Sep-1988*)*

Contoh :

import java.time.LocalDateTime; // Import the LocalDateTime class

import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class MyClass {

public static void main(String[] args) {

LocalDateTime myDateObj = LocalDateTime.now();

System.out.println("Before Formatting: " + myDateObj);

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MMM-yyyy HH:mm:ss");

String formattedDate = myDateObj.format(myFormatObj);

System.out.println("After Formatting: " + formattedDate);

}

}

Outputnya adalah:

Before Formatting: 2019-12-15T00:08:41.312324   
After Formatting: 15-Dec-2019 00:08:41

* *E, MMM dd yyyy (*Thu, Sep 29 1988*)*

Contoh :

import java.time.LocalDateTime; // Import the LocalDateTime class

import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class MyClass {

public static void main(String[] args) {

LocalDateTime myDateObj = LocalDateTime.now();

System.out.println("Before Formatting: " + myDateObj);

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("E, MMM dd yyyy HH:mm:ss");

String formattedDate = myDateObj.format(myFormatObj);

System.out.println("After Formatting: " + formattedDate);

}

}

Outputnya adalah:

Before Formatting: 2019-12-15T00:08:43.941490   
After Formatting: Sun, Dec 15 2019 00:08:43

## 2.17 Analisa Hasil Percobaan Java ArrayList

Kelas ArrayList adalah [array](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=en&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_arrays.asp&xid=17259,15700021,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhiWBp5Ez-N_jsk4r5yJhTUsZPF0fw) resizable, yang dapat ditemukan dalam paket java.util .

Perbedaan antara array bawaan dan ArrayList di Java, adalah bahwa ukuran array tidak dapat dimodifikasi (jika Anda ingin menambah atau menghapus elemen ke / dari array, Anda harus membuat yang baru). Sementara elemen dapat ditambahkan dan dihapus dari ArrayList kapan pun Anda mau. Sintaksnya juga sedikit berbeda:

Contoh

Buat objek ArrayList disebut **mobil** yang akan menyimpan string:

import java.util.ArrayList; // import the ArrayList class ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object

* **Tambahkan Item**

Kelas ArrayList memiliki banyak metode yang berguna. Misalnya, untuk menambahkan elemen ke ArrayList , gunakan metode add() :

Contoh

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

System.out.println(cars);

}

}

}

Outputnya adalah:

[Volvo, BMW, Ford, Mazda]

* **Akses suatu Item**

Untuk mengakses elemen di ArrayList , gunakan metode get() dan lihat nomor indeks:

Contoh

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

System.out.println(cars.get(0));

}

}

Outputnya adalah:

Volvo

**Ingat:** Indeks array dimulai dengan 0: [0] adalah elemen pertama. [1] adalah elemen kedua, dll.

* **Ubah Item**

Untuk memodifikasi elemen, gunakan metode set() dan lihat nomor indeks:

Contoh

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

cars.set(0, "Opel");

System.out.println(cars);

}

}

Outputnya adalah:

[Opel, BMW, Ford, Mazda]

* **Hapus Item**

Untuk menghapus elemen, gunakan metode remove() dan lihat nomor indeks:

Contoh

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

cars.remove(0);

System.out.println(cars);

}

}

Outputnya adalah:

[BMW, Ford, Mazda]

Untuk menghapus semua elemen di ArrayList , gunakan metode clear() :

Contoh

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

cars.clear();

System.out.println(cars);

}

}

Outputnya adalah:

[]

* **Ukuran ArrayList**

Untuk mengetahui berapa banyak elemen yang dimiliki ArrayList, gunakan metode size :

Contoh

cars.size();

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

System.out.println(cars.size());

}

}

Outputnya adalah:

4

* **Loop Through ArrayList**

Loop melalui elemen ArrayList dengan for , dan gunakan metode size() untuk menentukan berapa kali loop harus dijalankan:

Contoh

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

for (int i = 0; i < cars.size(); i++) {

System.out.println(cars.get(i));

}

}

}

Outputnya adalah:

Volvo

BMW

Ford

Mazda

Anda juga dapat mengulang melalui ArrayList dengan **untuk masing-masing** loop:

Contoh

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

for (String i : cars) {

System.out.println(i);

}

}

}

Outputnya adalah:

Volvo

BMW

Ford

Mazda

* **Tipe yang lain**

Elemen dalam ArrayList sebenarnya adalah objek. Dalam contoh di atas, kami membuat elemen (objek) dari tipe "String". Ingatlah bahwa String di Java adalah objek (bukan tipe primitif). Untuk menggunakan tipe lain, seperti int, Anda harus menentukan [kelas pembungkus yang](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=en&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_wrapper_classes.asp&xid=17259,15700021,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjTkKWbPZOkNjzN2n4-pUQTjguBmQ) setara: Integer . Untuk jenis primitif lainnya, gunakan: Boolean untuk boolean, Character untuk char, Double untuk double, dll:

Contoh

Buat ArrayList untuk menyimpan angka (tambahkan elemen bertipe Integer ):

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<Integer> myNumbers = new ArrayList<Integer>();

myNumbers.add(10);

myNumbers.add(15);

myNumbers.add(20);

myNumbers.add(25);

for (int i : myNumbers) {

System.out.println(i);

}

}

}

Outputnya adalah:

10

15

20

25

* **Sortir Daftar Array**

Kelas lain yang berguna dalam paket java.util adalah kelas Collections , yang mencakup metode sort() untuk menyortir daftar berdasarkan abjad atau angka:

Contoh

Urutkan Daftar Array dari String:

import java.util.ArrayList;

import java.util.Collections; // Import the Collections class

public class MyClass {

public static void main(String[] args) {

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

Collections.sort(cars); // Sort cars

for (String i : cars) {

System.out.println(i);

}

}

}

Outputnya adalah:

BMW

Ford

Mazda

Volvo

* **Sortir ArrayList of Integer:**

import java.util.ArrayList;

import java.util.Collections; // Import the Collections class

public class MyClass {

public static void main(String[] args) {

ArrayList<Integer> myNumbers = new ArrayList<Integer>();

myNumbers.add(33);

myNumbers.add(15);

myNumbers.add(20);

myNumbers.add(34);

myNumbers.add(8);

myNumbers.add(12);

Collections.sort(myNumbers); // Sort myNumbers

for (int i : myNumbers) {

System.out.println(i);

}

}

}

Outputnya adalah:

8

12

15

20

33

34

## 2.18 Analisa Hasil Percobaan Java HashMap

Anda belajar dari bab sebelumnya, bahwa Array menyimpan item sebagai koleksi yang dipesan, dan Anda harus mengaksesnya dengan nomor indeks (tipe int ). Namun, HashMap , menyimpan item dalam pasangan " **kunci** / **nilai** ", dan Anda dapat mengaksesnya dengan indeks jenis lain (mis. String ).

Satu objek digunakan sebagai kunci (indeks) ke objek lain (nilai). Ini dapat menyimpan berbagai jenis: Kunci String dan nilai Integer , atau tipe yang sama, seperti: Kunci String dan nilai String :

Contoh

Buat objek HashMap disebut **capitalCities** yang akan menyimpan **kunci** String dan **nilai-nilai** String :

import java.util.HashMap; // import the HashMap class HashMap<String, String> capitalCities = new HashMap<String, String>();

* **Tambahkan Item**

Kelas HashMap memiliki banyak metode yang berguna. Misalnya, untuk menambahkan item ke dalamnya, gunakan metode put() :

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

System.out.println(capitalCities);

}

}

Outputnya adalah:

{USA=Washington DC, Norway=Oslo, England=London, Germany=Berlin}

* **Akses suatu Item**

Untuk mengakses nilai di HashMap , gunakan metode get() dan lihat kuncinya:

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

System.out.println(capitalCities.get("England"));

}

}

Outputnya adalah:

London

* **Hapus Item**

Untuk menghapus item, gunakan metode remove() dan lihat kunci:

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

capitalCities.remove("England");

System.out.println(capitalCities);

}

}

Outputnya adalah:

{USA=Washington DC, Norway=Oslo, Germany=Berlin}

Untuk menghapus semua item, gunakan metode clear() :

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

capitalCities.clear();

System.out.println(capitalCities);

}

}

Outputnya adalah:

{}

* **Ukuran HashMap**

Untuk mengetahui berapa banyak item yang ada, gunakan metode size :

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

System.out.println(capitalCities.size());

}

}

Outputnya adalah:

4

* **Loop Through HashMap**

Ulangi item-item dari HashMap dengan **setiap** loop.

**Catatan:** Gunakan metode keySet() jika Anda hanya menginginkan kunci, dan gunakan metode values() jika Anda hanya menginginkan nilai:

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

for (String i : capitalCities.keySet()) {

System.out.println(i);

}

}

}

Outputnya adalah:

USA

Norway

England

Germany

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

for (String i : capitalCities.values()) {

System.out.println(i);

}

}

}

Outputnya adalah:

Washington DC  
Oslo  
London  
Berlin

Contoh

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

HashMap<String, String> capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

capitalCities.put("Germany", "Berlin");

capitalCities.put("Norway", "Oslo");

capitalCities.put("USA", "Washington DC");

for (String i : capitalCities.keySet()) {

System.out.println("key: " + i + " value: " + capitalCities.get(i));

}

}

}

Outputnya adalah:

key: USA value: Washington DC  
key: Norway value: Oslo  
key: England value: London  
key: Germany value: Berlin

* **Tipe yang lain**

Kunci dan nilai dalam HashMap sebenarnya adalah objek. Dalam contoh di atas, kami menggunakan objek bertipe "String". Ingatlah bahwa String di Java adalah objek (bukan tipe primitif). Untuk menggunakan tipe lain, seperti int, Anda harus menentukan [kelas pembungkus yang](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&rurl=translate.google.co.id&sl=auto&sp=nmt4&tl=id&u=https://www.w3schools.com/java/java_wrapper_classes.asp&xid=17259,15700021,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhh0NgnvWGvhkFWK1063FsvRUyKU4Q) setara: Integer . Untuk jenis primitif lainnya, gunakan: Boolean untuk boolean, Character untuk char, Double untuk double, dll:

Contoh

Buat objek HashMap disebut **orang** yang akan menyimpan **kunci** String dan **nilai** Integer :

// Import the HashMap class

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

// Create a HashMap object called people

HashMap<String, Integer> people = new HashMap<String, Integer>();

// Add keys and values (Name, Age)

people.put("John", 32);

people.put("Steve", 30);

people.put("Angie", 33);

for (String i : people.keySet()) {

System.out.println("Name: " + i + " Age: " + people.get(i));

}

}

}

Outputnya adalah:

Name: Angie Age: 33

Name: Steve Age: 30

Name: John Age: 32

## 2.19 Analisa Hasil Percobaan Java Wrapper Classes

**Kelas Java Wrapper**

Kelas wrapper menyediakan cara untuk menggunakan tipe data primitif ( int , boolean , dll.) Sebagai objek.

Tabel di bawah ini menunjukkan tipe primitif dan kelas pembungkus yang setara:

|  |  |
| --- | --- |
| **Primitive Data Type** | **Wrapper Class** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |
| char | Character |

Terkadang Anda harus menggunakan kelas wrapper, misalnya saat bekerja dengan objek Koleksi, seperti ArrayList , di mana tipe primitif tidak dapat digunakan (daftar hanya dapat menyimpan objek):

Contoh

import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<Integer> myNumbers = new ArrayList<Integer>();

myNumbers.add(10);

myNumbers.add(15);

myNumbers.add(20);

myNumbers.add(25);

for (int i : myNumbers) {

System.out.println(i);

}

}

}

Outputnya adalah:

10  
15  
20  
25

* Membuat Objek Wrapper

Untuk membuat objek wrapper, gunakan kelas wrapper alih-alih tipe primitif. Untuk mendapatkan nilai, Anda cukup mencetak objek:

Contoh

public class MyClass {

public static void main(String[] args) {

Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';

System.out.println(myInt);

System.out.println(myDouble);

System.out.println(myChar);

}

}

Outputnya adalah:

5  
5.99  
A

Karena Anda sekarang bekerja dengan objek, Anda dapat menggunakan metode tertentu untuk mendapatkan informasi tentang objek tertentu.

Misalnya, metode berikut digunakan untuk mendapatkan nilai yang terkait dengan objek pembungkus yang sesuai: intValue() , byteValue() , shortValue() , longValue() , floatValue() , doubleValue() , charValue() , booleanValue() .

Contoh ini akan menampilkan hasil yang sama seperti contoh di atas:

Contoh

public class MyClass {

public static void main(String[] args) {

Integer myInt = 5;

Double myDouble = 5.99;

Character myChar = 'A';

System.out.println(myInt.intValue());

System.out.println(myDouble.doubleValue());

System.out.println(myChar.charValue());

}

}

Outputnya adalah:

5  
5.99  
A

Metode lain yang bermanfaat adalah metode toString() , yang digunakan untuk mengubah objek pembungkus menjadi string.

Dalam contoh berikut, kami mengonversi Integer ke String , dan menggunakan metode length() dari kelas String untuk menampilkan panjang "string":

Contoh

public class MyClass {

public static void main(String[] args) {

Integer myInt = 100;

String myString = myInt.toString();

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

}

}

Outputnya adalah:

3

## 2.19 Analisa Hasil Percobaan Java Wrapper Classes

**Pengecualian Java**

Saat menjalankan kode Java, kesalahan yang berbeda dapat terjadi: kesalahan pengkodean yang dibuat oleh programmer, kesalahan karena input yang salah, atau hal-hal yang tidak terduga lainnya.

Ketika kesalahan terjadi, Java biasanya akan berhenti dan menghasilkan pesan kesalahan. Istilah teknis untuk ini adalah: Java akan melempar **pengecualian** (melempar kesalahan).

* **Java mencoba dan menangkap**

Pernyataan try memungkinkan Anda untuk menentukan blok kode yang akan diuji untuk kesalahan saat sedang dieksekusi.

Pernyataan catch memungkinkan Anda untuk menentukan blok kode yang akan dieksekusi, jika kesalahan terjadi di blok coba.

Kata kunci try dan catch berpasangan:

* **Sintaksis**

try { // Block of code to try } catch(Exception e ) { // Block of code to handle errors }

Perhatikan contoh berikut:

Ini akan menghasilkan kesalahan, karena **myNumbers [10]** tidak ada.

public class MyClass {

public static void main(String[] args) {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

}

}

Outputnya akan seperti ini:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
        at MyClass.main(MyClass.java:4)

Jika terjadi kesalahan, kita bisa menggunakan try...catch untuk menangkap kesalahan dan menjalankan beberapa kode untuk menanganinya:

Contoh

public class MyClass {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

} catch (Exception e) {

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

}

}

}

Outputnya adalah:

Something went wrong.

* **Akhirnya**

Pernyataan finally memungkinkan Anda menjalankan kode, setelah try...catch , terlepas dari hasilnya:

Contoh

public class MyClass {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

} catch (Exception e) {

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

} finally {

System.out.println("The 'try catch' is finished.");

}

}

}

Outputnya adalah:

Something went wrong.  
The 'try catch' is finished.

* **Kata kunci lemparan**

Pernyataan throw memungkinkan Anda untuk membuat kesalahan khusus.

Pernyataan throw digunakan bersama dengan **tipe pengecualian** . Ada banyak jenis pengecualian yang tersedia di Jawa: ArithmeticException , FileNotFoundException , ArrayIndexOutOfBoundsException , SecurityException , dll:

Contoh

Lempar pengecualian jika **usia di** bawah 18 (cetak "Akses ditolak"). Jika usia 18 tahun atau lebih, cetak "Akses diberikan":

public class MyClass {

static void checkAge(int age) {

if (age < 18) {

throw new ArithmeticException("Access denied - You must be at least 18 years old.");

} else {

System.out.println("Access granted - You are old enough!");

}

}

public static void main(String[] args) {

checkAge(15);

}

}

Outputnya adalah:

Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.  
        at MyClass.checkAge(MyClass.java:4)  
        at MyClass.main(MyClass.java:12)

Jika **usia** 20 tahun, Anda **tidak** akan mendapatkan pengecualian:

Contoh

public class MyClass {

static void checkAge(int age) {

if (age < 18) {

throw new ArithmeticException("Access denied - You must be at least 18 years old.");

} else {

System.out.println("Access granted - You are old enough!");

}

}

public static void main(String[] args) {

checkAge(20);

}

}

Outputnya adalah:

Access granted - You are old enough!

Top of Form

## Uji Diri Anda Dengan Latihan

**Olahraga:**

Masukkan bagian yang hilang untuk menangani kesalahan dalam kode di bawah ini.

 {

int [] myNumbers = {1, 2, 3};

System.out.println (myNumbers [10]);

}  (Pengecualian e) {

System.out.println ("Ada yang salah.");

}

Bottom of Form

**BAB V**

## KESIMPULAN

Pemrograman Berorientasi Objek ( PBO ) dalam bahasa inggris dikenal dengan Object Oriented Programming adalah sebuah paradigma pemrograman yang memandang penyelesaian masalah berdasarkan objek yang di targetkan atau hendak di buat. Pemrograman ini digunakan dalam beberapa bahasa Pemrograman seperti C++, Pascal, Java, dan lain lain. Pemrograman berorientasi objek sangat kental kaitannya dengan bahasa pemrograman Java, karena java adalah bahasa pemrograman yang menitik beratkan pada Objeck Oriented Programming atau Pemrograman Berorientasi Objek.

## SARAN

Semoga praktikum bahasa pemograman java lebih di tingkatkan lagi. Agar mahasiswa dapat membangun sebuah aplikasi atau sistem informasi yang kemudian bisa diterapkan dalam kehidupan kebutuhan informasi.

Akhirnya penulis mengucapkan banyak terima kasih kepada semua pihak yang telah membantu hingga terselesaikannya laporan praktikum ini, penulis juga mengucapkan terima kasih kepada Bapak M. Imron Rosadi, S.Kom., M.Kom. selaku dosen pembimbing praktikum yang telah membimbing kami selama praktikum pemrograman java ini berlangsung di Universitas Yudharta.

**Daftar pustaka** : https://www.w3schools.com/java