### **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 Bahasa Pemrograman Java (OOP) ini dengan baik. Penulisan laporan ini dibuat agar dapat memenuhi sebagi salah satu syarat untuk mendapatkan nilai tambahan serta dalam 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.

3. Bapak Imron Rosadi S.Kom, selaku dosen Bahasa Pemrograman Java (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, 12 Desember 2019

Penulis

Fahriza Aulia.

### **DAFTAR ISI**

[**KATA PENGANTAR** i](#_Toc29844986)

[**DAFTAR ISI** ii](#_Toc29844987)

[**BAB I** 1](#_Toc29844988)

[**PENDAHULUAN** 1](#_Toc29844989)

[**A.** **Latar Belakang** 1](#_Toc29844990)

[**B.** **Rumusan Masalah** 1](#_Toc29844991)

[**C.** **Tujuan Praktikum** 2](#_Toc29844992)

[**BAB II** 3](#_Toc29844993)

[**LANDASAN TEORI** 3](#_Toc29844994)

[**A.** **Konsep OOP** 3](#_Toc29844995)

[**B.** **Pengenalan Objek dan Class** 4](#_Toc29844996)

[**C.** **Java Class Attribute** 5](#_Toc29844997)

[**D.** **Java Class Methods** 8](#_Toc29844998)

[**E.** **Java Constructor** 14](#_Toc29844999)

[**F.** **Java Modifier** 17](#_Toc29845000)

[**G.** **Java Encapsulation** 25](#_Toc29845001)

[**H.** **Java API/ Packages** 27](#_Toc29845002)

[**I.** **Java Inheritance** 30](#_Toc29845003)

[**J.** **Java Polymorishm** 32](#_Toc29845004)

[**K.** **Java Inner Classes** 34](#_Toc29845005)

[**L.** **Java Abstraction** 38](#_Toc29845006)

[**M.** **Java Interface** 40](#_Toc29845007)

[**N.** **Java Enums** 43](#_Toc29845008)

[**O.** **Java User Input** 46](#_Toc29845009)

[**P.** **Java Date** 49](#_Toc29845010)

[**Q.** **Java Array List** 55](#_Toc29845011)

[**R.** **Java HashMap** 63](#_Toc29845012)

[**S.** **Java Wrapper Classes** 68](#_Toc29845013)

[**T.** **Java Exceptions** 71](#_Toc29845014)

[**BAB III** 74](#_Toc29845015)

[**HASIL PERCOBAAN** 74](#_Toc29845016)

[**A.** **Classes/ Objects** 74](#_Toc29845017)

[**B.** **Attributes** 75](#_Toc29845018)

[**C.** **Method** 79](#_Toc29845019)

[**D.** **Construktors** 80](#_Toc29845020)

[**E.** **Modifiers** 81](#_Toc29845021)

[**F.** **Encapsulation** 89](#_Toc29845022)

[**G.** **Package/ API** 91](#_Toc29845023)

[**H.** **Inherintence** 93](#_Toc29845024)

[**I.** **Pholimorphism** 94](#_Toc29845025)

[**J.** **Inner Class** 94](#_Toc29845026)

[**K.** **Abstraction** 97](#_Toc29845027)

[**L.** **Interface** 98](#_Toc29845028)

[**M.** **Enums** 99](#_Toc29845029)

[**N.** **Use Input** 101](#_Toc29845030)

[**O.** **Java Date** 103](#_Toc29845031)

[**P.** **Java Array List** 106](#_Toc29845032)

[**Q.** **Java HashMap** 108](#_Toc29845033)

[**R.** **Java Wrapper** 109](#_Toc29845034)

[**S.** **Java Exceptions** 109](#_Toc29845035)

[**BAB IV** 110](#_Toc29845036)

[**STUDI KASUS dan ANALISIS** 110](#_Toc29845037)

[**A.** **Java Class and Objects** 110](#_Toc29845038)

[**B.** **Java Class Attributes** 111](#_Toc29845039)

[**C.** **Java Class Methods** 111](#_Toc29845040)

[**D.** **Java Constructors** 112](#_Toc29845041)

[**E.** **Java Modifiers** 113](#_Toc29845042)

[**F.** **Java Encapsulation** 115](#_Toc29845043)

[**G.** **Java Package/API** 116](#_Toc29845044)

[**H.** **Java Inherintence** 122](#_Toc29845045)

[**I.** **Java Polymorphism** 128](#_Toc29845046)

[**J.** **Java Inner Class** 129](#_Toc29845047)

**[K.](#_Toc29845048)****[Java Abstrak](#_Toc29845048)** [132](#_Toc29845048)

[**L.** **Java Interface** 134](#_Toc29845049)

[**M.** **Java Enums** 135](#_Toc29845050)

[**N.** **Java User Input (Scanner)** 137](#_Toc29845051)

[**O.** **Java Date** 138](#_Toc29845052)

[**P.** **Java ArrayList** 141](#_Toc29845053)

[**Q.** **Java HashMap** 143](#_Toc29845054)

[**R.** **Java Wrapper** 144](#_Toc29845055)

[**S.** **Java Exceptions** 145](#_Toc29845056)

[**BAB V** 147](#_Toc29845057)

[**PENUTUP** 147](#_Toc29845058)

[**DAFTAR PUSTAKA** 148](#_Toc29845059)

### **BAB I**

### **PENDAHULUAN**

### **Latar Belakang**

Pemrograman Berorientasi Objek (*Object Oriented Programming* atau OOP) merupakan paradigm pemrograman yang berorientasikan kepada objek. Objek adalah struktur data yang terdiri dari bidang data dan metode bersama dengan interaksi mereka untu merancang aplikasi dan program computer. Semua data dan fungsi di dalam paradigm ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya. Pada jaman sekarang, banyak bahasa pemrograman yang mendukung OOP.

OOP adalah paradigm pemrograman yang cukup mendominasi saat ini, karena mampu memberikan solusi kaidah pemrograman modern. Meskipun demikian, bukan berarti bahwa pemrogaman procedural sudah tidak layak lagi. OOP diciptakan karena dirasakan masih adanya keterbatasan pada bahasa pemrograman tradisional. Konsep OOP sendiri adalah semua pemecahan masalah dibagi kedalam objek. Dalam OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek. Pembahasan mengenai orientasi objek tidak akan terlepas dari konsep objek seperti *inheritance* atau pewarisan, *encapsulation* atau pembungkusan, dan *polymorphism* atau kebanyakrupaan. Konsep-konsep ini merupakan fundamental dalam orientasi objek yang perlu dipahami serta digunakan dengan baik, dan menghindari penggunaanya yang tidak tepat.

OOP lebih mudah dipelajari bagi pemula dibandingkan dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

### **Rumusan Masalah**

1. Bagaimana pengertian OOP
2. Apa saja jenis-jenis yang ada dalam OOP
3. Bagaimana cara menjalankan program yang ada dalam OOP

### **Tujuan Praktikum**

1. Mengetahui pengertian OOP
2. Untuk mengetahui jeni-jenis OOP
3. Untuk megetahui cara menjalankan program pada OOP

### **BAB II**

### **LANDASAN TEORI**

### **Konsep OOP**

OOP adalah singkatan **dari Pemrograman Berorientasi Objek** .

Pemrograman prosedural adalah tentang prosedur atau metode penulisan yang melakukan operasi pada data, sedangkan pemrograman berorientasi objek adalah tentang membuat objek yang berisi data dan metode.

Pemrograman berorientasi objek memiliki beberapa keunggulan dibandingkan pemrograman prosedural:

* OOP lebih cepat dan lebih mudah untuk dieksekusi.
* OOP menyediakan struktur yang jelas untuk program-program tersebut.
* OOP membantu menjaga kode Java KERING "Don't Repeat Yourself", dan membuat kode lebih mudah untuk mempertahankan, memodifikasi dan men-debug.
* OOP memungkinkan untuk membuat aplikasi yang dapat digunakan kembali secara penuh dengan kode yang lebih sedikit dan waktu pengembangan yang lebih singkat.

Kelas dan objek adalah dua aspek utama dari pemograman berorientasi objek. Lihatlah ilustrasi berikut untuk melihat perbedaan antara kelas dan objek:

|  |  |
| --- | --- |
| KELAS | OBJEK |
| Buah | Apel |
|  | Mangga |
|  | Pisang |

Jadi, kelas adalah template untuk objek, dan objek adalah turunan dari kelas. Ketika objek individu dibuat, mereka mewarisi semua variabel dan metode dari kelas.

### **Pengenalan Objek dan Class**

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

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

* Mendeklarsaikan Kelas

Sebuah kelas dideklarasikan dengan menggunakan kata keyword ***class***. kelas ini merupakan objek template yang berisi spesifikasi umum dari suatu objek

Membuat kelas :

public class MyClass {

int x = 5;

}

* Mendeklarasikan objek

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 keyword **new**

public class MyClass {

int x = 5;

public static void main(String[] args) {

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

System.out.println(myObj.x);

}

}

Output:

5

### **Java Class Attribute**

Attributes merupakan nilai data  yang  terdapat pada  suatu object  yang berasal dari class. Attributes merepresentasikan karakteristik dari suatu object. Atau Atribut adalah sesuatu yang dimiliki oleh objek.  Setiap objek yang dibuat dari kelas yang sama akan memiliki atribut yang sama.

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:

public class MyClass {

int x = 5;

int y = 3;

}

* Mengakses Attribute

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:

public class MyClass {

int x = 5;

public static void main(String[] args) {

MyClass myObj = new MyClass();

System.out.println(myObj.x);

}

}

Outputan:

5

* Mengubah Nilai Attribute
* Menetapkan nilai attribute

Example kita mengubah nilai x menjadi 40

public class MyClass {

int x;

public static void main(String[] args) {

MyClass myObj = new MyClass();

myObj.x = 40;

System.out.println(myObj.x);

}

}

Output:

40

* Mengubah nilai yang ada menjadi nilai baru

Example mengubah nilai 10 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);

}

}

Output:

25

* Jika ingin membuat banyak objek dalam satu kelas maka cukup mengubah nilai attribute di sabtu objek ,tanpa mengubah nilai attribute di yang lain :

Example

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

}

}

Output :

5

25

* Banyak Attribute

Anda juga bisa menentukan banyak attribute sebanyak yang anda inginkan

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);

}

}

Output :

Jhon Doe

24

### **Java Class Methods**

Method merupakan suatu operasi berupa fungsi-fungsi yang dapat dikerjakan oleh suatu object. Method didefinisikan pada class akan tetapi dipanggil melalui object. Method menentukan prilaku objeck, akni apa yang terjadi ketika objeck itu di buat serta berbagai operasi yang dapat dilakukan objrk sepanjang hidupnya.

* Mengakses Methods

metode dideklarasikan dalam kelas, dan digunakan untuk melakukan tindakan tertentu:

Example :

Buatlah methods bernama myMethod() di Myclass:

public class MyClass {

static void myMethod() {

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

}

}

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

public class MyClass {

static void myMethod() {

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

}

public static void main(String[] args) {

myMethod();

}

}

// Outputs

"Hello World!"

* Method di bedakan menjadi 2 yaitu :
* Static

Staic yaitu dapat diakses tanpa membuat objek kelas.

* Non Static

Non Static yaitu hanya dapat diakses oleh objek itu sendiri.

Dalam contoh di atas, kami membuat metode static , yang berarti dapat diakses tanpa membuat objek kelas, tidak seperti public , yang hanya dapat diakses oleh objek:

* Metode Akses dengan Objeck

Contoh :

Buatlah method dengan kelas bernama **myCar** . Panggil method **fullThrottle();** dan **speed();** pada objek myCar

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

}

}

Output:

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

***Contoh* Scoude Code :**

// Create a Car class

public class Car {

// Create a fullThrottle() method

public void fullThrottle() {

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

}

// Create a speed() method and add a parameter

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

}

}

Output:

The car is going as fast as it can!

Max speed is: 200

**Penjelasan dari contoh di atas :**

1) Kami menciptakan kelas Car khusus dengan kata kunci class

2) Kami menciptakan metode fullThrottle() dan speed() di kelas Car .

3) Metode fullThrottle() dan metode speed() akan mencetak 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() .

* **Catatan Penting :**

1. Titik (.) Digunakan untuk mengakses atribut dan metode objek.
2. Untuk memanggil metode di Java, tulis nama metode diikuti dengan seperangkat tanda kurung () , diikuti dengan tanda titik koma ( ; ).
3. Kelas harus memiliki nama file yang cocok ( Car dan Car.java )

Untuk mengakses method dengan menggunakan objek kita bisa menggunakan banyak kelas. Dan yang harus di ingat nama file harus sesuai dengan nama kelas

Dari contoh kelas Car di atas kita coba menggunakan banyak kelas :

***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

}

}

Maka Outputan nya akan sama:

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

### **Java Constructor**

Konstruktor di Jawa 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

Buat konstruktor:

// Create a MyClass class

public class MyClass {

int x; // Create a class attribute

// Create a **class constructor** for the MyClass class

public MyClass() {

x = 5; // Set the initial value for the class attribute x

}

public static void main(String[] args) {

MyClass myObj = new MyClass(); // Create an object of class MyClass (This will **call the constructor**)

System.out.println(myObj.x); // Print the value of x

}

}

// Outputs 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.

* Constructor Parameters

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:

Contoh:

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);

}

}

// Outputs 5

Anda dapat memiliki banyak parameter yang Anda inginkan:

Contoh

public class Car {

int modelYear;

String modelName;

public Car(int year, String name) {

modelYear = year;

modelName = name;

}

public static void main(String[] args) {

Car myCar = new Car(1969, "Mustang");

System.out.println(myCar.modelYear + " " + myCar.modelName);

}

}

// Outputs 1969 Mustang

### **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
* Access Modifiers.
* Untuk **kelas** , Anda dapat menggunakan *public* atau default :
* **Modifiers** : Public

**Deskripsi** : kelas ini diakses oleh lain kelas

Contoh :

public class MyClass {

public static void main(String[] args) {

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

}

}

Output:

Hello World

* **Modifiers :** Default

**Deskripsi :** Kelas ini hanya diakses oleh kelas di sama paket. Ini adalah digunakan ketika anda tidak specify sebuah modifier. Anda akan belajar lebih banyak tentang packages di packages bab.

Contoh :

class MyClass {

public static void main(String[] args) {

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

}

}

Output:

Hello World

* Untuk **atribut, metode, dan konstruktor** , Anda dapat menggunakan salah satu dari yang berikut ini:
* **Modifiers** : Public

**Deskripsi**  : Kode adalah di akses untuk semua kelas

Contoh :

class MyClass {

public static void main(String[] args) {

Person myObj = new Person();

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

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

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

}

}

Output :

Name: John Doe  
Email: john@doe.com  
Age: 24

* **Modifiers** : Private

**Deskripsi**  : Kode ini m,hanya diakses dalam menyatakan kelas

Contoh :

public class Person {

private String fname = "John";

private String lname = "Doe";

private String email = "john@doe.com";

private int age = 24;

public static void main(String[] args) {

Person myObj = new Person();

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

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

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

}

}

Output:

Name: John Doe  
Email: john@doe.com  
Age: 24

* **Modifiers** : Default

**Deskripsi**  : Kode tidak hanya diakses disama paket. Ini adalah digunakan ketika anda tidah spesifysebuah modifier. Anda akan belajar lebih banyak tentang packages di sebuah packages bab

Contoh :

class Person {

String fname = "John";

String lname = "Doe";

String email = "john@doe.com";

int age = 24;

public static void main(String[] args) {

Person myObj = new Person();

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

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

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

}

}

Output:

Name: John Doe  
Email: john@doe.com  
Age: 24

* **Modifiers** : Protected

**Deskripsi**  : Kode tidak hanya diakses disama paket dan subclasses. Anda akan belajar lebih banyak tentang sublasses dan sup~erclasses di sebuah inheritance bab.

Contoh :

class Person {

protected String fname = "John";

protected String lname = "Doe";

protected String email = "john@doe.com";

protected int age = 24;

}

class Student extends Person {

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);

}

}

Output:

Name: John Doe  
Email: john@doe.com  
Age: 24  
Graduation Year: 2018

* Non-Access Modifiers
* Untuk **kelas** , Anda dapat menggunakan final atau abstract :

|  |  |
| --- | --- |
| **Modifiers** | **Deskripsi** |
| Final | Kelas tidak dapat diwarisi oleh kelas lain (Anda akan belajar lebih banyak tentang warisan dalam bab Warisan) |
| Abstract | Kelas tidak dapat digunakan untuk membuat objek (Untuk mengakses kelas abstrak, itu harus diwarisi dari kelas lain. Anda akan belajar lebih banyak tentang pewarisan dan abstraksi di bab Warisan dan Abstraksi) |

* Untuk **atribut dan metode** , Anda dapat menggunakan salah satu dari yang berikut:

|  |  |
| --- | --- |
| **Modifiers** | **Deskripsi** |
| Final | Atribut dan metode tidak dapat diganti atau dimodifikasi |
| Static | Atribut dan metode milik kelas, bukan objek |
| Abstract | Hanya dapat digunakan dalam kelas abstrak, dan hanya dapat digunakan pada metode. Metode ini tidak memiliki tubuh, misalnya abstrak void run (); . Tubuh disediakan oleh subclass (diwarisi dari). Anda akan belajar lebih banyak tentang warisan dan abstraksi dalam bab-bab Warisan dan Abstraksi |
| Transient | Atribut dan metode dilewati saat membuat serialisasi objek yang memuatnya |
| Synchronized | Metode hanya dapat diakses oleh satu utas pada satu waktu |
| Volatile | Nilai atribut tidak di-cache thread-lokal, dan selalu dibaca dari "memori utama" |

* Final

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

Contoh :

public class MyClass {

**final** int x = 10;

**final** double PI = 3.14;

public static void main(String[] args) {

MyClass myObj = new MyClass();

// 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);

System.out.println(myObj.PI);

}

}

Output:  
10

3,14

* Statis

Metode static berarti dapat diakses tanpa membuat objek kelas, tidak seperti public :

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 output an error

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

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

}

}

Output:

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

* Abstrak

Metode abstract milik kelas abstract , dan tidak memiliki tubuh. Tubuh disediakan oleh subclass:

Contoh:

// Code from filename: Person.java

// abstract class  
abstract class Person {

public String fname = "John";

public int age = 24;

public **abstract** void study(); // abstract method

}

// Subclass (inherit from Person)

class Student extends Person {

public int graduationYear = 2018;

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

System.out.println("Studying all day long");

}

}

// End code from filename: Person.java

// Code from filename: MyClass.java

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("Graduation Year: " + myObj.graduationYear);

myObj.study(); // call abstract method  
 }

}

Output:

Name: John  
Age: 24  
Graduation Year: 2018  
Studying all day long

### **Java Encapsulation**

Enkapsulasi (encapsulation) adalah untuk memastikan bahwa data “sensitive” disembunyikan dari pengguna. Artinya encapsulation pada java digunakan untuk membungkus class dan menjaga apa saja yang ada dalam class baik itu method maupun attribute , agar tidak dapat di akses oleh kelas lain. Untuk mencapai ini maka perlu diperhatikan hal-hal berikut ini :

* Mendeklarasikan variabel / attribute kelas sebagai Private
* Memberikan metode public **Get** dan **Set**  untuk mengakses dan memperbarui nilai suatu Private variable
* Get dan Set

Variabel Private hanya dapat diakses didalam kelas yang sama (kelas luar tidak memiliki akses). Namun, memungkinkan mengaksesnya jika diberi tambahan metode Public **Get** dan **Set.**

Get adalah metode mendapatkan nilai variabel , dan Set metode mengatur nilai.

Sintaks untuk keduanya adalah bawa dimulai dengan salah satu get atau set , diikuti oleh nama variabel , dengan huruf pertama menggunakan huruf kapital.

Berikut adalah Contoh dari encapsulation

Disini kita membuat 2 kelas yaitu kelas ‘main’ (1) & kelas yang menampilkan hasil ‘output’ (2)

public class Person {

private String name; // private = restricted access

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String newName) {

this.name = newName;

}

}

Metode Set artinya mengambil parameter (newName) dan mengisi data ke dalam objek ,kata this merupakan kunci yang digunakan untuk merujuk ke objek saat ini.

Variabel ‘nama’ dideklarasikan sebagai ‘public’ agar dapat mengaksesnya di luar kelas sedangkan jika kita mendeklarasikan dengan ‘private’ maka tidak dapat diakses diluar kelas.

1. *import getandset.Person;*

Kode di atas maksudnya adalah mengimpor class dan interface yang kita butuhkan. Mengimport fungsi get dan set pada kelas ‘main’.

1. *public class MyClass {*

menggunakan public karena akan di akses dari luar kelas

1. *public static void main(String[] args){*

*Person myObj = new Person();*

Digunakan untuk Menambahkan objek ke dalam kelas

1. *myObj.setName("John");*

Digunakan untuk mengisi data kedalam objek

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

Untuk memanggil objek

*}*

*}*

Maka outputannya adalah “John”. Kesimpulannya yaitu menggunakan metode getName() dan setName() untuk mengakses dan memperbarui variabel.

Tujuan dari Encapsulation antara lain :

* mengamankan atribut dan metode kelas dengan baik
* Atribut kelas dapat dibuat hanya-baca (jika hanya menggunakan get method) atau hanya-menulis (jika hanya menggunakan set method)
* Fleksibel artinya, Programmer dapat mengubah satu bagian kode tanpa mempengaruhi bagian yang lainnya.
* Peningkatan terhadap keamanan data.

### **Java API/ Packages**

Paket di Java digunakan untuk mengelompokkan kelas 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 Built-in (paket dari Java API)

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. Daftar lengkap dapat ditemukan di situs web Oracles: [*https://docs.oracle.com/javase/8/docs/api/*](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://docs.oracle.com/javase/8/docs/api/&xid=17259,15700021,15700186,15700190,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhiQjj-j2JCLVpyCTQLdaKKME_87ng) .

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** :

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

import *package*.*name*.\*; // Import the whole package

* Impor Kelas

Class yang bisa kita digunakan untuk mengambil input dari user, yaitu *Scanner,* *BufferedReader* dan *JOptionPane*, ketiga class tersebut mempunyai fungsi yang sama, akan tetapi pada JOptionPane, class ini memiliki tampilan visual atau GUI.

* **Class Scanner**

Untuk menggunakan Scanner, kita perlu membuat objek dari class Scanner, class tersebut berada didalam package *java.util.\**, jadi kita perlu menginport packagenya terlebih dahulu, setelah package dan objek selesai dibuat, selanjutnya kita membutuhkan sebuah variabel untuk menampung nilai yang di inputkan dari user

* + **Menyimpan Data**

Untuk menyimpan data String, kita bisa menggunakan method *next()* atau *nextLine()* pada Scanner, selain tipe data String, ada beberapa method lainnya yang bisa kalian gunakan pada Scanner, diantaranya: **nextInt** - Menerima input bilangan bulat/Integer

**nextBoolean** - Menerima input Boolean (true/false)

**nextFloat**  - Menerima input pecahan/Float

**nextDouble**  - Menerima input pecahan/Double

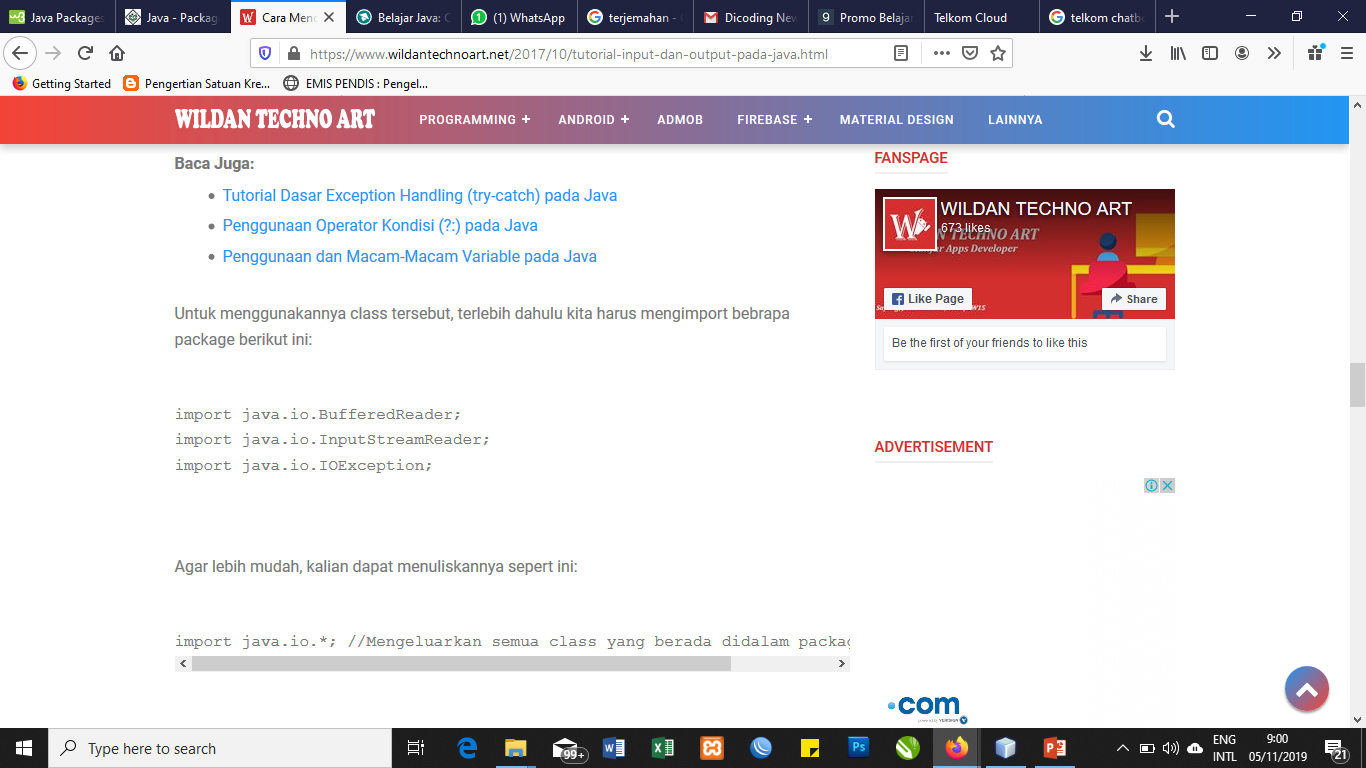
**nextByte** - Menerima bilangan bulat/Byte

**nextLong**  - Menerima bilangan bulat/Long

**nextShort**  - Menerima bilangan bulat/Short

* **Buffer Reader**

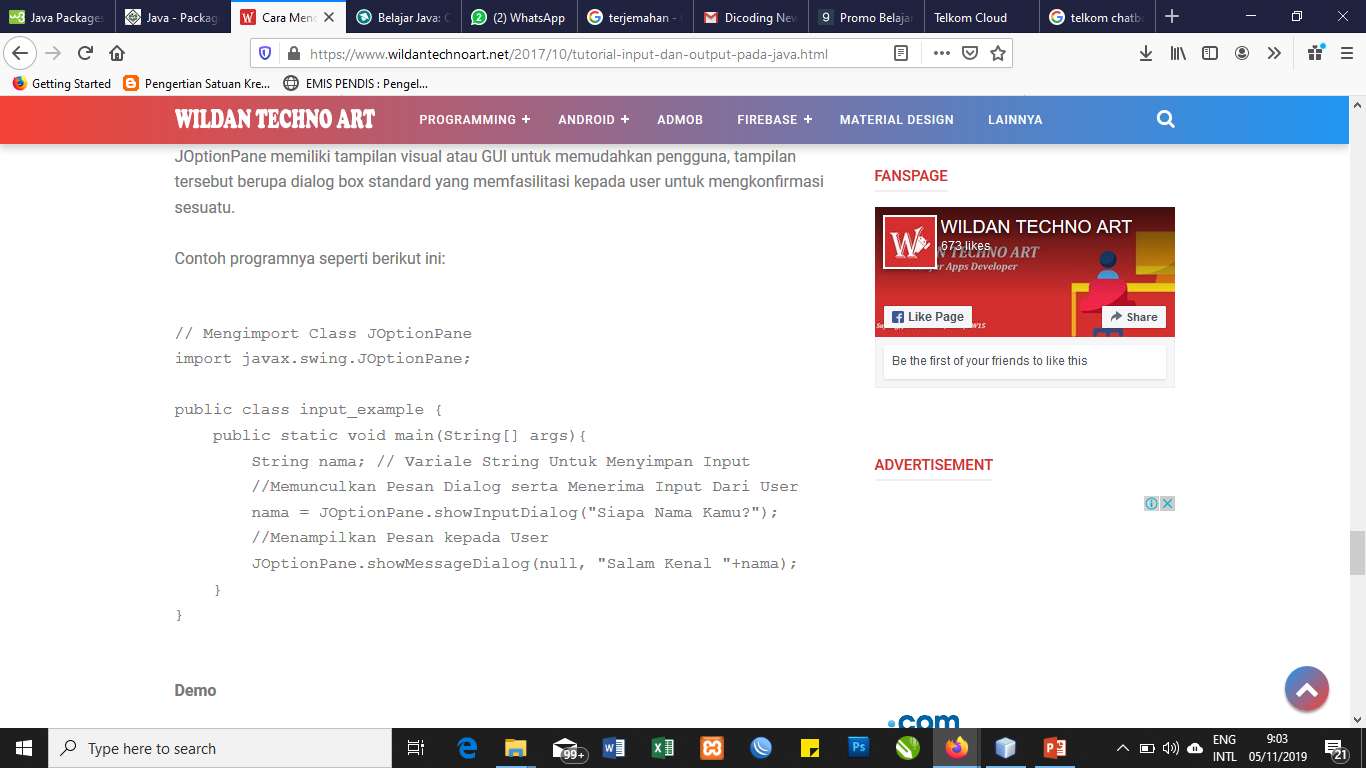
Pada bagian ini, kita akan menggunakan class BufferedReader, class tersebut berada di dalam package *java.io*, class ini dignakan untuk mendapatkan input dari user, hampir sama seperti Scanner, yang membedakan hanya cara penulisan syntaxnya saja.



* **Class JOptionPane**

Cara terakhir untuk mendapatkan input dari user adalah dengan menggunakan class JOptionPane calss tersebut terdapat pada package *javax.swing*. berbeda dari class sebelunya, JOptionPane memiliki tampilan visual atau GUI untuk memudahkan pengguna, tampilan tersebut berupa dialog box standard yang memfasilitasi kepada user untuk mengkonfirmasi sesuatu.

Contoh :



* **Class Console**

Class Console hampir sama dengan BufferReader. Dia juga menggunakan fungsi readLine() untuk mengambil input.

Akan tetapi…

Class ini hanya bisa digunakan di lingkungan console saja, seperti Terminal dan CMD.

Class Console tidak bisa digunakan langsung di Netbeans.

Maka dari itu, kita harus kompilasi secara manual.

Untuk menggunakan class ini, kita perlu mengimpornya terlebih dahulu.

### **Java Inheritance**

**Inheritance** (Pewarisan) adalah salah satu contoh [Konsep OOP](http://www.okedroid.com/search/label/Konsep%20OOP?max-results=5), yang merupakan suatu proses dimana , suatu class yang bisa disebut *super class* ,dapat mewarisi sifat atau ciri-ciri,  seperti atribut dan method, ke dalam class turunanya yaitu *sub class*. Super class akan mewarisi nilai dari atribut atau behavior ke Class turunanya atau class lain. Sistem penurunan sifatnya akan dilakukan secara hirarki (bertingkat-tingkat) , semakin ke bawah class, maka akan semakin detail.

* **subclass** (anak) - kelas yang mewarisi dari kelas lain.
* **superclass** (orang tua) - kelas yang diwarisi dari

Untuk mewarisi dari kelas, gunakan keyword extends yaitu sebagai kata kunci. Keyword **extends** adalah salah satu [jenis keyword](https://www.okedroid.com/2017/09/belajar-mengenal-jenis-keyword-pada-java.html) , di dalam bahasa pemrograman java, yang di gunakan untuk mewarisi sifat-sifat dari class parent atau super, seperti atribut dan method, ke dalam suatu class turunan.

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);

}

}

Output :

Tuut, tuut!  
Ford Mustang

* Keyword Final

Ketika suatu class dideklarasikan sebagai final maka class tersebut tidak bisa diwariskan atau mengakibatkan output menjadi eror. Beberapa tujuan dari kelas FINAL:

Ada tiga tujuan utama kenapa suatu class dibuat final. Tujuan tersebut adalah:

1. Untuk mencegah inheritance karena class final tidak bisa di extends.
2. Untuk membuat suatu class immutable. Class tidak akan bisa dibuat immutable jika sebelumnya class itu tidak final.
3. Class yang dibuat final sangat berguna ketika akan digunakan untuk tujuan tingkat keamanan tinggi dalam setiap aplikasi.

Contoh:

final class Vehicle {

...

}

class Car extends Vehicle {

...

}

Output :

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

* Jadi meng-exetend-kan class final akan menyebabkan error
* Ketika suatu class dideklarasikan sebagai final maka class tersebut tidak bisa diwariskan.
* Misalnya pada package diatas anda bisa melihat jika [**class String**](http://bahasajava.com/belajar-java-class-string-fundamental-dan-method/) adalah class final. Maksudnya class String tidak dapat dideklarasikan menjadi kelas final.

### **Java Polymorishm**

Polymorphism merupakan konsep sederhana dalam bahasa pemrograman berorientasi obyek yang berarti kemampuan sebuah obyek untuk menghasilkan aksi yang berbeda, bila method yang sama dipanggil, dimana aksi method yang dikerjakan tergantung dari tipe obyeknya. Polymorphism merupakan suatu konsep yang menyatakan sesuatu yang sama dapat memiliki berbagai bentuk dan perilaku berbeda. Jadi dapat disimpulkan bahwa polymorphism adalah satu objek yang memiliki banyak bentuk.

Konsep dari Polymorphism menggunakan 2 metode. Metode pertama yaitu Pewarisan yang menggunakan konsep overriding yaitu membuat method yang bernama dan berstruktur sama akan tetapi isi method berbeda pada class induk dan class anak. Metode yang kedua adalah overloading. Overloading adalah method bernama yang sama dengan method lain pada suatu class tetapi dengan parameter da nisi yang berbeda.

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();

}

}

Output :

The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow

* Keuntungan Polymorphism:
* Kita dapat menggunakan kelas-kelas yang kita buat (sebagai super kelas) dan membuat kelas kelas baru berdasar superkelas tersebut dengan karakteristik yang lebih khusus dari behaviour umum yang dimiliki superkelas.
* kita dapat membuat super kelas yang hanya mendefinisikan behaviuor namun tidak memberikan implementasidari metode-metode yang ada.Hal ini berguna jika kita ingin membuat semacam template kelas, kelas semacam ini disebut kelas abstrak karena behaviuor nya masih abstrak dan belum di implementasikan. Subkelas - subkelas dari kelas semacam ini yang disebut kelas konkret,mengimplementasikan behaviuor abstrak tersebut sesuai dengan kebutuhan masing-masing.
* Menghindari duplikasi object,kita dapat menciptakan class baru dari class yang sudah ada,sehingga tidak perlu menuliskan code dari nol ataupun mengulangnya, namun tetap bisa menambahkan attribute dan atau method unik dari class itu sendiri. Dalam konsep yang lebih umum sering kali polymorphism disebut dalam istilah satu interface banyak aksi.

### **Java Inner Classes**

Di dalam java, inner class merupakan kelas besarang (kelas dalam kelas). Tujuan dari inner class untuk mengelompokkan kelas-kelas yang termasuk bersama, yang membuat sebuah source code lebih mudah dibaca dan di pelihara.

Untuk mengakses inner class, buatlah object dari kelas luar, dan kemudian buat objek dari kelas dalam. Berikut adalah contohnya:

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);

}

}

Output:

15

* Private Inner Class

Tidak seperti kelas regular, kelas dalam bisa **private**  atau **protected.** Jika tidak ingin object luar mengakses kelas dalam, maka nyatakan dalam kelas sebagai **private**. Seperti contoh dibawah ini:

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 dilihat dalam gambar, maka muncul garis merah pada syntax yang salah. Dan jika tetap ingin mengakses private inner class dari kelas luar (MyMainClass), maka terdapat kesalahan (error):

MyMainClass.java:12: error: OuterClass.InnerClass has private access in OuterClass  
OuterClass.InnerClass myInner = myOuter.new InnerClass();  
              ^

* Static Inner Class

Kelas dalam juga bisa menggunakan static, yang berarti bahwa dapat mengaksesnya tanpa membuat objek dari kelas luar. Berikut adalah contoh dari source kode menggunakan static:

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);

}

}

Output :

5

Perlu diingat bahwa, sama seperti **static** atribut dan metode, **static**  kelas dalam tidak memiliki akses ke anggota kelas kelas luar.

* Access Outer Class dari Inner Class

Terdapat satu keuntungan dari adanya kelas dalam, adalah mereka dapat mengakses atribut dan metode dari kelas luar, contoh gambar dibawah ini merupakan source code yang dapat mengakses kelas luar dari kelas dalam:

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());

}

}

Output :

10

### **Java Abstraction**

Abstraksi data adalah proses menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna. Abstraksi dapat dicapai dengan kelas atau interface abstrak.

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

1. Kelas abstrak : adalah kelas terbatasyang tidak dapat digunakan untuk membuat objek (untuk mengaksesnya, itu harus diwarisi dari kelas lain).
2. Metode abstrak : hanya dapat digunakan dalam kelas abstrak, dan tidak memiliki tubuh, tubuh disediakan dari subclass(pewarisan dari).

Kelas abstrak dapat memiliki metode abstrak dan regular, seperti gambar dibawah ini :

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

}

}

Dari contoh diatas, tidak mungkin membuat objek dari kelas Animal:

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

Untuk mengakses elas abstrak, harus diwarisi dari kelas lain. Maka dari itu, konversikan kelas Animal pada bab polymorphism menjadi kelas abstrak. Kata kunci yang digunakan pada bab inheritance saat mewarisi dari kelas adalah **extends.**

// 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();

}

}

Output:

The pig says: wee wee

Zzz

### **Java Interface**

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

// interface

interface Animal {

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

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

}

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

// Interface

interface Animal {

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

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

}

// Pig "implements" the Animal interface

class Pig implements Animal {

public void animalSound() {

// The body of animalSound() is provided here

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

}

public void sleep() {

// The body of sleep() is provided here

System.out.println("Zzz");

}

}

class MyMainClass {

public static void main(String[] args) {

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

myPig.animalSound();

myPig.sleep();

}

}

Output:

The pig says: wee wee  
Zzz

**Catatan tentang inteface:**

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

#### **Mengapa Dan Kapan Menggunakan interface ?**

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

2) Java tidak mendukung "multiple inheritance" (sebuah kelas hanya dapat diwarisi dari satu superclass). Namun, ini dapat dicapai dengan interface, karena kelas dapat **mengimplementasikan** banyak interface.

* Multiple Interface

**Catatan:** Untuk mengimplementasikan beberapa interface, pisahkan dengan koma.**Contoh :**

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("Some text..");

}

public void myOtherMethod() {

System.out.println("Some other text...");

}

}

class MyMainClass {

public static void main(String[] args) {

DemoClass myObj = new DemoClass();

myObj.myMethod();

myObj.myOtherMethod();

}

}

Output :

Some text...  
Some other text

### **Java Enums**

Enum (enumeration)adalah salah satu kelas library di J2SE (versi 5), yang dapat kita gunakan untuk membuat kumpulan variable yang telah didefinisikan menjadi tipe data **konstanta** (variabel tidak bisa diubah, seperti final variabel).setiap objek baru yang didefinisikan berdasarkan tipe data Enum, objek tersebut dapat mengakses atribut atau method (Enum), lalu mengaplikasikan nilai atau value yang ada di dalamnya. Enum bisa juga disebut array, namun dalam bentuk konstanta, dan juga bertipe class.

jika sebelumnya mendefinisikan nilai pada variable bertipe data konstanta dengan baris per baris dengan menggunakan keyword **enum**, kita dapat mengklasifikan variable-variabel bertipe data konstanta tersebut didalam **block{}**.

Untuk membuat enum, menggunakan enum kata kunci (bukan kelas atau interface), dan memisahkan konstanta dengan koma. Catatan bahwa mereka harus dalam huruf besar:

enum Level {

LOW,

MEDIUM,

HIGH

}

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

Level myVar = Level.MEDIUM;

* Enum di dalam Kelas

Anda juga dapat memiliki enum di dalam kelas:

public class MyClass {

enum Level {

LOW,

MEDIUM,

HIGH

}

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

System.out.println(myVar);

}

}

Output :

MEDIUM

* Enum dalam pernyataan SWITCH

Merupakan pernyataan yang dirancang khusus untuk menangani suatu keputusan atau suatu hasil akhir dalam membuat project, switch-case memiliki kegunaan yang sama seperti if-else. Tetapi switch-case memiliki suatu batasan yaitu

1. Data yang diperiksa harus bertipe integer(INT) atau karakter (CHR).
2. Range data yang diperiksa bernilai 0 s/d 255.

Bentuk Penulisannya :

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;

}

}

}

Output :

Medium level

* Loop melalui enum

Yaitu untuk mengulang kelas enum di atas atau mengembalikan sebuah array dari semua konstanta enum. Metode values() ini berguna ketika Anda ingin loop melalui konstanta enum.

* Ada dua method yang sering digunakan pada enum atara lain
  + 1. Values ();

Method ini mengembalikan keseluruhan dari enumeration

* + 1. ValueOf(String);

Method ini mengembalikan isi dari enumeration yang sesuai dengan parameter string

bentuk penulisannya:

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

System.out.println(myVar);

}

Output:

LOW  
MEDIUM  
HIGH

* Catatan Penting

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).

### **Java User Input**

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 String.

Contoh:

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

class MyClass {

public static void main(String[] args) {

Scanner myObj = new Scanner(System.in); // Create a Scanner object

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

String userName = myObj.nextLine(); // Read user input

System.out.println("Username is: " + userName); // Output user input

}

}

Output:

Enter username

* Jenis Input

Pada contoh di atas, kami menggunakan metode nextLine() , yang digunakan ntuk 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);

}

}

Output :

Enter name, age and salary:

### **Java Date**

***Java Date And Time***

* Tanggal Java

Java tidak memiliki kelas tanggal bawaan , tetapi dapat mengimpor paket ***java.time*** agar berfungsi dengan API tanggal dan waktu. Paket terdiri atas beberapa kelas tanggal dan waktu .

Contoh :

|  |  |
| --- | --- |
| Class | Deskripsi |
| Local Date | Menjelaskan tentang tanggal (tahun,bulan,hari  (yyyy-MM-dd)) |
| Local Time | Menjelaskan tentang waktu (jam,menit,detik dan seperinu detik (HH-mm-ss-zzz)) |
| Local Date Time | Menjelaskan tentang tanggal dan waktu (yyyy-MM-dd-HH-mm-ss.zzz) |
| Date Time Formatter | Menampilkan dan menguraikan objek tanggal dan waktu |

* Menampilkan 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

}

}

Output :

2020-01-02

* Menampilkan 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);

}

}

Output :

21:15:36.661289

* Menampilkan 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);

}

}

Output :

2020-01-02T21:15:36.694152.

* Tanggal dan Waktu

‘T’ pada contoh diatas digunakan untuk memisahkan tanggal dan waktu . bisa juga menggunakan kelas DateTimeFormatter dengan metode ofPattern() dalam paket yang sama untuk memformat atau menguraikan objek 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);

}

}

Output:

Before Formatting: 2020-01-02T21:15:36.695663  
After Formatting: 02-01-2020 21:15:36

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

1. yyyy-MM-dd

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);

}

}

Output :

Before Formatting: 2020-01-02T21:33:37.141384  
After Formatting: 02-01-2020 21:33:37

1. dd/MM/yyyy

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);

}

}

Output :

Before Formatting: 2020-01-02T21:37:55.764861  
After Formatting: 02/01/2020 21:37:55

1. dd-MMM-yyyy

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);

}

}

Output :

Before Formatting: 2020-01-02T21:38:25.998142  
After Formatting: 02-Jan-2020 21:38:25

1. E, MMM dd yyyy

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);

}

}

Output :

Before Formatting: 2020-01-02T21:38:56.112575  
After Formatting: Thu, Jan 02 2020 21:38:56

### **Java Array List**

Kelas ArrayList adalah [array](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.w3schools.com/java/java_arrays.asp&xid=25657,15700021,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhiTH6rPlA-QuvlB_z7aByHZNOCRkQ) 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);

}

}

Output:

[Volvo, BMW, Ford, Mazda]

* Akses suatu Item

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

Contoh :

cars.get(0);

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));

}

}

Output :

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 :

cars.set(0, "Opel");

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);

}

}

Output :

[Opel, BMW, Ford, Mazda]

* Hapus Sebuah Item

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

Contoh :

cars.remove(0);

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);

}

}

Output :

[BMW, Ford, Mazda]

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

cars.clear();

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);

}

}

Output :

[]

* 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());

}

}

Output :

4

* Loop Through ArrayList

Ulangi elemen ArrayList dengan for loop, 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));

}

}

}

Output :

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);

}

}

}

Output:

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&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.w3schools.com/java/java_wrapper_classes.asp&xid=25657,15700021,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhgVuCoieuJyi6KxJTUJaORwg-c7ZQ) 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);

}

}

}

Output:

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:

Sortir ArrayList of Strings:

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);

}

}

}

Output :

BMW  
Ford  
Mazda  
Volvo

Contoh:

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);

}

}

}

Output:

8  
12  
15  
20  
33  
34

### **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>();

* Add Item

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

// Import the HashMap class

import java.util.HashMap;

public class MyClass {

public static void main(String[] args) {

// Create a HashMap object called capitalCities

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

// Add keys and values (Country, City)

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

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

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

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

System.out.println(capitalCities);

}

}

Output:

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

* Access an Item

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

Contoh:

capitalCities.get("England");

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"));

}

}

Output :

London

* Remove an Item

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

Contoh :

capitalCities.remove("England");

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);

}

}

Output :

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

Untuk menghapus semua item, gunakan metode clear() :

Contoh:

capitalCities.clear();

* Ukuran HashMap

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

capitalCities.size();

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());

}

}

Output :

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 :**

// Print keys

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

System.out.println(i);

}

Contoh:

// Print values

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

System.out.println(i);

}

Contoh:

// Print keys and values

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

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

}

* 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 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("key: " + i + " value: " + people.get(i));

}

}

}

Output:

Name: Angie Age: 33  
Name: Steve Age: 30  
Name: John Age: 32

### **Java Wrapper Classes**

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

Dan berikut ini merupakan table yang menjelaskan macam-macam tipe data primitif dan referensi pada Java:

|  |  |
| --- | --- |
| **Tipe Data Primitif** | **Tipe Data Referensi** |
| Char | Character |
| Byte | Byte |
| Short | Short |
| Int | Integer |
| Long | Long |
| boolean | Boolean |
| Float | Float |
| Double | Double |

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):

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);

}

}

Output:

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:

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);

}

}

Output :

5  
5.99  
A

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

Sebagai contoh, metode berikut ini 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:

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());

}

}

Output:

5  
5.99  
A

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

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

public class MyClass {

public static void main(String[] args) {

Integer myInt = 100;

String myString = myInt.toString();

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

}

}

Output:

3

### **Java Exceptions**

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.");

}

}

}

Output:

Something went wrong.

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.");

}

}

}

Output :

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

* Kata kunci lemparan

Pernyataan **throw** memungkinkan saya 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 abwah 18 (cetak “Akses ditolak”). Jika usian 18 tahun atau lebih cetak “Akses diberika”

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); // Set age to 15 (which is below 18...)

}

}

Output :

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, saya  tidak akan mendapatkan pengecualian:

Contoh :

checkAge(20);

Output:

Access granted - You are old enough!

### **BAB III**

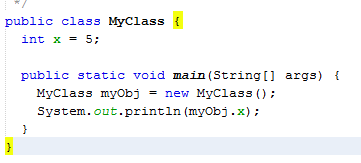
### **HASIL PERCOBAAN**

### **Classes/ Objects**

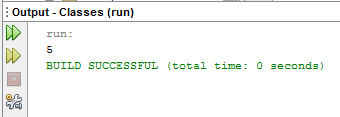
* Buat kelas



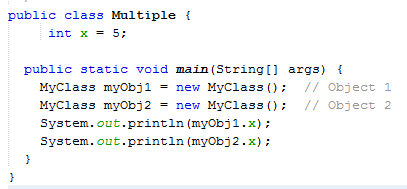
* Buat Objek



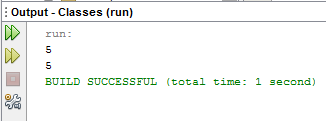
Hasil



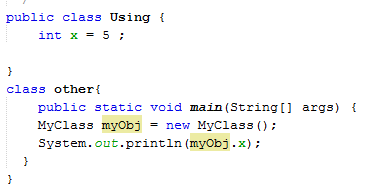
* Menggunakan Banyak Objek



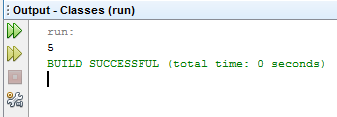
Hasil



* Menggunakan Banyak Kelas

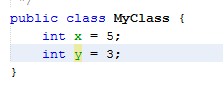


Hasil

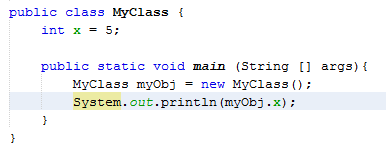


### **Attributes**

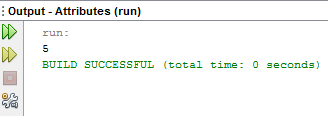
* Attribute Kelas java



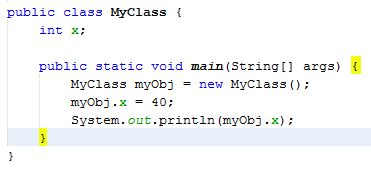
* Mengakses Attribute



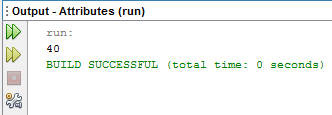
Hasil



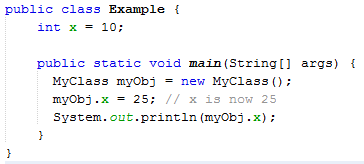
* Ubah Attribute
* Mengubah nilai attribute



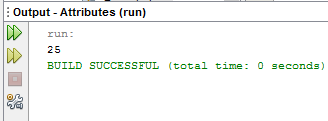
Hasil



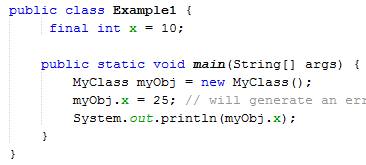
* Menimpa nilai



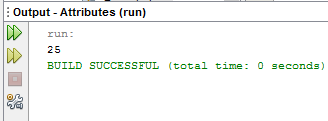
Hasil



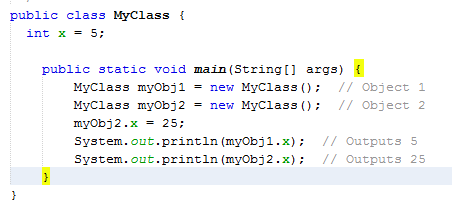
* Menimpa nilai yang ada dengan menyatakan atribut sebagai final



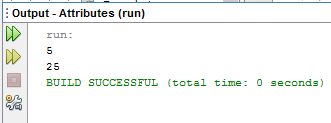
Hasil



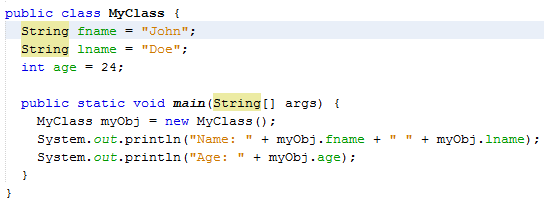
* Banyak Objek



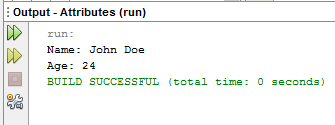
Hasil



* Banyak Attribut

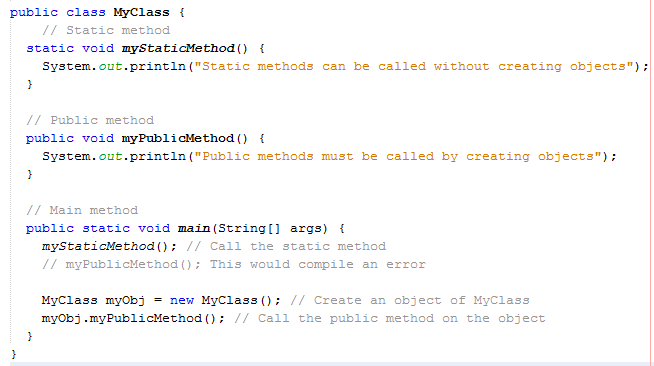


Hasil

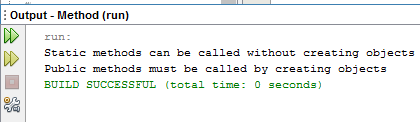


### **Method**

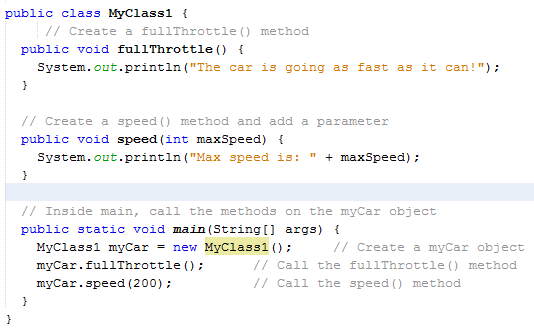
* Perbedaan static dan public



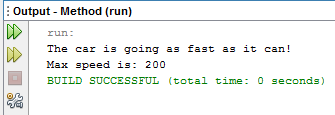
Hasil



* Metode Akses dengan Objek

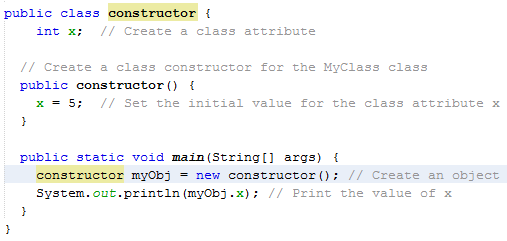


Hasil

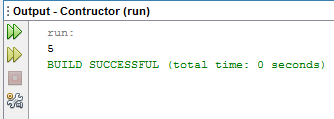


### **Construktors**

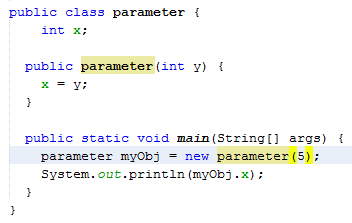
* Constructor



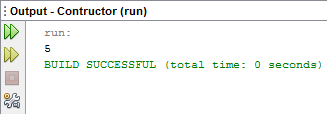
Hasil



* Constructor Parameter



Hasil

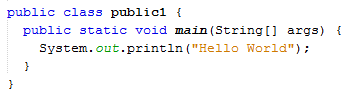


### **Modifiers**

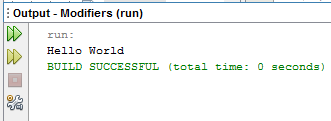
* Access Modifiers

Untuk kelas, dapat menggunakan public dan default

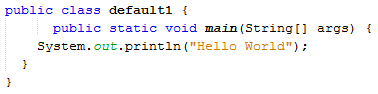
* Public



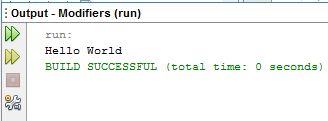
Hasil



* Default

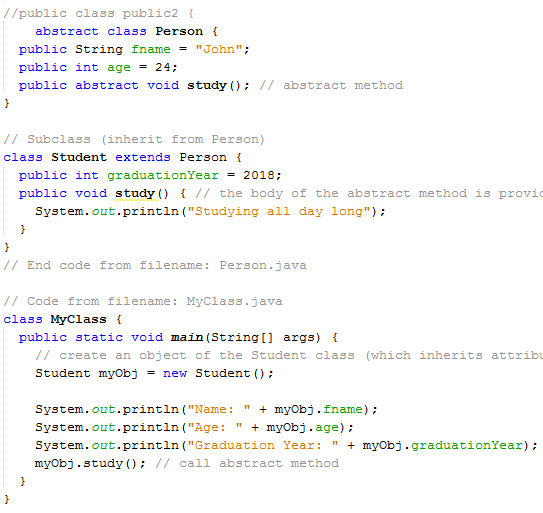


Hasil

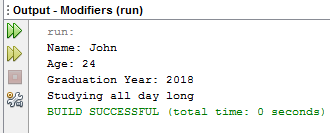


Untuk attributte, method, dan kontructor

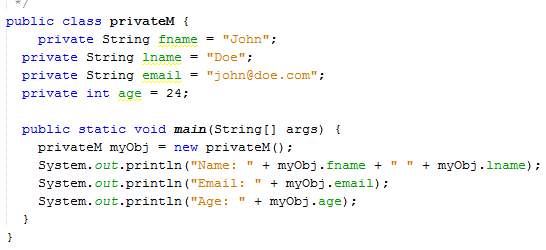
* Public



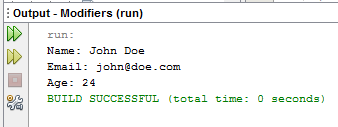
Hasil



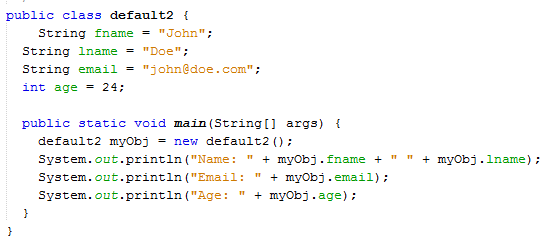
* Private



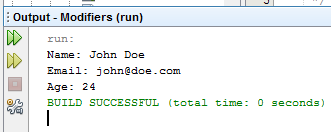
Hasil



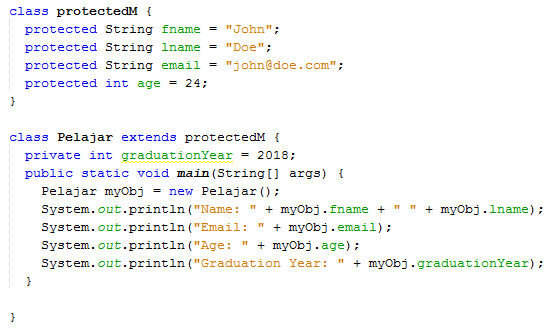
* Default



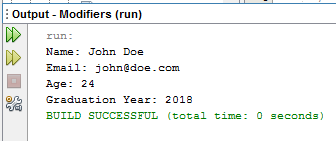
Hasil



* Protected

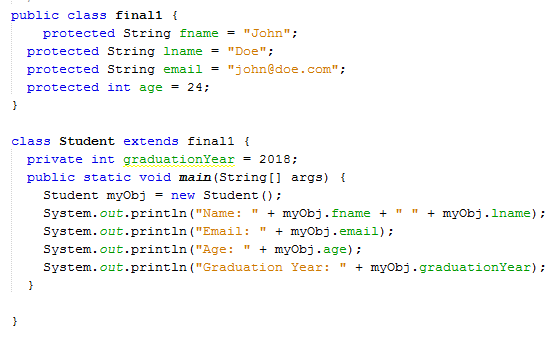


Hasil

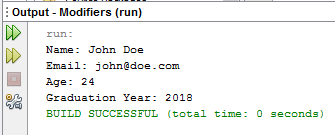


* Non-Access Modifiers

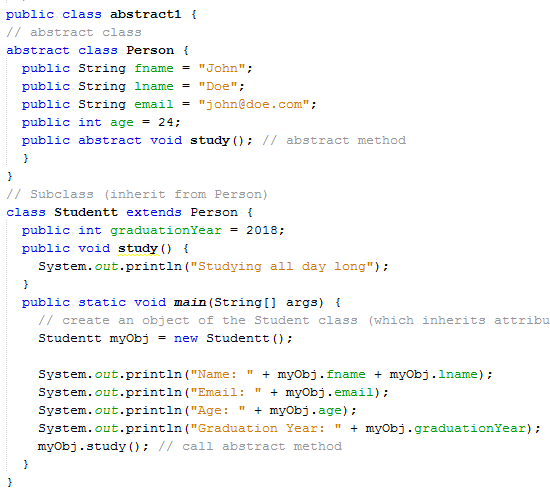
Untuk kelas, dapat menggunakan final dan abstract

* Final 

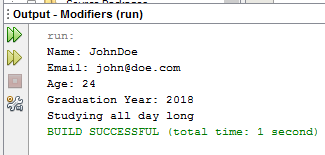
Hasil



* Abstract

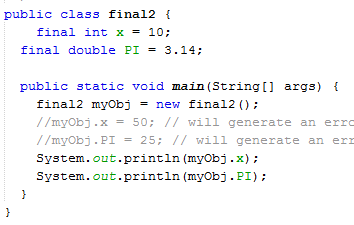


Hasil

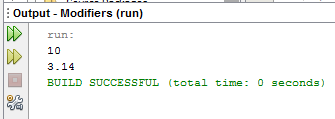


Untuk attribute dan method

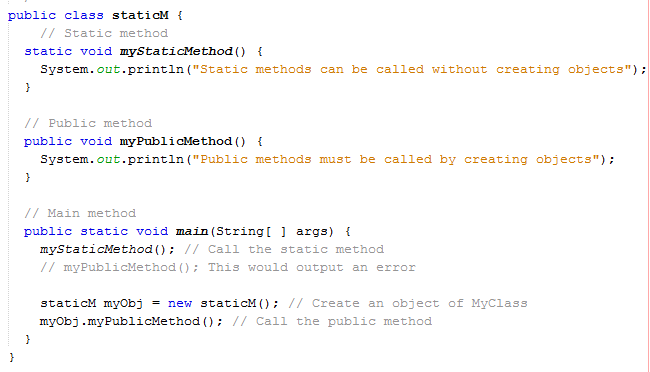
* Final



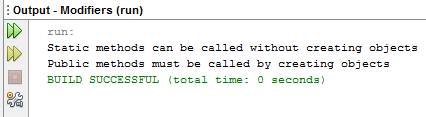
Hasil



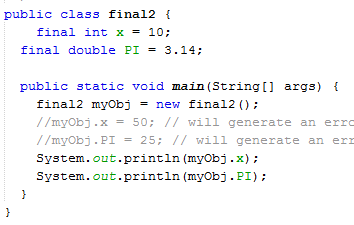
* Static



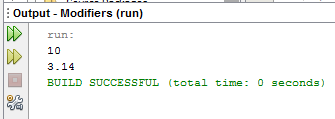
Hasil



* Final

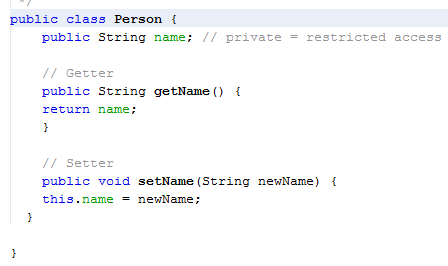


Hasil

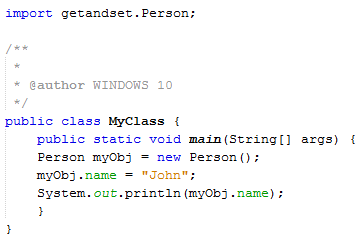


### **Encapsulation**

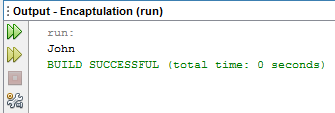
* Get and Set



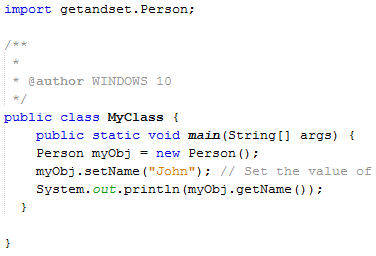
* Contoh 1



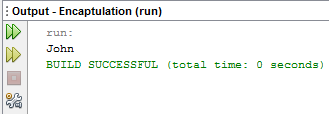
Hasil



* Contoh 2

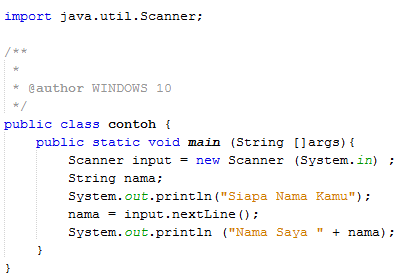


Hasil

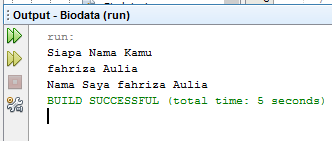


### **Package/ API**

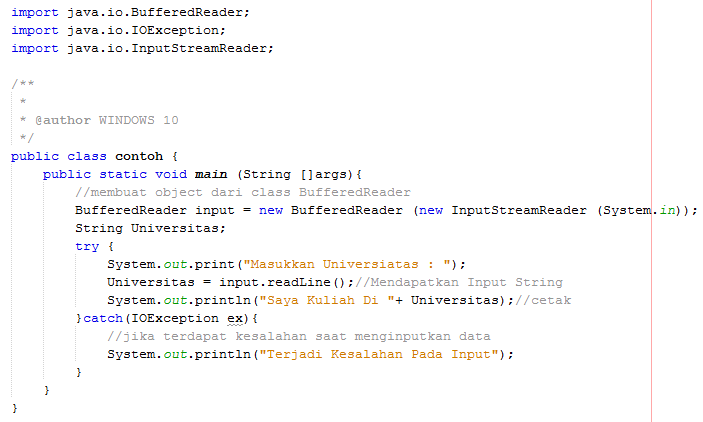
* Scanner



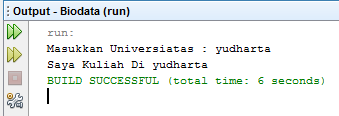
Hasil



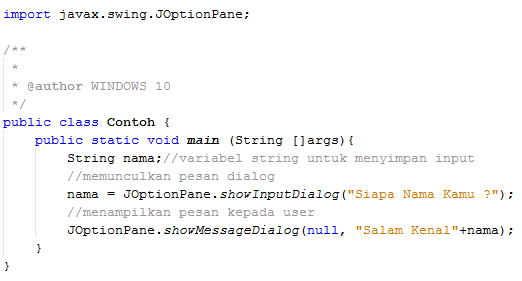
* Buffer Reader



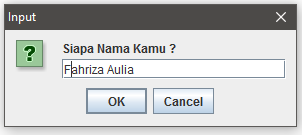
Hasil

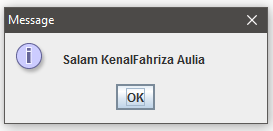


* JOption Pane



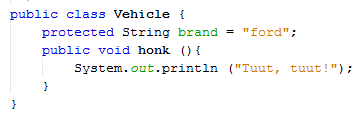
Hasil

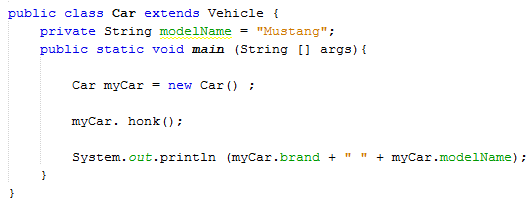




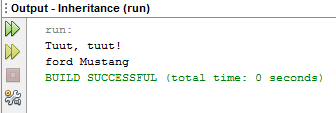
### **Inherintence**

* Percobaan 1



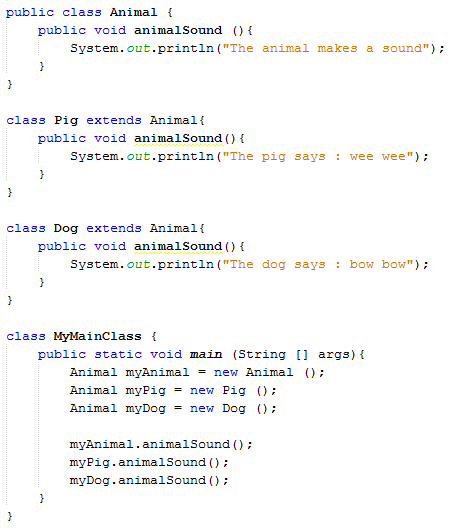


Hasil

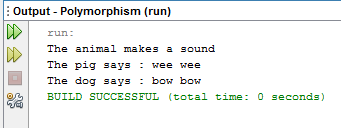


### **Pholimorphism**

* Percobaan 1

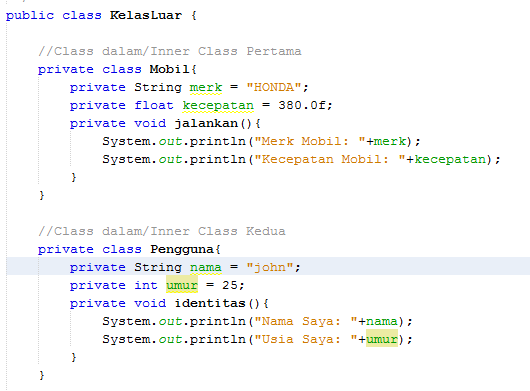


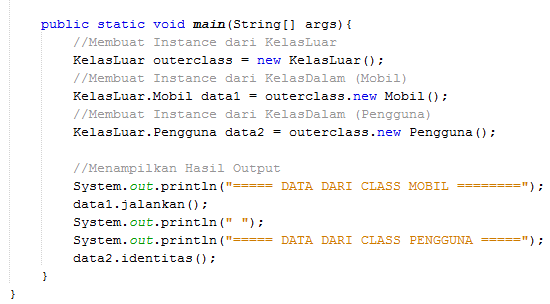
Hasil



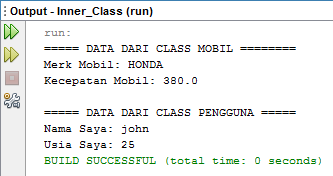
### **Inner Class**

* Non Static

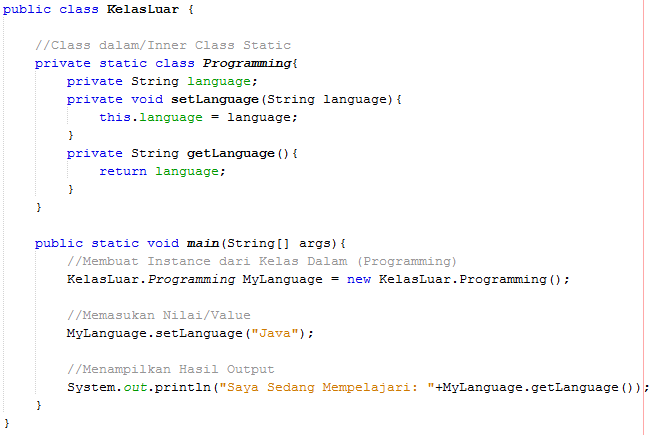




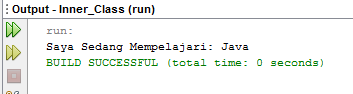
Hasil



* Static

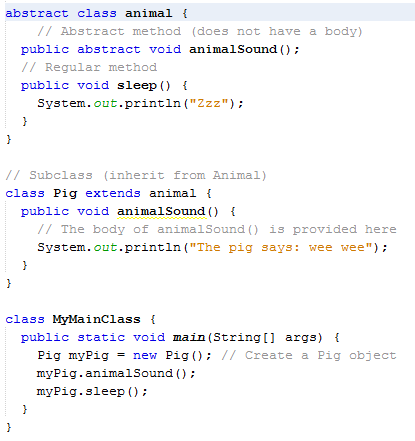


Hasil

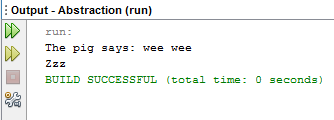


### **Abstraction**

* Percobaan

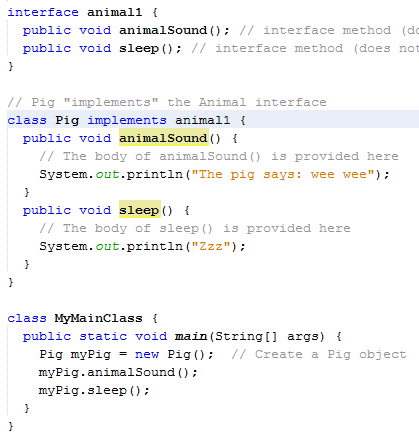


Hasil

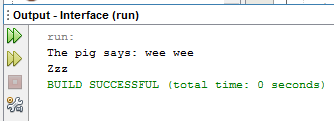


### **Interface**

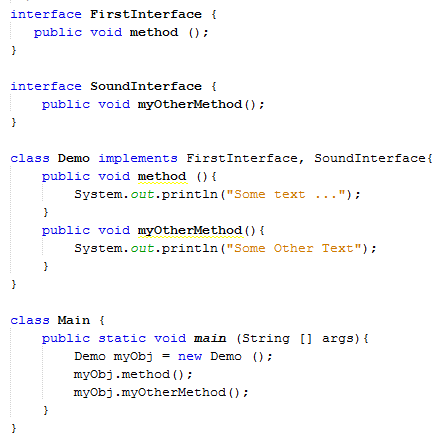
* Abstract



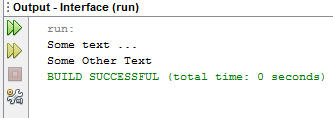
Hasil



* Multiple Interface

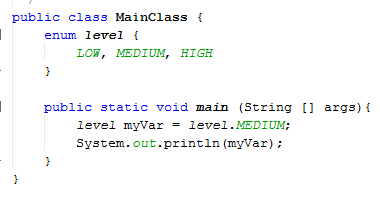


Hasil

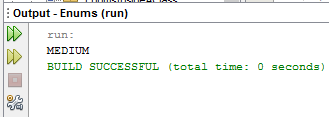


### **Enums**

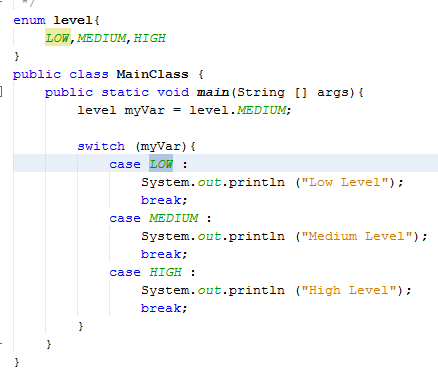
* Inside a class



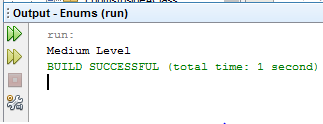
Hasil



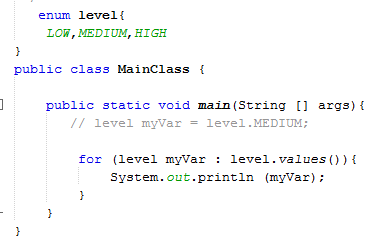
* In a Switch Statement



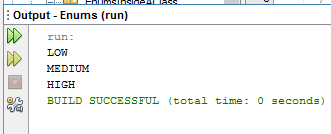
Hasil



* Loop Through

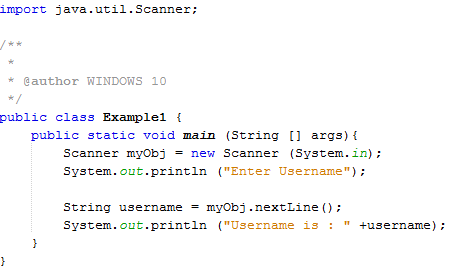


Hasil

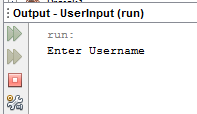


### **Use Input**

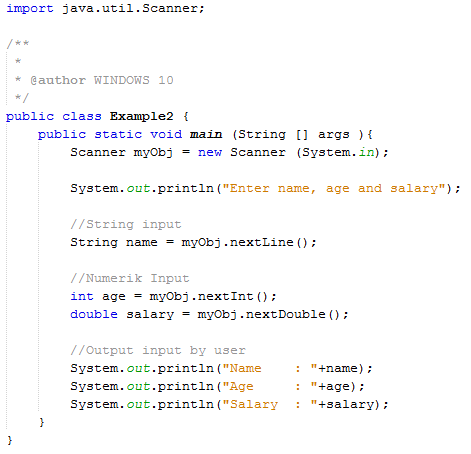
* Percobaan 1



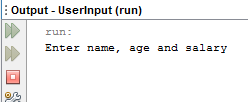
Hasil



* Percobaan 2



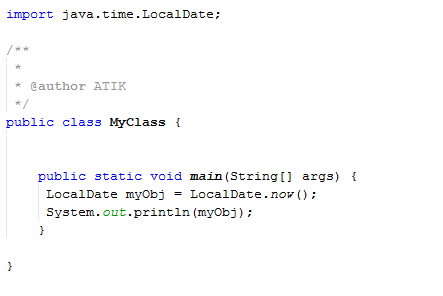
Hasil

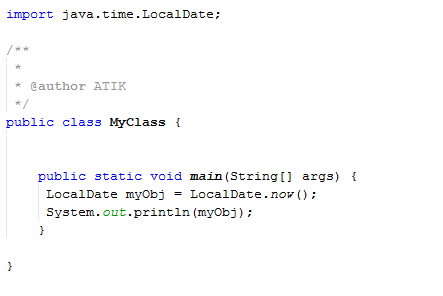


### **Java Date**

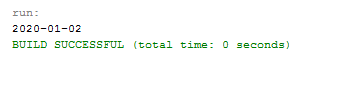
* Menampilkan Tanggal Saat ini

Contoh :



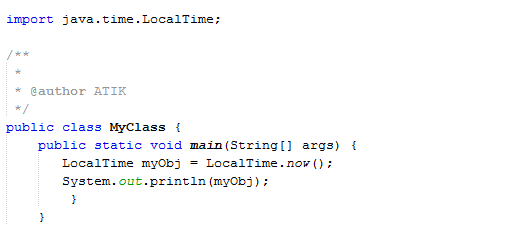


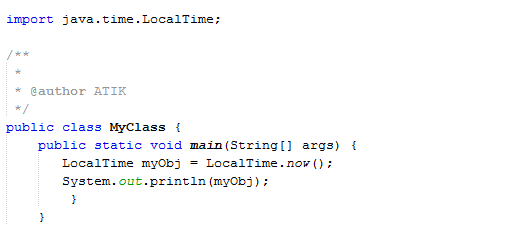
Hasil:

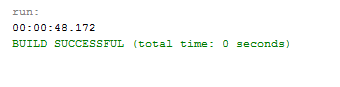


* Menampilkan Waktu Saat Ini

Contoh :

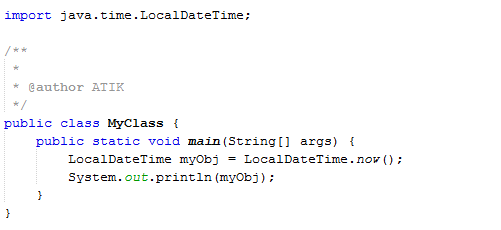


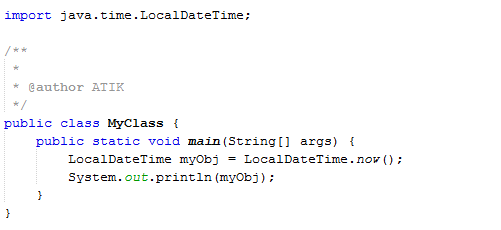
 Hasi :

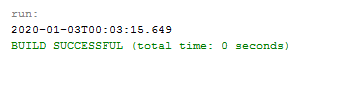


* Menampilkan Waktu dan Tanggal Saat Ini

Contoh :

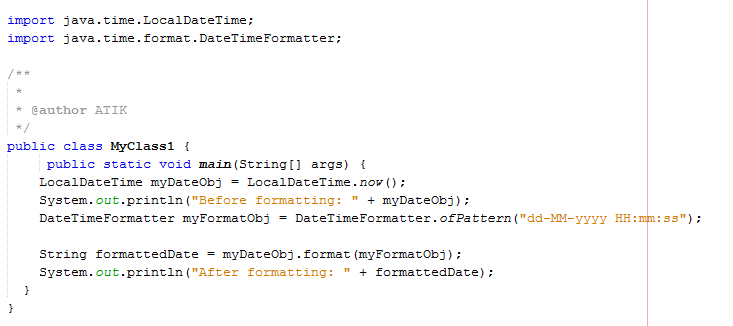


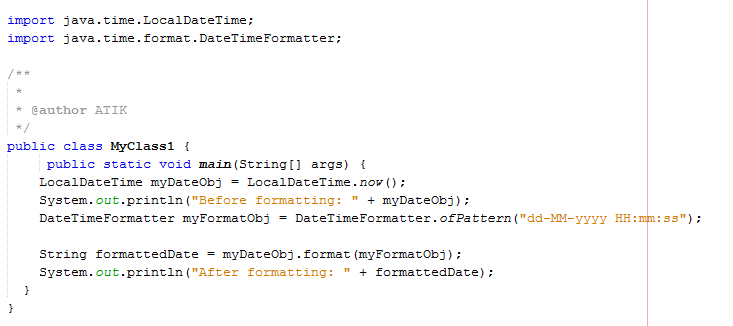
 Hasil:

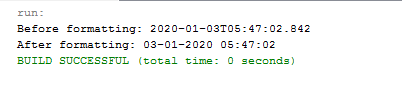


* Memformat Tanggal dan Waktu
  1. Value: yyy – MM – dd

Contoh :

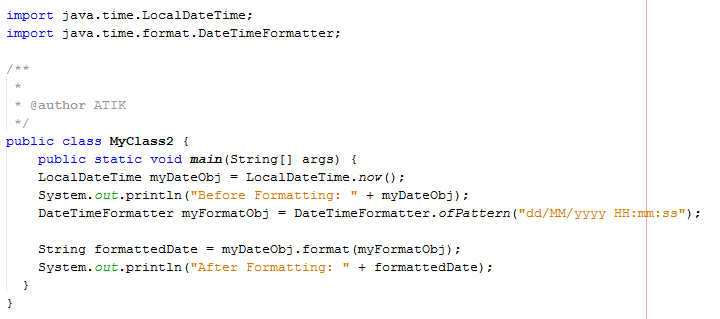


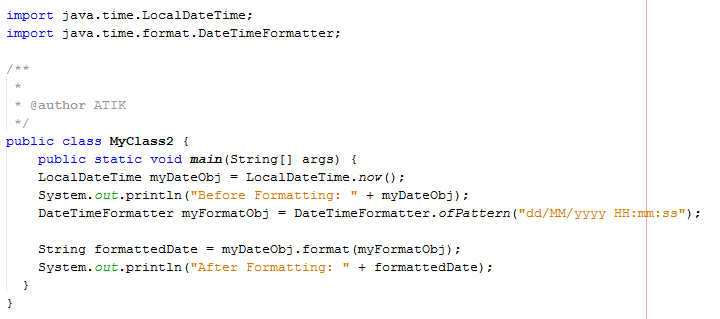
 Hasil:

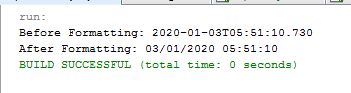


* 1. Value: dd/MM/yyy

Contoh:

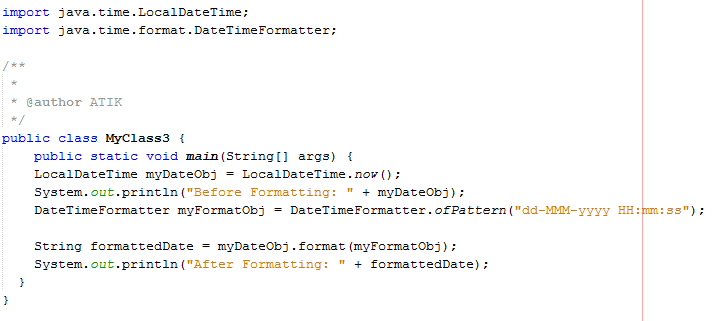


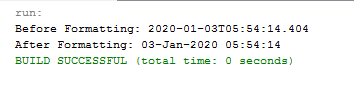
 Hasil:



* 1. Value: dd – MMM – yyyy

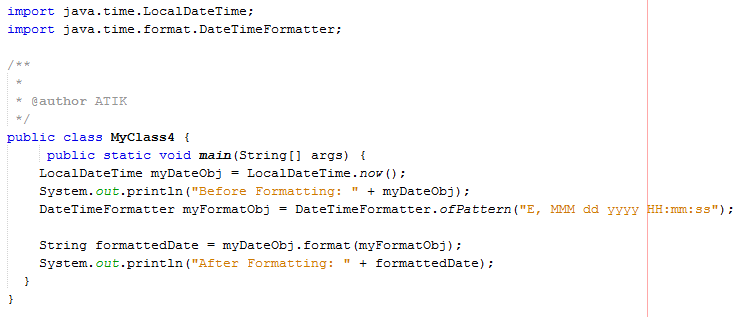
Contoh:

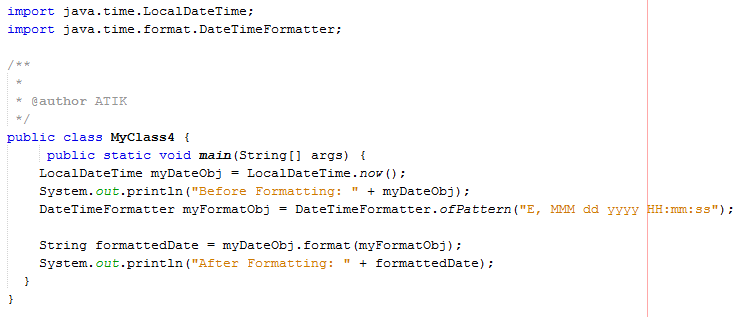
 Hasil :

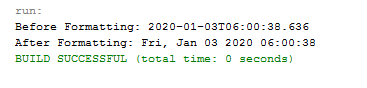


1. Value: E, MMM dd yyyy

Contoh:



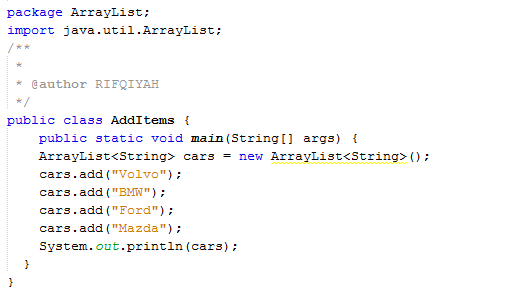
 Hasil:

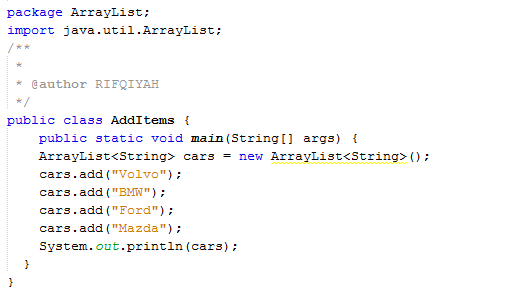


### **Java Array List**

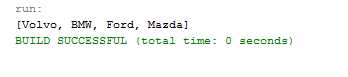
* Menambahkan Item

Contoh:



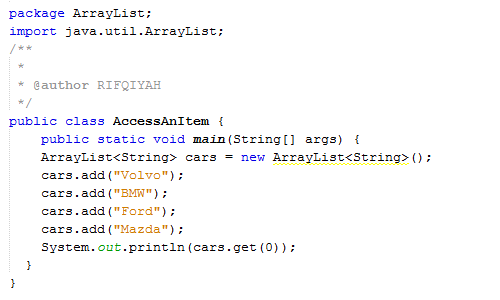


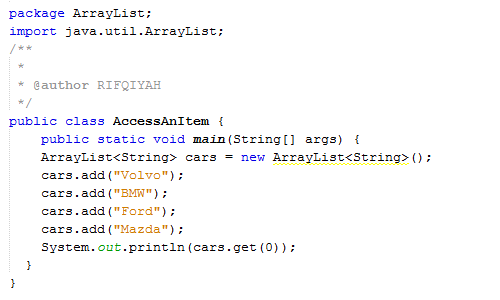
Hasil :



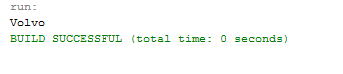
* Mengakses Item

Contoh :



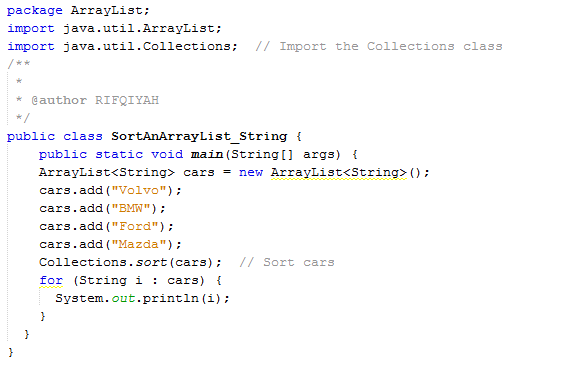


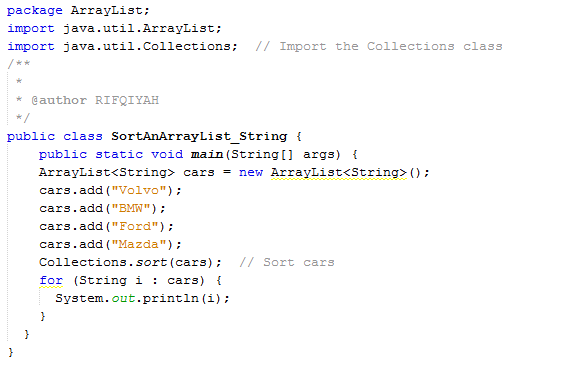
Hasil:

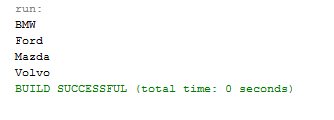


* Mensortir Daftar Array

Contoh :



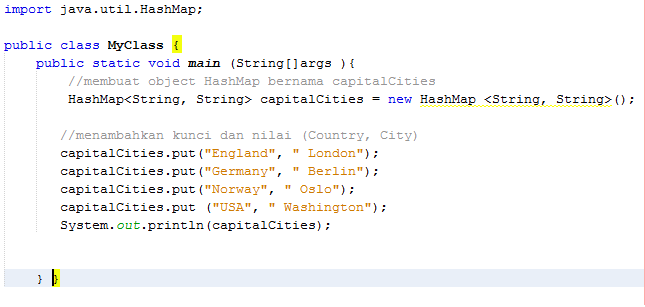
 Hasil :



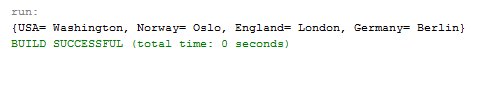
### **Java HashMap**

* Menambahkan Item

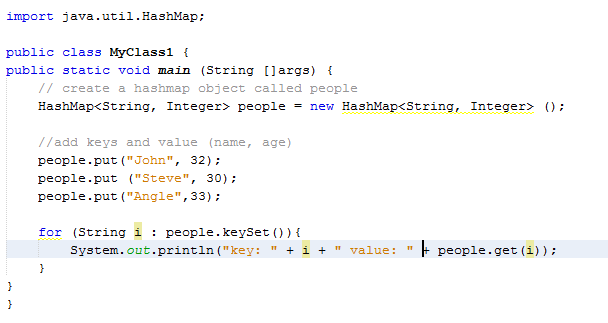
Contoh : untuk menambahkan item dengan kata kunci string dan nilai string

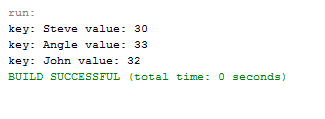


Hasil:



Contoh : untuk menambahkan item dengan kunci String, dan nilai Integer

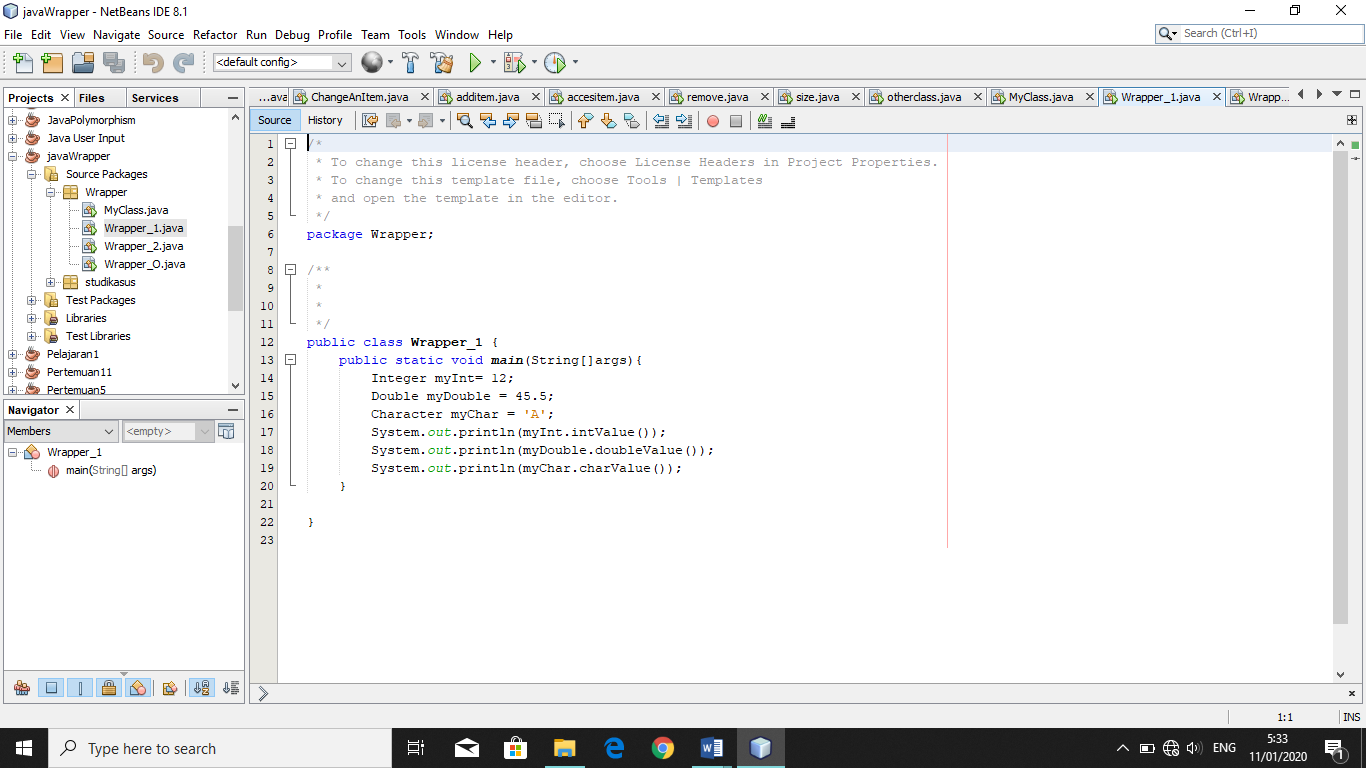
 Hasil :

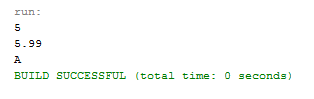


### **Java Wrapper**

* Membuat Objek

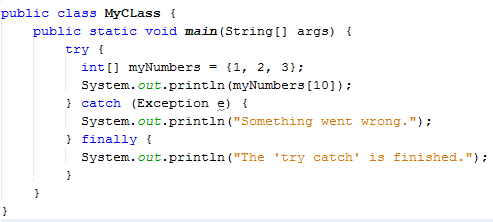
Contoh :

 Hasil :

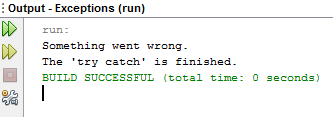


### **Java Exceptions**

Contoh : kata kunci try and catch



Hasil :

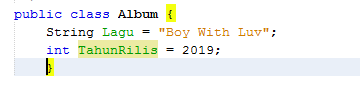


### **BAB IV**

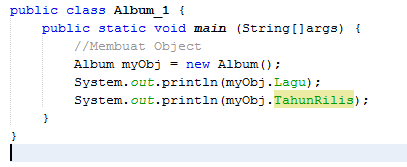
### **STUDI KASUS dan ANALISIS**

### **Java Class and Objects**

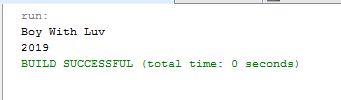
Untuk membuat sebuah studi kasus tentang java class and object, maka buatlah 2 class, bernama Album, dan Album\_1. Pada class Album dibuat sebuah variabel Lagu dan TahunRilis beserta tipe variabelnya. Pembuatan sebuah variabel digunakan sebagai penampung nilai yang telah dibuat.



Dalam class Album\_1 telah dibuat sebuah object bernama Album dan memberikan 2 object di dalamnya, karena pada class Album terdapat 2 variabel. Jika dituliskan kodenya seperti berikut:



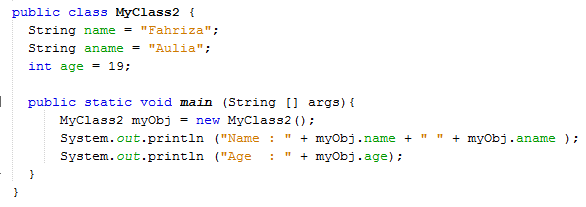
Untuk menjalankan sebuah program, hanya dapat menjalankan class yang memiliki contains main class.

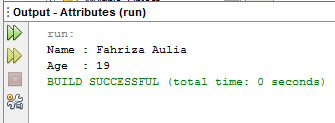


Di dalam tampilan output diatas, telah menampilkan object yang dipanggil.

### **Java Class Attributes**

Pada studi kasus java class attribute, telah dibuat class bernama MyClass2 yang didalamnya terdapat 3 variabel masing-masing bertipe String dan Integer. String untuk data berupa teks, dan integer untuk data berupa bilangan bulat. Di dalam class MyClass2, dibuat Objek bernama mahasiswa, dan source code untuk memanggil atribut.

Hasil:

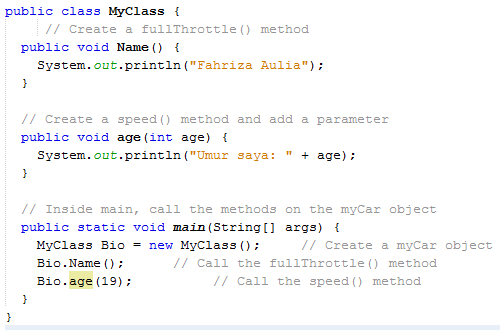


### **Java Class Methods**

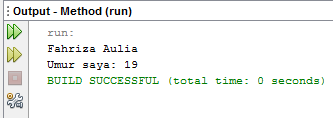
Didalam studi kasus java class method, telah digunakan method berupa getter and setter untuk enkapsulasi. Method setter dan getter harus diberikan modifier public, karena method ini akan diakses dari luar class.

Perbedaan method setter dan getter terletak pada nilai kembalian, parameter, dan isi dari methodnya. Method setter tidak memiliki nilai kembalian void (kosong), karena hanya bertugas untuk mengisi data ke atribut. Sedangkan method getter memiliki niali kembalian sesuai dengan tipe data yang akan diambil.

Berikut adalah source code lengkapnya untuk studi kasus java class method

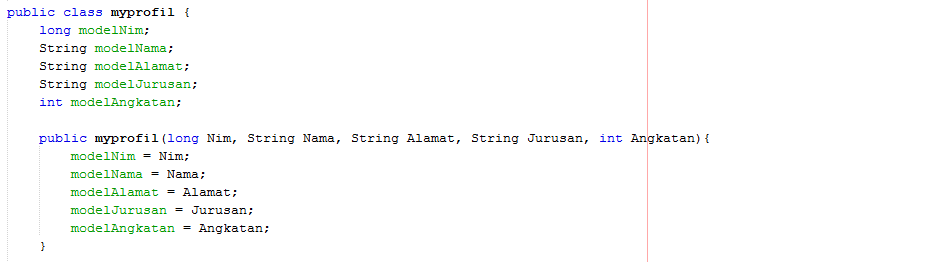


Hasil:

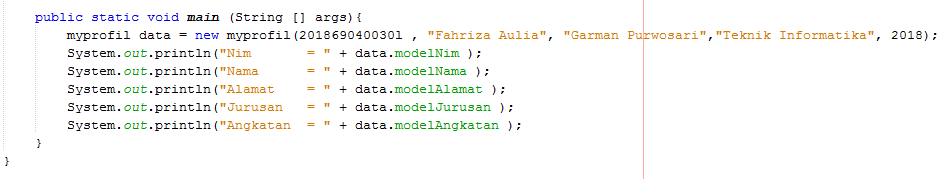
****

### **Java Constructors**

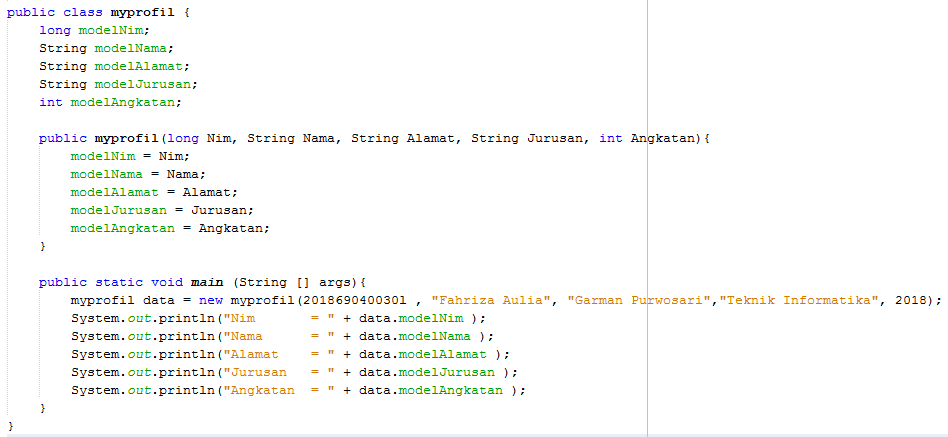
Pada studi kasus untuk java constructor digunakan constructor dengan parameter yang biasanya digunakan untuk menyiapkan data untuk class. Untuk melakukan ini, telah dibuat parameter sebagai inputan untuk constructor.



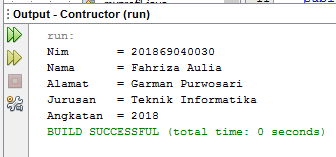
Pada kode class User diatas, telah ditambahkan parameter **modelNim, modelNama, modelAlamat, modelJurusan, modelAngkatan** ke dalam constructor. Berarti saat membuat object, harus menambahkan nilai parameter seperti berikut:



Code pada program java secara lengkap:



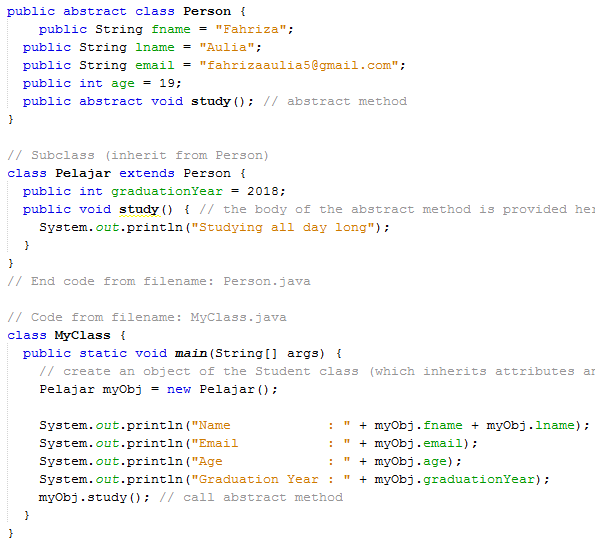
Hasil :



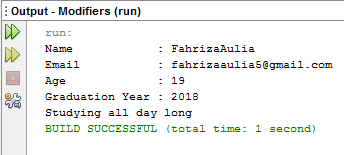
### **Java Modifiers**

Pada study kaus modifiers ini saya membuat dari public yang terdapat pada acces modifiers. Disini saya membuat class yang bernama **Person** berisikan data yang mau ditampilakan dan disini aya juga membuat clas yang bernama **Pelajar** clas ini di gabungkan dengan class **Person** dan membuat class **MyClass** untuk memangil data yang sudah di inputkan si class person dan pelajar.

Contoh :

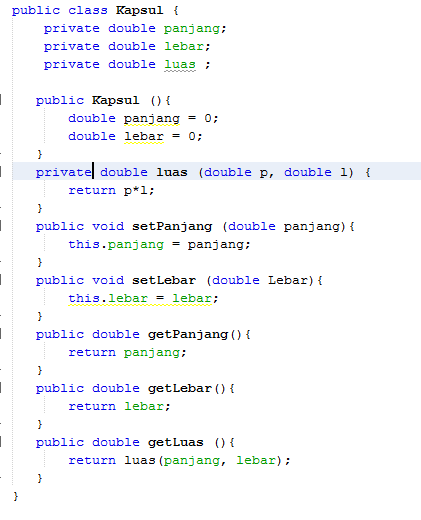


Hasil :

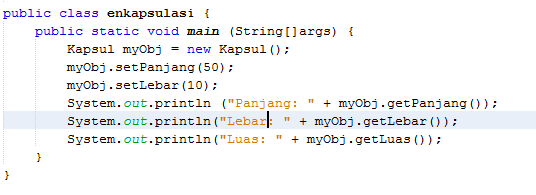


### **Java Encapsulation**

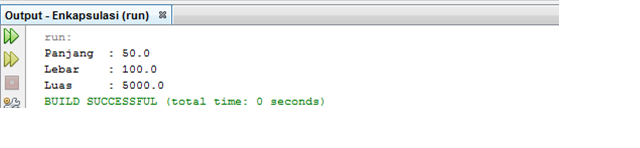
Untuk studi kasus pada java encapsulation,digunakan kode untuk menghitung luas persegi dengan bilangan yang sudah ditetapkan. Telah dibuat dua class dengan nama Kapsul dan enkapsulasi. Pada class kapsul diisikan dengan variabel dan method get and set agar dapat dipanggil dalam class enkapsulasi.



Pada class enkapsulasi berisi main method serta atribut untuk memanggil variabel pada class kapsul, dan telah dibuat object baru dengan nama myObj.



Hasil :



### **Java Package/API**

* Class Scanner

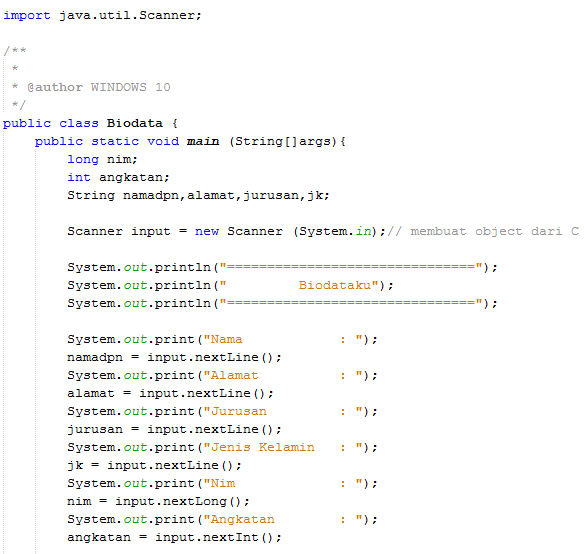
Pada studi kasus java package/API class scanner, akan dibuat contoh menginputkan biodata mahasiswa. Untuk membuat class scanner diperlukan mengimport dulu.

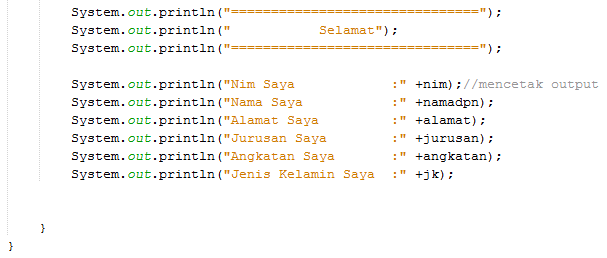


Pada penggunaan scanner untuk menerima input, perlu di deklarasikan sebuah variabel sebagai untuk mewakili scanner pada system.in, perhatikan gambar berikut ini. Jadi, pada saat membuat sebuah fungsi input, hanya perlu memanggil variabel “input” untuk mewakili fungsi system.in



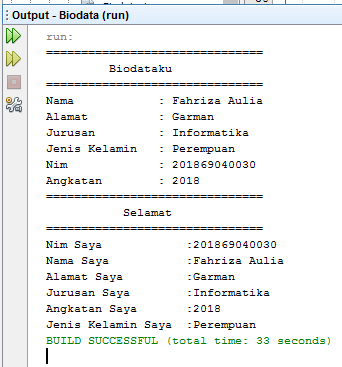
Berikut ini adalah source code lengkapnya:





Perlu diketahui tentang code yang diberi garis merah dibwahnya. Untuk class scanner di haruskan untuk mengimport scanner terlebih dahulu, dan juga membuat object scanner yang berguna untuk membuat object. Untuk menginputkan data pada class scanner terbagi menjadi beberapa tipe data, yaitu untuk tipe data String maka menggunakan **input.next();** , jika tipe data Long (untuk angka berkapasitas besar) maka menggunakan **input.nextLong();** , dan untuk tipe data integer maka menggunakan **input.nextInt();**

Hasil :



* Buffer Reader

**BufferedReader** adalah class yang berada pada oaket java.io , io adalah singkatan dari input output. Class BufferedReader juga mampu menerima input dari file dari sebuah jaringan. Sehingga untuk penggunaan BufferedReader di java harus mengimportkan 3 class pada java.io sekaligus yaitu:

**import java.io.BufferedReader;**

**import java.io.IOException;**

**import java.io.inputStreamReader;**

atau untuk lebih mempersingkat pengkodean dapat menggunakan langsung dari clas io, karena dapat memanggil 3 sekaligus class pada java paket io. Untuk membuat java package/API dalam buffer reader diperlukan untuk mengimport terlebih dahulu pada package.

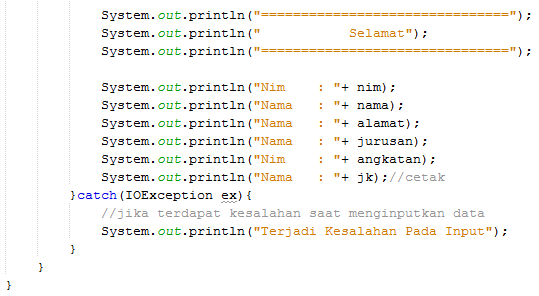


Dan membuat object baru pada class agar dapat memuat object yang nantinya akan di inputkkan.



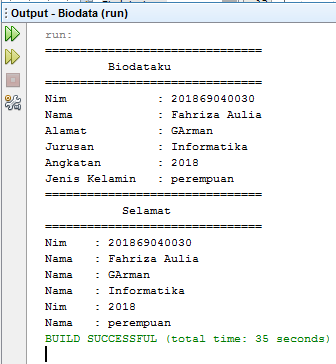
Untuk lebih jelasnya mengetahui perbedaan java package/API pada class scanner, buffer reader, console, dan JOption Pane. Perhatikan source code yan digaris bawahi.





Terdapat beberapa perbedaan dengan class scanner yaitu mengimport di package, membuat object baru, dan untuk menginputkan datanya. Untuk membaca input data tipe string maka menggunakan readLine(), dan untuk tipe data integer menggunakan read().

Hasil:



* JOptionPane

JOptionPane merupakan sebuah class yang menyediakan jendela dialog. Sebelum menggunakan JOptionPane, harus mengimpor terlebih dahulu ke dalam program, yaitu:

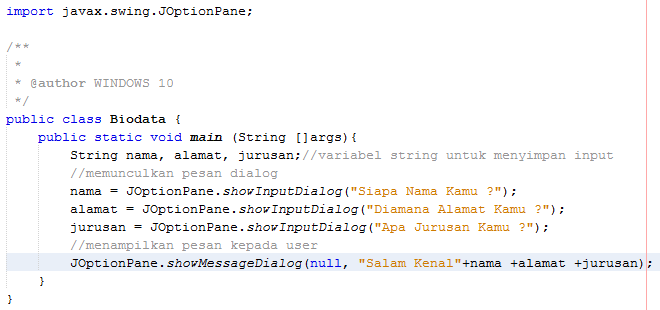


Class JOptionPane dapat digunakan lansgung tanpa harus membuat object, seperti gambar dibawah ini:

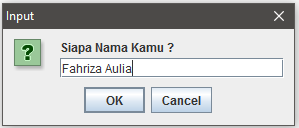


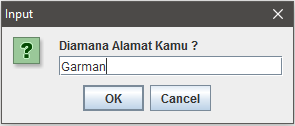
Null adalah parameteruntuk komponen induknya, bisa diberikan null atau this.

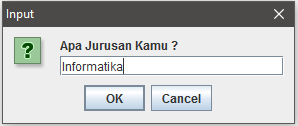
Untuk lebih jelasnya, dapat dilihat pada studi kasus program java JOption Pane.

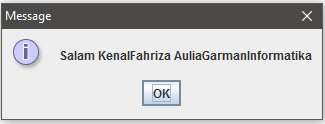


Hasil:





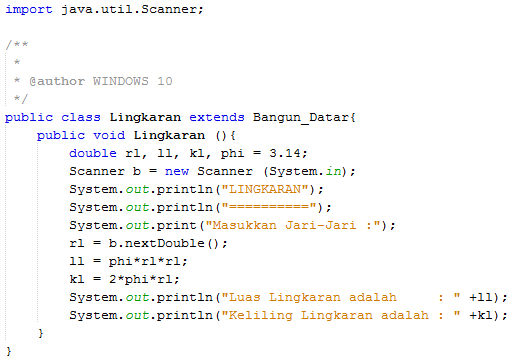




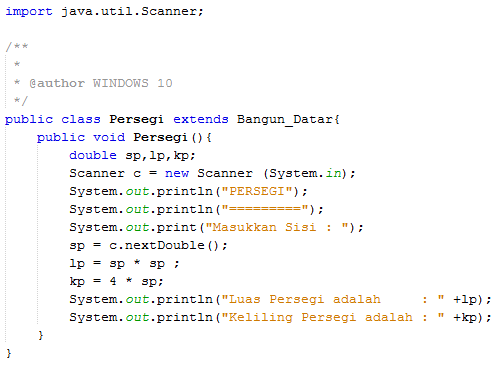
### **Java Inherintence**

Program java yang dibuat untuk studi kasus java inheritance ini adalah perhitungan luas bangun datar persegi,segitiga, lingkaran, dan persegi panjang dengan menginputkan data sendiri, maka dari itu harus mengimport class scanner. Buat sebuah project dalam satu package bernama **inheritance** dan class yang terdiri dari 5 class, masing-masing bernama segitiga, lingkaran, persegi, persegi panjang, dan main.

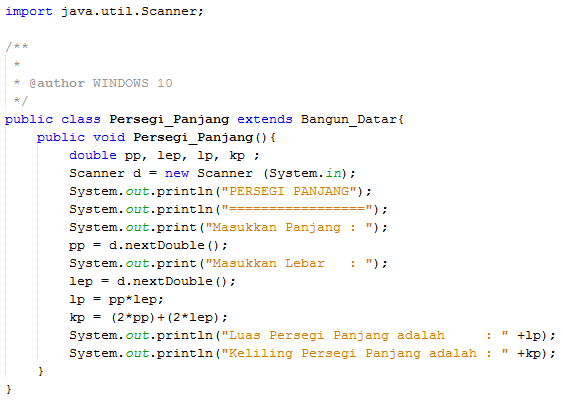
* Lingkaran

****

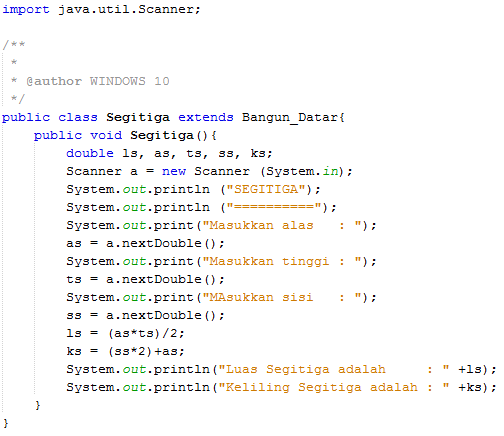
* Persegi



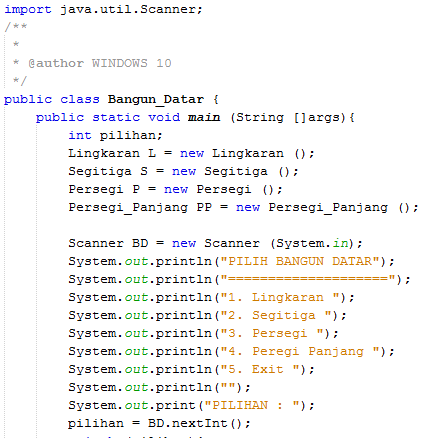
* Persegi Panjang

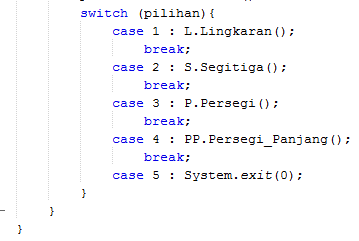


* Segitiga



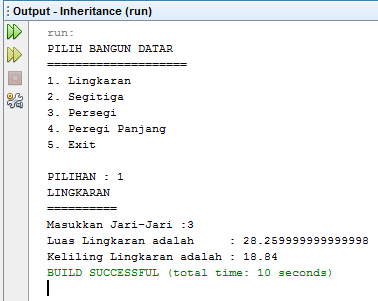
* Bangun Datar/Main



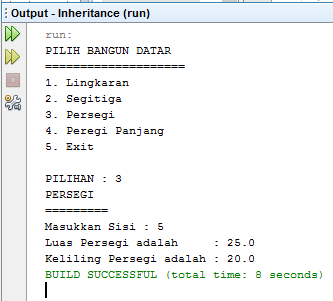


Hasil :

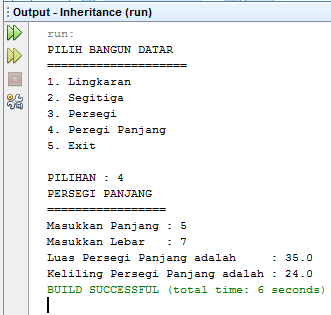
* Lingkaran



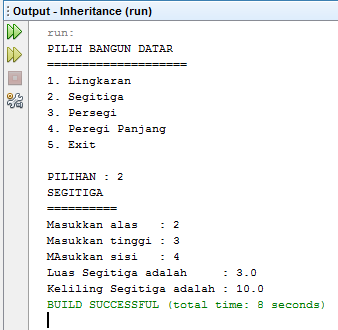
* Persegi



* Persegi Panjang

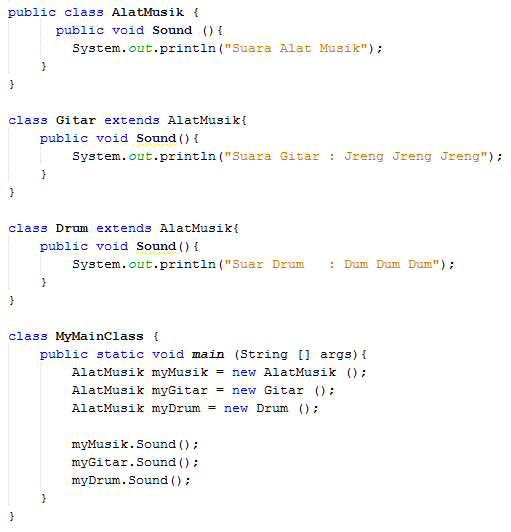


* Segitiga

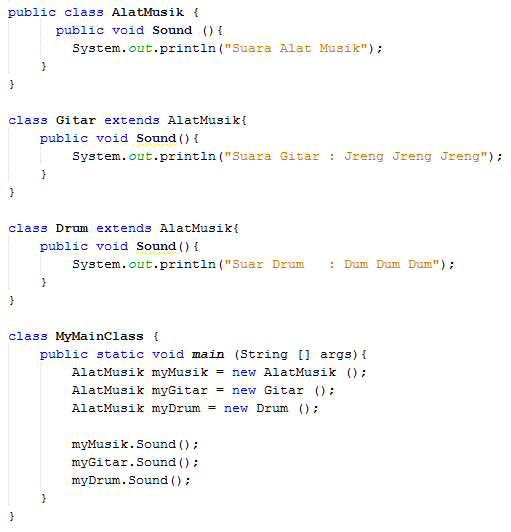


### **Java Polymorphism**

Polymorphism artinya memiliki banyak bentuk.

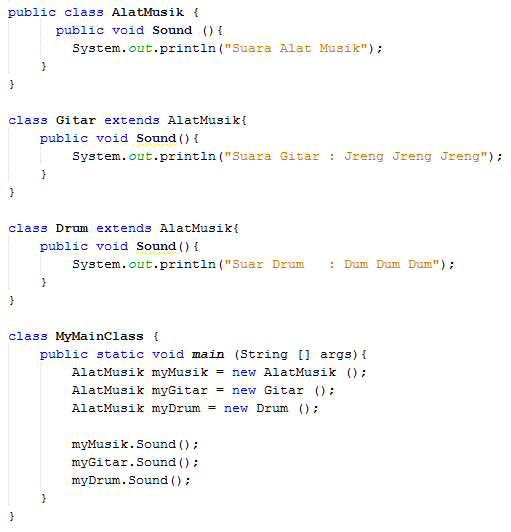


Pada kode diatas, objek hewan hanya dapat melakukan satu jenis aktifitas, yaitu suara. Namun, perl diingat aktifitas itu bisa dilakukak oleh objek sub class dari class hewan atau class apa saja yang diatasnya terdapat hewan yang terkait dalam hirarki inheritance.

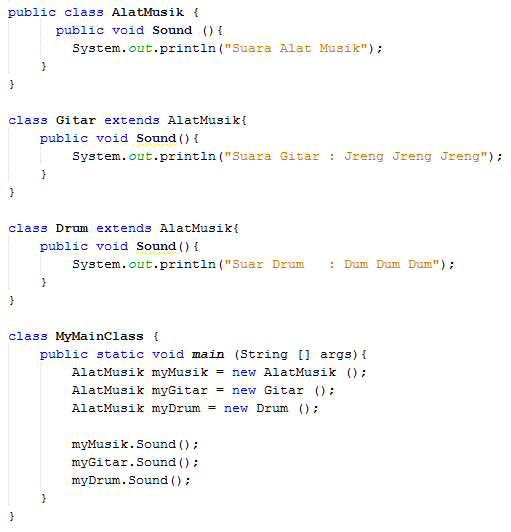


Berikut adalah kode lengkap dari beberapa class

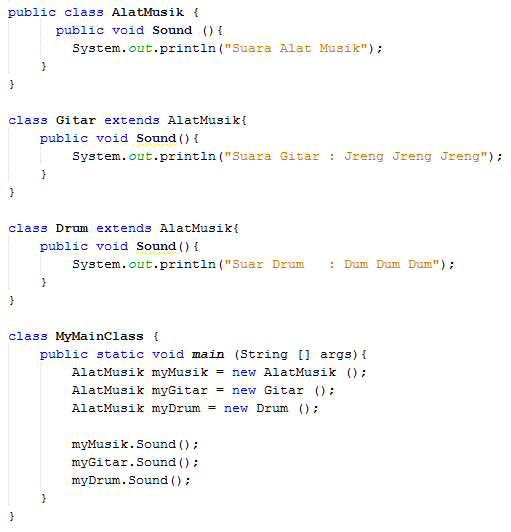
* Alat Musik

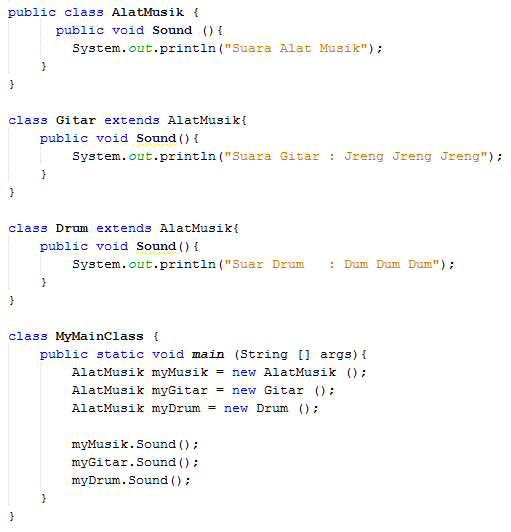


* Gitar

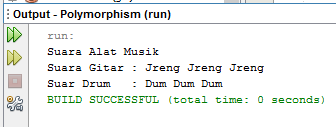


* Drum



* Main 

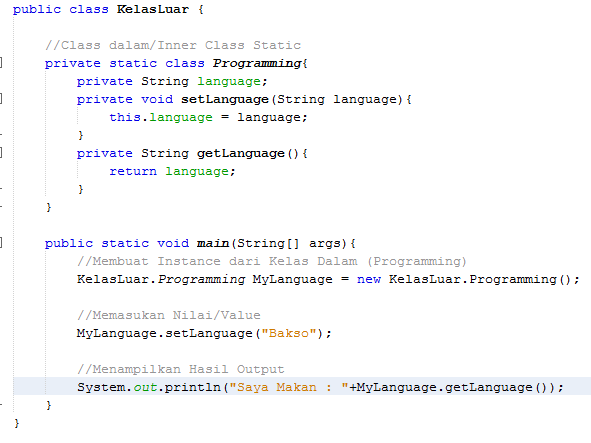
Hasil:



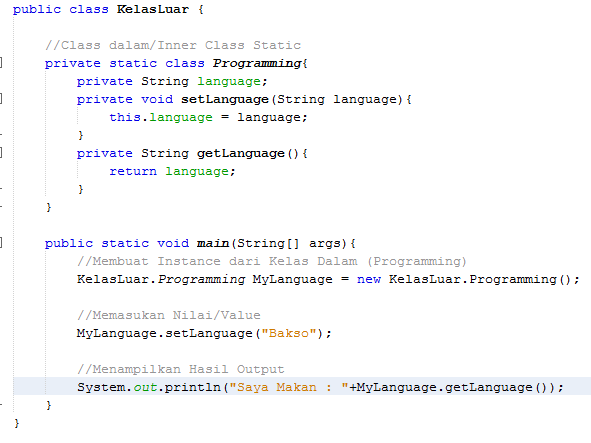
### **Java Inner Class**

* Static

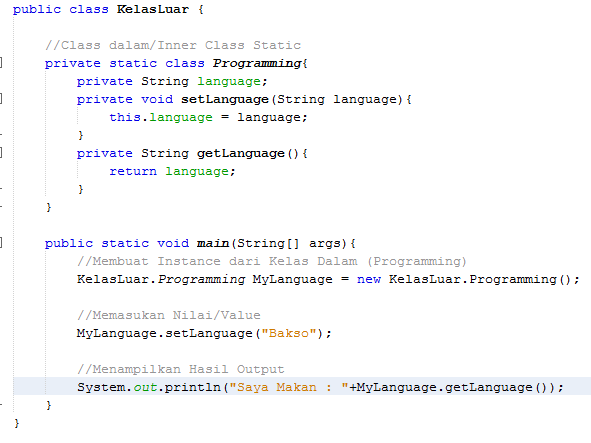
Untuk mengubah sebuah Inner Class menjadi Static, hanya perlu menambahkan kata kunci static di belakang class.



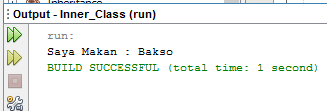
Inner class static hanya membuat instance dari class dalam saja.



Berikut adalah kode lengkap dari studi kasus inner class static

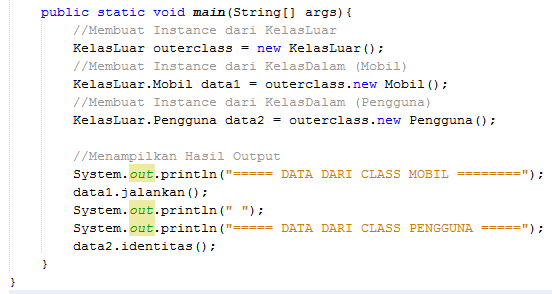


Hasil:

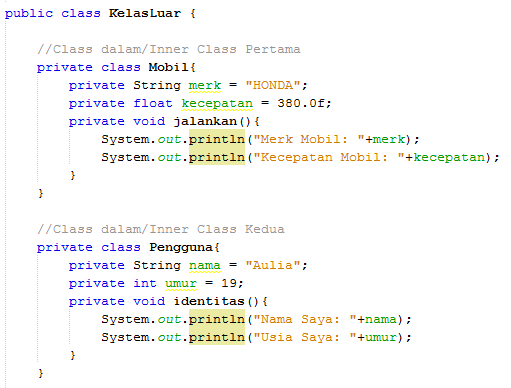


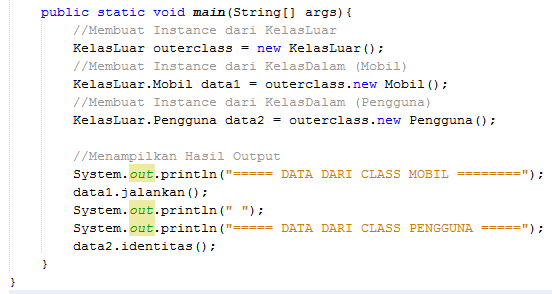
* Non-Static

Untuk mengakses variabel atau method pada class luar, perlu membuat instance/objek class luar di dalam inner class.

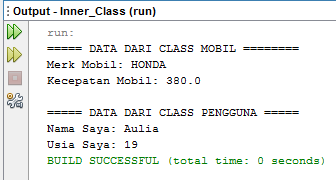


Source code lengkapnya seperti dibawah ini:



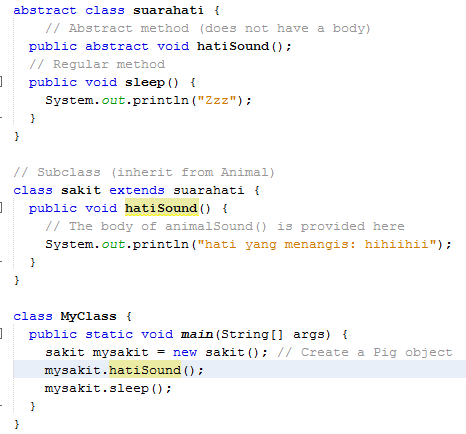


Hasil :

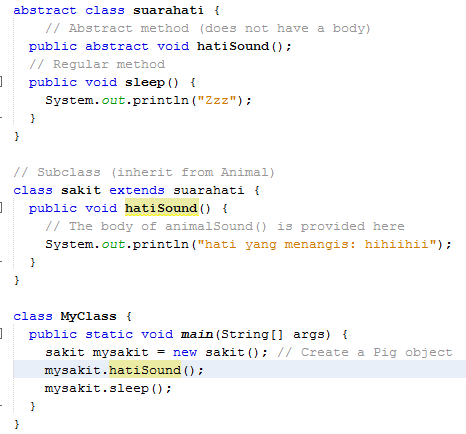


### **Java Abstrak**

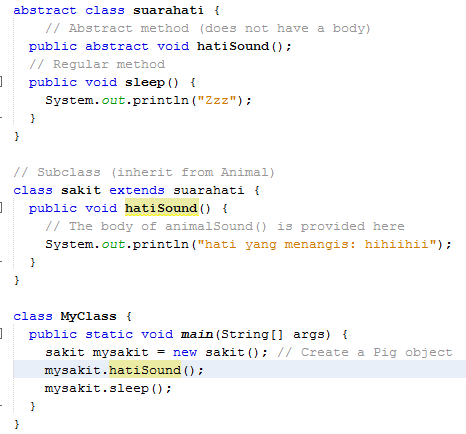
Untuk java abstraksi, digunakan tambahan kata kunci abstract pada classnya.



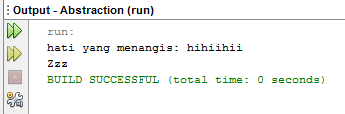
Pada abstract class tidak memiliki body atau method didalamnya.



Berikut adalah code lengkapnya:

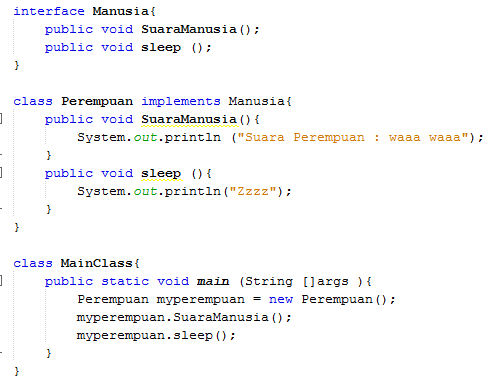


Hasil :



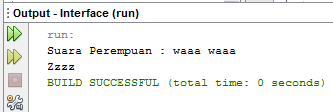
### **Java Interface**

Berikut adalah studi kasus dari java interface, dengan membuat class bernama Manusia.



Class MainClass menggunakan interface Manusia dengan keyword implements. Program diatas, menggunakan interface sebagai induknya, yang membedakan dengan class induk yaitu interface hanya mendeklarasikan method-method nya. Kemudian method tersebut di implementasikan oleh class-class yang mengimplementasikan interface tersebut.

Hasil output:

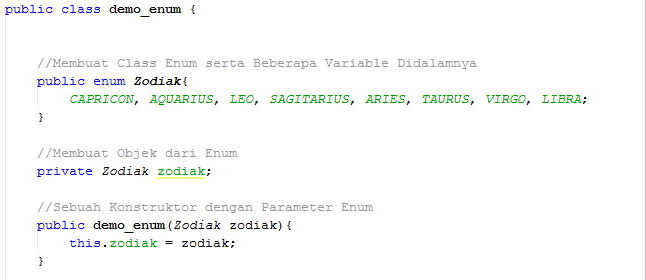


### **Java Enums**

Enum merupakan salah satu class yang dapat digunakan untuk memuat kumpulan variabel yang telah didefinisikan menjadi tipe data konstanta. Setiap objek baru yang didefinisikan berdasarkan tipe data enum, objek tersebut dapat mengakses atribut atau method (Enum) lalu mengaplikasikan nilai atau value yang ada di dalamnya.

Pada studi kasus java enum, telah dibuat class enum bernama demo\_enum, di dalam Main class, yang terdiri dari nama-nama zodiac, dimulai dari zodiac aquarius.

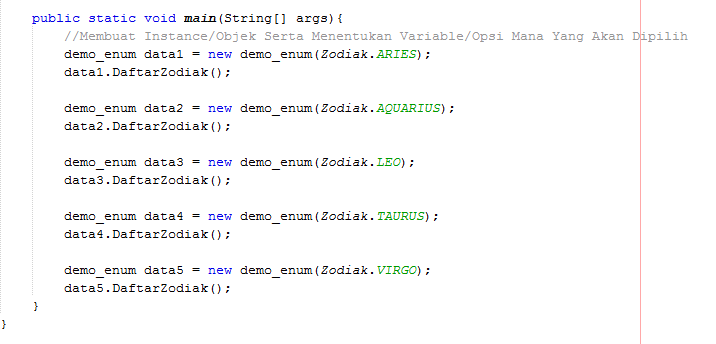
Pada baris code, dibuat variabel refernsi zodiac, lalu membuat constructor milik Main Class.



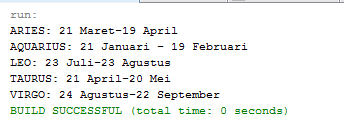
Untuk menentukan nilai output, berdasarkan variabel yang dideklarasikan pada enum zodiac, disini menggunakan switch case yang didefinisikan di dalam method yang bernama DaftarZodiak.



Setelah itu, pada method lain, akan di eksekusi program java dengan cara membuat dan mengintalasi objek baru, llau memanggil method DaftarZodiak().



Hasil:



### **Java User Input (Scanner)**

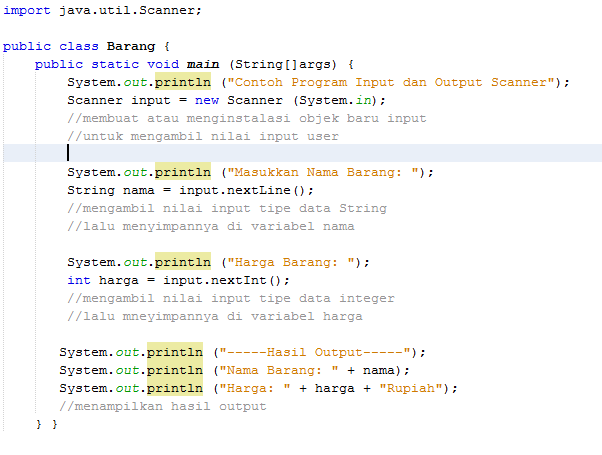
Pada studi kasus java user input, pertama akan dibuat atau menginstalasi objek baru kedalam class scanner yaitu input.



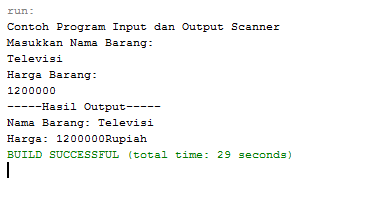
Disini, digunakan system.in agar user atau pengguna dapat menginputkan nilai lewat keyboard. Pada code in, merupakan objek dari class InputStream yang didefinisikan pada class System.

Kemudian objek tersebut digunakan untuk mengakses method input.nextLine(), yang berfungsi untuk mengambil nilai input tipe data string, dan method input.nextInt() befungsi untuk mengambil nilai input tipe data integer.

Berikut ini adalah kode lengkap dari studi kasus java user input menggunakan scanner.



Hasil :



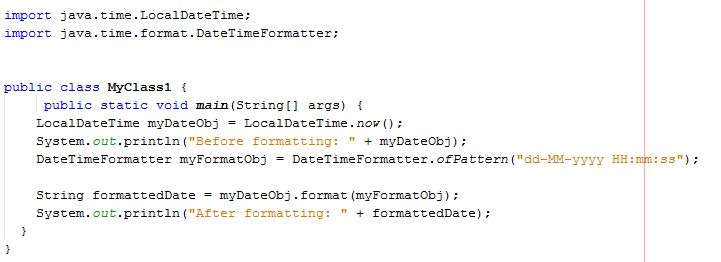
### **Java Date**

Pada studi kasus java date untuk menampilkan tanggal dan waktu saat ini, buat class dengan nama MyClass. Date untuk menampilkan tanggal sekarang dengan format yang diinginkan, sedangkan time untuk menampilkan waktu sekarang, waktu yang akan datang, waktu yang akan ditampilkan sesuai dengan inisiasi output waktu yang akan ditampilkan.

Berikut adalah source code lengkap untuk membuat tampilan tanggal dan waktu sekarang dengan format yang berbeda.

1. Tanggal: dd-MM-yyyy

Waktu: HH:mm:ss



Berikut penjelasan source code diatas:



Source code untuk menampilkan tanggal dan waktu sekarang, juga untuk memformat tanggal sesuai yang diinginkan.



Source code membuat object baru untuk menampilkan tanggal dan waktu saat ini sebelum di format.

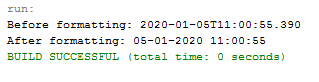


Source code membuat object baru untuk memformat tanggal dan waktu.



Source code untuk memanggil object baru yang telah di format.

Hasil :



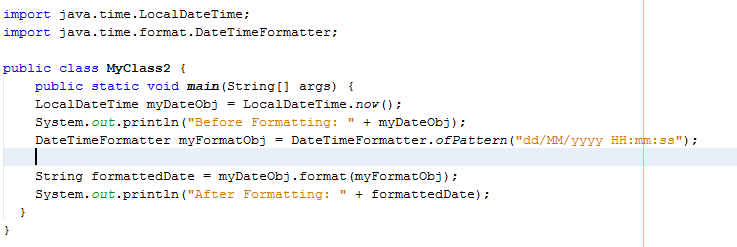
1. Tanggal: dd/MM/yyyy

Waktu: HH:mm:ss

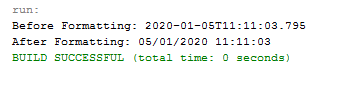
Source code untuk menampilkan format tanggal dan waktu saat ini hampir sama dengan sebelumnya,namun ada perbedaan sedikit pada source code untuk format tanggal dan waktunya.



Source code lengkapnya:



Hasil:



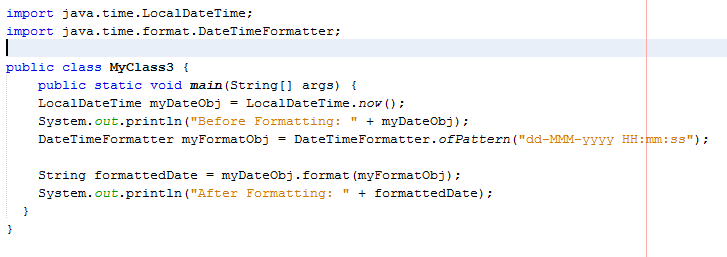
1. Tanggal: dd-MMM-yyyy

Waktu: HH:mm:ss

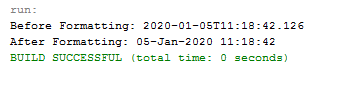
Source code untuk memformat tanggal dan waktu



Source code lengkapnya yaitu:



Hasil :



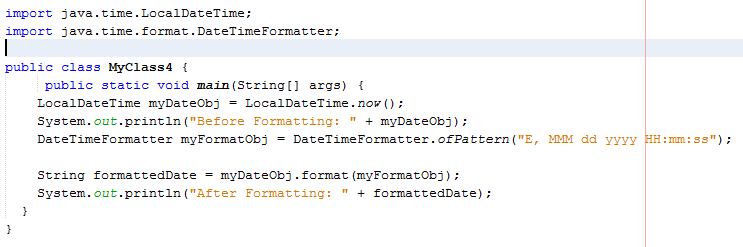
1. Tanggal: E, MMM dd yyyy

Waktu: HH:mm:ss

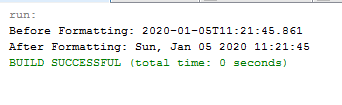
Source code untuk memformat tanggal dan waktu sesuai yang dituliskan.



Source code lengkap untuk menampilkan dan memformat tanggal dan waktu sesuai yang dituliskan.



Hasil:



### **Java ArrayList**

ArrayList merupakan class yang memungkinkan membuat sebuag object untuk menampung apapun.

Untuk menggunakan ArrayList, harus mengimpor terlebih dahulu.



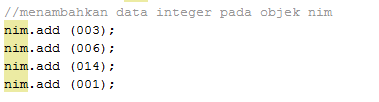
Kemudian, membuat object sesuai dengan tipe data.



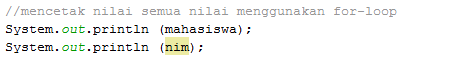
Source code diatas untuk membuat object baru dengan tipe data String. Gambar dibawah ini merupakan source code untuk membuat object dengan tipe data Integer.



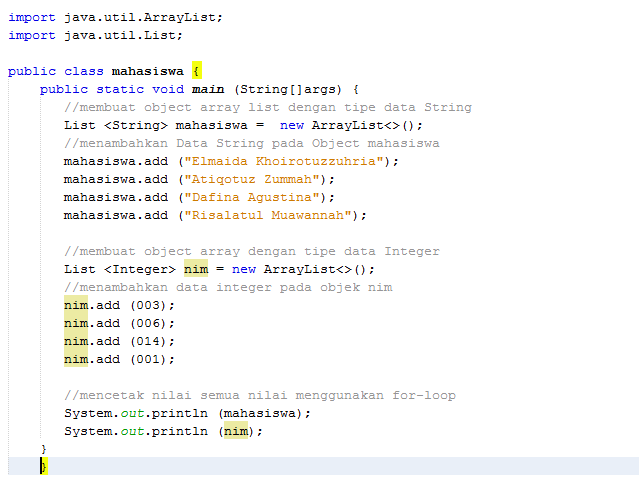
Source code untuk menambahkan data pada tipe data String menggunakan method put().



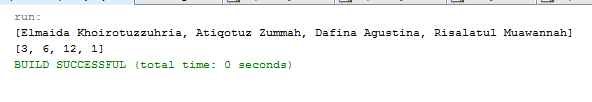
Source code untuk menambahkan data pada tipe data Integer menggunakan method put(). Setelah menambahkan data pada masing-masing object, kemudian panggil semua object menggunakan method for-loop.



Source code lengkapnya yaitu:



Hasil :

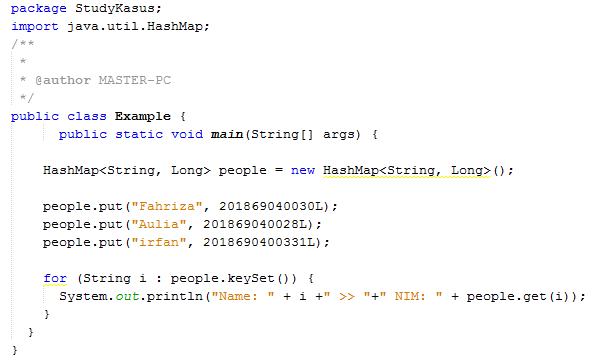


### **Java HashMap**

Sebelum menggunakan HashMap, terlebih dahulu harus mengimport HashMap pada package.

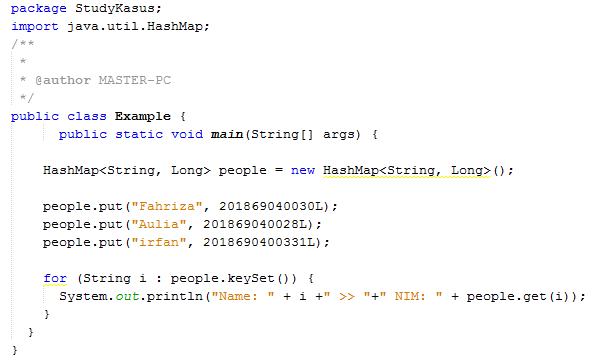
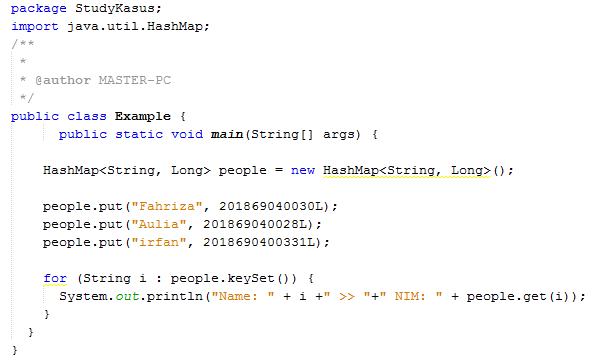


Untuk menggunakan HashMap, diharuskan untuk membuat objeknya terlebih dahulu. Object dari hashMap dapat dibuat dengan kata kunci **new.**

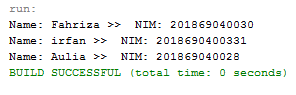


Pada source code diatas, telah dibuat object HashMap bernama books. Object ini dapat digunakan untuk menyimpan koleksi data

Contoh :

Hasil :



### **Java Wrapper**

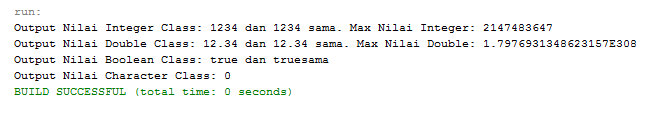
Source code untuk program java wrapper class yaitu:



Pada studi kasus diatas digunakan wrapper class seperti integer, double, Boolean, dan character. Kemudian menginstalasi objek tersebut dan menggunakan constructor untuk mendefinisikan nilainya pada pernyataan.

Kecuali pada character class, di dalam wrapper class dapat juga digunakan dua tipe pernyataan dengan menggunakan tipe data primitive pada umumnya dan juga bertipe data string. Untuk mengakses method dapat juga dilakukan secara langsung pada wrapper class dengan method keyword static yang telah didefinisikan di dalam wrapper class. Seperti contohnya **Integer.MAX\_VALUE,** dan juga **Double.MAX\_VALUE** yang digunakan untuk menampilkan output hasil, nilai maximal dari tipe data integer dan juga double.

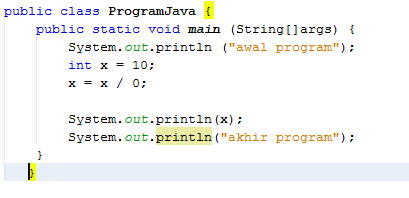
Hasil:



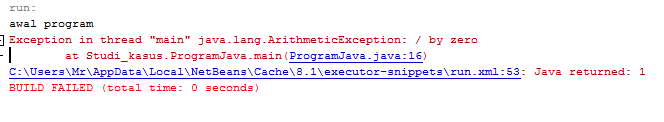
### **Java Exceptions**

Untuk menangani error pada java,digunakan sebuah statement yang bernama try..catch. statement tersebut digunakan untuk mengurung eksekusi yang menampilkan error dan dapat membuat program tetap berjalan tanpa dihentikan secara langsung. Error yang ditangani oleh try..catch biasa disebut dengan exception.

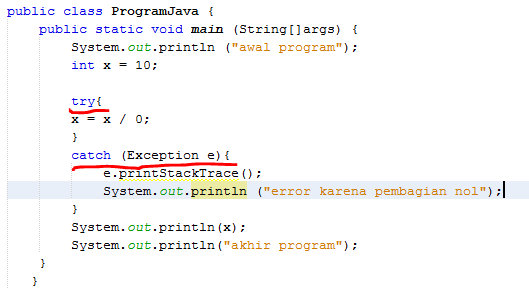
Untuk java exception, dibuat sebuah studi kasus yang mengandung error ketika dijalankan. Berikut adalah source code nya:



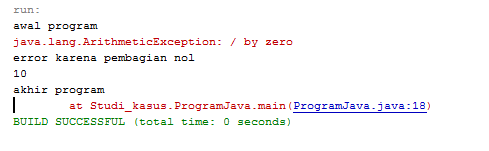
Ketika dijalankan maka akan mendapat hasil output sebagai berikut:



Berbeda ketika operasi pembagian nol diatas dikurung dengan try..catch, maka hasil eksekusi program akan sedikit berbeda.

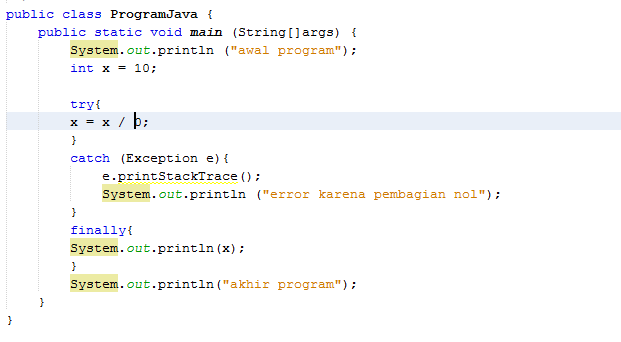


Hasil output:

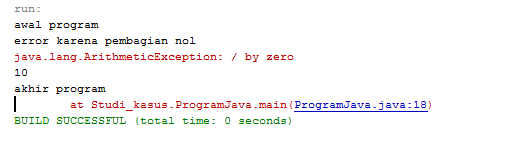


Ketika program di eksekusi kembali maka program akan dijalankan bagian akhir program walaupun terjadi error di tengah-tengah eksekusi.

Untuk menghindari error, akan digunakan statemen finally. Perhatikan source code berikut:



Ketika menjalankan program tersebut maka akan didapat:



Tetap diperoleh hasil error karena pembagi tidak boleh nol.

### **BAB V**

### **PENUTUP**

1. **KESIMPULAN**

Pemrograman Berorientasi Objek (*Object Oriented Programming* atau OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Objek adalah struktur data yang terdiri dari bidang data dan metode bersama dengan interaksi mereka untu merancang aplikasi dan program computer. Dalam OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek. Pembahasan mengenai orientasi objek tidak akan terlepas dari konsep objek seperti *inheritance* atau pewarisan, *encapsulation* atau pembungkusan, dan *polymorphism* atau lainnya. Konsep-konsep ini merupakan fundamental dalam orientasi objek yang perlu dipahami serta digunakan dengan baik, dan menghindari penggunaanya yang tidak tepat. Dalam laporan praktikum ini dilengkapi dengan studi kasus serta hasil percobaan pada setiap babnya.

1. **SARAN**
2. Laporan praktikum ini dapat dikembangkan lebih lanjut sehingga memajukan pendidikan teknologi informasi dan komunikasi Indonesia, terutama pada bidang studi pemrograman brorientasi objek menggunakan bahasa pemrograman java.

### **DAFTAR PUSTAKA**

W3Schools. 2019. *Java*.

<https://www.w3schools.com/java/default.asp>

(diakses tanggal 24 Desember 2019 pukul 10.09 WIB)