LAPORAN PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK



Laporan Untuk Melengkapi Tugas Terstruktur Mata Kuliah Pemrograman Berorientasi Object

Dosen Pengajar

Imron Rosadi, MM

Disusun oleh :

TEKNIK INFORMATIKA-3A

Ainun Gita Risqi Ilma (201869040092)

**UNIVERSITAS YUDHARTA** **PASURUAN**

Jl.Yudharta No.7 (Pondok Pesantren Ngalah)

Sengonagung Purwosari Pasuruan Telp.(0343)611186

**2019**

**KATA PENGANTAR**

Segala puji dan syukur kami panjatkan kehadirat Allah.S.W.T atas hidayahnya, sehingga kami dapat menyelesaikan laporan praktikum pada mata kuliah pemrgraman berorientasi objek dengan baik, penulisan laporan ini dibuat agar dapat memenuhi 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, karena dengan anugerahnya penulis dapat menyelesaikan laporan praktikum ini dengan baik.
2. Orang tua saya yang telah memberikan semangat dan dukungan sehingga laporan praktikum ini dapat terselesaikan dengan baik serta mendapatkan nilai yang diinginkan.
3. Bapak Imron Rosadi, MM selaku dosen pemrograman berorientasi objek
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.

Purwosari, 05 desember 2019

Penulis

Ainun Gita Risqi Ilma

**DAFTAR ISI**

**HALAMAN JUDUL i**

**KATA PENGANTAR i**

**DAFTAR ISI ii**

**BAB I PENDAHULUAN 1**

1. Latar Belakang 1
2. Tujuan 1
3. Manfaat 2

**BAB II LANDASAN TEORI 3**

A. Java OOP 3

B. Java Classes/Objects 3

C. Java Class Attributes 4

D. Java Class Methods 8

E. Java Constructors 13

F. Java Modifiers 15

G. Java Encapsulation 23

H. Java Package/API 26

I. Java Inheritance (Pewarisan) 29

J. Java Polymorphism 30

K. Java Inner Class 31

L. Java Abstraksi 33

M. Java Interface 35

N. Java Enum 38

O. Java User Input (Scanner) 40

P. Java Date 41

Q. Java ArrayList 48

R. Java HashMap 58

S. Java Wrapper Classes 61

T. Java Exceptions 64

**BAB III HASIL PERCOBAAN PRAKTIKUM 67**

A. Java Class Attributes 67

B. Java Class Methods 69

C. Java Constructor 70

D. Java Modifiers 71

E. Java Encapsulations 76

F. Java Package / API 77

G. Java Inheritance 78

H. Java Polymorphism 79

I. Java Inner Class 80

J. Java Abstraksi 82

K. Java Interface 83

L. Java Enum 84

M. Java User Input 86

N. Java Date 87

O. Java ArrayList 89

P. Java HashMap 93

Q. Java Wrapper 94

R Java Exceptions 96

**BAB IV STUDI KASUS 99**

1. Java Constructors 99
2. Java Modifiers 100
3. Java Encapsulation 101
4. Java Package / API 102
5. Java Inheritance 106
6. Java Polymorphism 109
7. Java Inner Class 110
8. Java Abstraksi 112
9. Java Interface 113
10. Java Enum 114
11. Java User Input (Scanner) 116
12. Java Date 117
13. Java ArrayList 120
14. Java HashMap 122
15. Java Wrapper 123
16. Java Exception 124

**BAB V PENUTUP 127**

1. Kesimpulan 127
2. Saran 127

DAFTAR PUSTAKA 128

**BAB I**

**PENDAHULUAN**

1. **Latar Belakang**

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. 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 paradigma 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.

1. **Tujuan Praktikum**
2. Untuk melengkapi nilai tugas mata kuliah Pemrograman Berorientasi Objek
3. Mampu membuat program java pada Netbeans berdasarkan aturan OOP
4. Untuk mengkaji kembali pengetahuan yang telah didapatkan pada kuliah Pemrograman Berorientasi Objek
5. **Manfaat**
6. Dapat melengkapi nilai tugas mata kuliah Pemrograman Berorientasi Objek
7. Dapat membuat program Java pada Netbeans berdasarkan ilmu aturan OOP
8. Dapat mengkaji kembali pengetahuan yang telah didapatkan pada kuliah Pemrograman Berorientasi Objek

**BAB II**

**Landasan Teori**

1. **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 beroentasi objek

Lihatlah ilustrasi berikut untuk melihat perbedaan antara kelas dan objek:

|  |  |
| --- | --- |
| KELAS  Buah | OBJEK  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.

1. **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 :

|  |
| --- |
| MyClass.java  //Buat kelas bernama " MyClass " dengan variabel x:  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 | |

1. **Java Class Attribute**

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:

|  |
| --- |
| Create a class called "MyClass" with two attributes: x and y:  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);  }  }   |  | | --- | | Output ;  5 | |

Kita juga bisa mengubah nilai attribute dengan :

* 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();  MyClass myObj2 = new MyClass();  myObj2.x = 25;  System.out.println(myObj1.x);  System.out.println(myObj2.x);  }  }   |  | | --- | | Output :  5  25 | |

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 :  John Doe  24 | |

1. **Java Class Methods:**

* 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();  }  }  Output:  Hello World! |

* Static Vs Non Static

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

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

|  |
| --- |
| 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  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 |

## **Access Methods With an Object**

Contoh :

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

***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

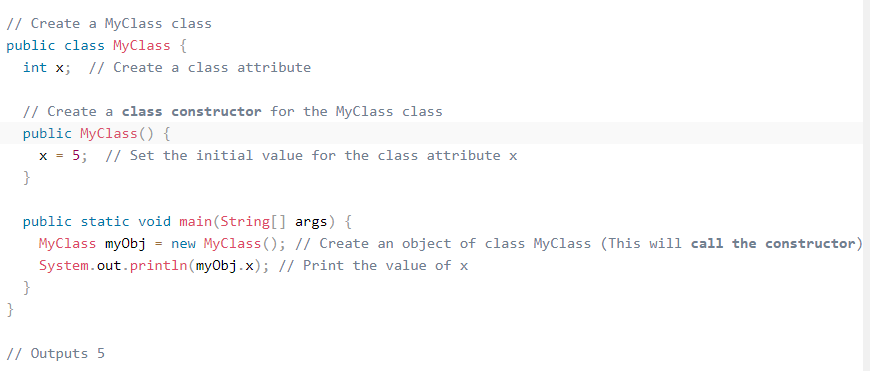
1. **Java Constructor :**

**- Java Constructors**

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:



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.**

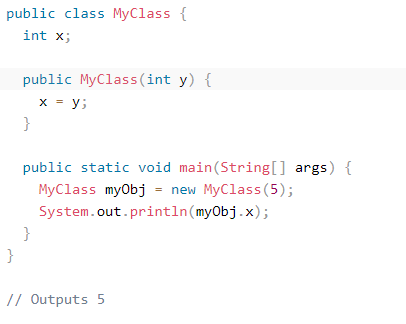
Semua kelas memiliki konstruktor secara default: jika Anda tidak membuat konstruktor kelas sendiri, Java membuat satu untuk Anda. Namun, maka Anda tidak dapat menetapkan nilai awal untuk atribut objek.

**- Constructor Parameters**

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

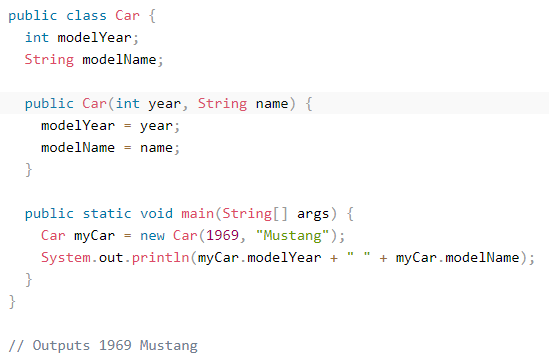
Contoh berikut menambahkan parameter int y ke konstruktor. Di dalam konstruktor kita atur x ke y (x = y). Ketika kita memanggil konstruktor, kita meneruskan parameter ke konstruktor (5), yang akan menetapkan nilai x ke 5:

Contoh



Anda dapat memiliki banyak parameter yang Anda inginkan:

Contoh



1. **Java Modifier :**

**C:\Users\S a m s u n g\Pictures\cici modifiers\1.PNG**Sekarang, Anda cukup akrab dengan kata kunci **public**yang muncul di hampir semua contoh kami:

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

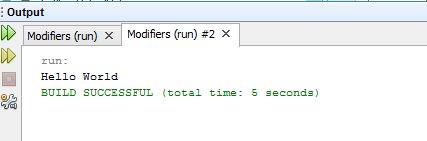
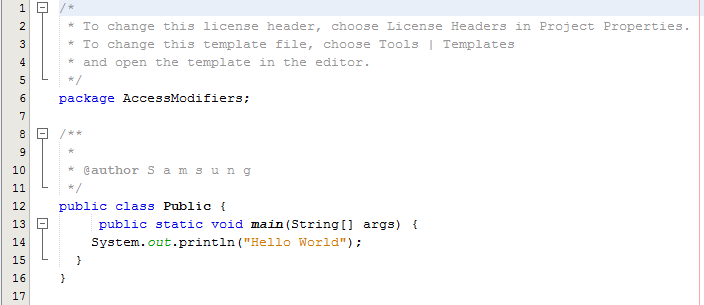
## Akses Pengubah

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

**Modifiers** : Public

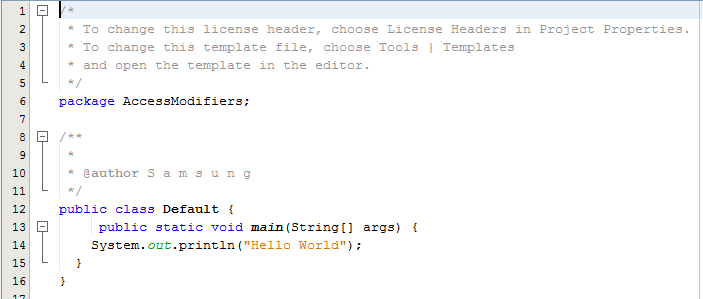
**Deskripsi** : kelas ini diakses oleh lain kelas

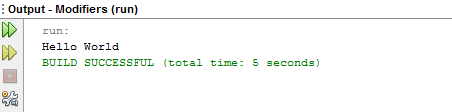
Contoh :

**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 :

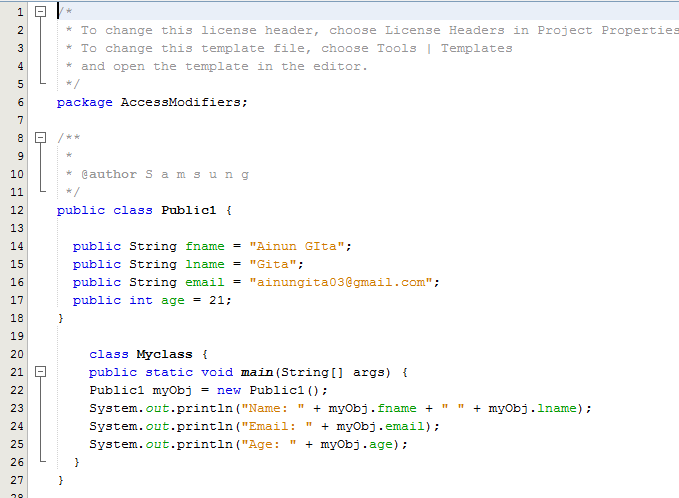
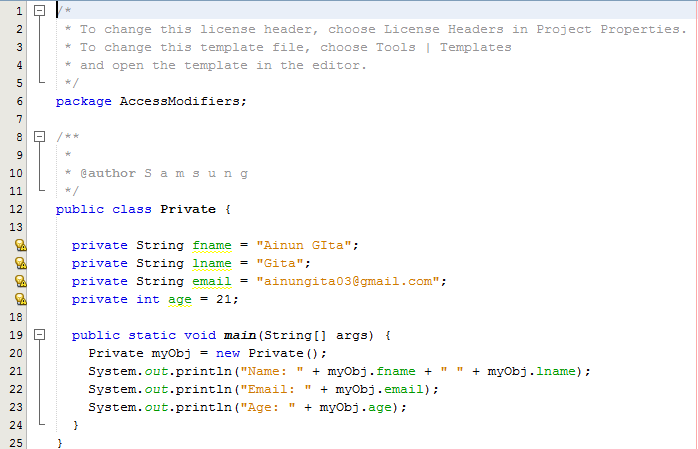
Hasil:

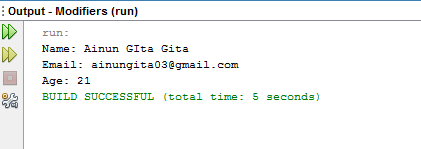
Untuk **atribut, metode, dan konstruktor** , Anda dapat menggunakan salah satu dari yang berikut ini:

**Modifiers** : public

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

Contoh :

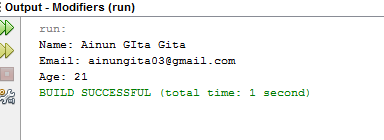


**Hasil :**

**Modifiers** : private

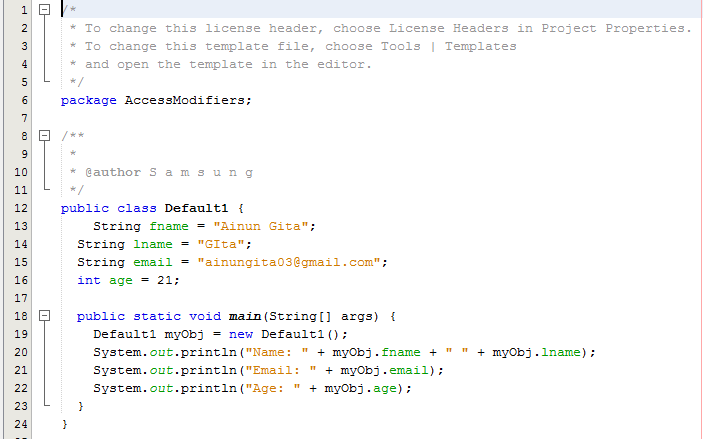
**Deskripsi**  : Kode ini m,hanya diakses dalam menyatakan kelas

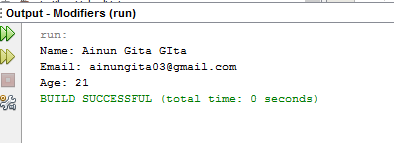
Contoh :

Hasil :

**Modifiers** : Default

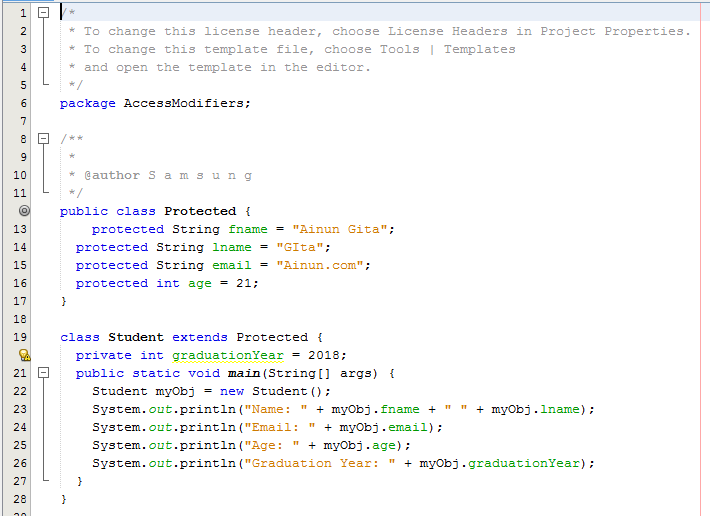
**Deskripsi**  : Kode tidak hanya diakses disama paket. Ini adalah digunakan ketika anda tidah spesifysebuah modifier. Anda akan belajar lebih banyak tentang packages di sebuah packages bab

Contoh :

Hasil :

**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 :

## C:\Users\S a m s u n g\Pictures\cici modifiers\21.PNGHasil :

## Non-Access Modifiers

Untuk **kelas** , Anda dapat menggunakan final atau abstract :

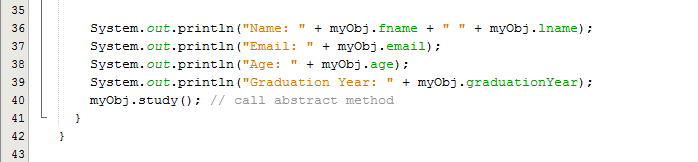
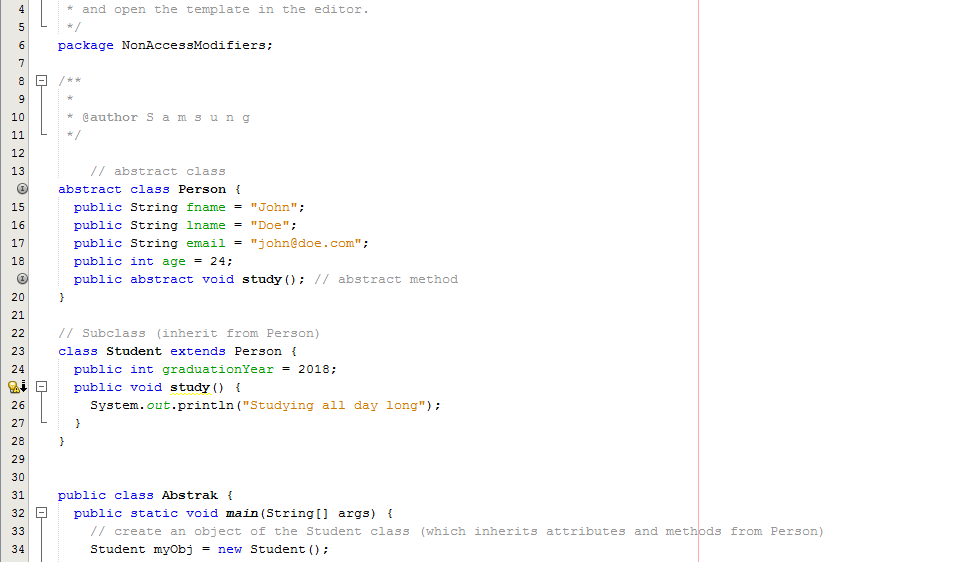
|  |  |
| --- | --- |
| **Modifiers** | **Deskripsi** |
| Final | Kelas tidak dapat diwarisi oleh kelas lain (Anda akan belajar lebih banyak tentang warisan dalam bab Warisan) |
| Abstract | Kelas tidak dapat digunakan untuk membuat objek (Untuk mengakses kelas abstrak, itu harus diwarisi dari kelas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

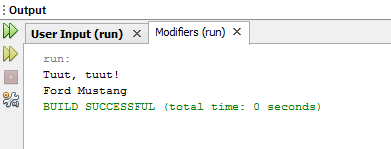
Untuk **atribut dan metode** , Anda dapat menggunakan salah satu dari yang berikut:

|  |  |
| --- | --- |
| **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" |

## **Keyword Final**

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

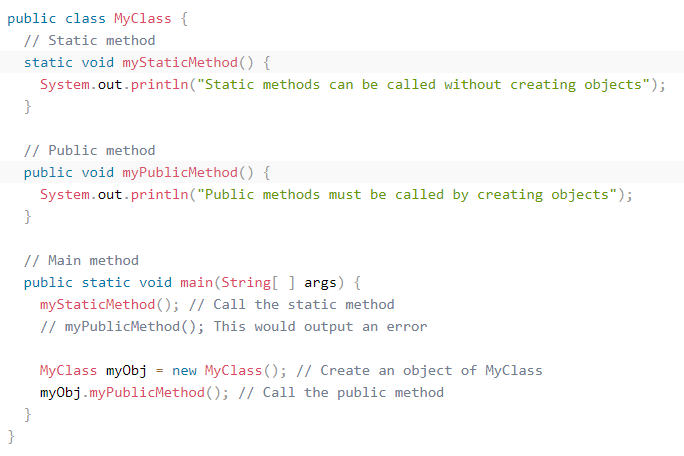
**Contoh :**

Hasil

## Statis

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

Contoh

Contoh untuk menunjukkan perbedaan antara metode static dan public :

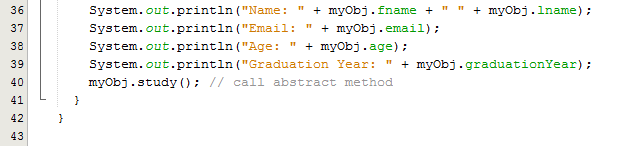
##### Hasil :

C:\Users\S a m s u n g\Pictures\cici modifiers\5.PNG

## Abstrak

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

**Contoh**



1. **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 variabel

**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)

1. *public class Person {*

*public String name;*

membuat kelas Person , menggunakan tipe variabel String karena mengambil contoh ‘nama orang’.

*//getter*

*public String getName(){*

*return name;*

*}*

Metode get artinya mengembalikan nilai variabel nama.

*//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’.

*public class MyClass {*

menggunakan public karena akan di akses dari luar kelas

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

*Person myObj = new Person();*

digunakan untuk Menambahkan objek ke dalam kelas

*myObj.setName("John");*

digunakan untuk mengisi data kedalam objek

*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

1. **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)

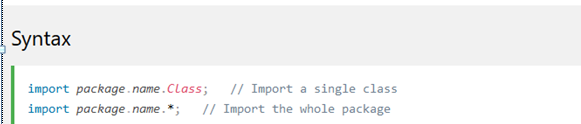
**Paket Bawaan**

Java API adalah pustaka kelas yang sudah ditulis sebelumnya, yang bebas digunakan, termasuk dalam Java Development Environment.

Perpustakaan berisi komponen untuk mengelola input, pemrograman basis data, dan banyak lagi lainnya. 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** :



## 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

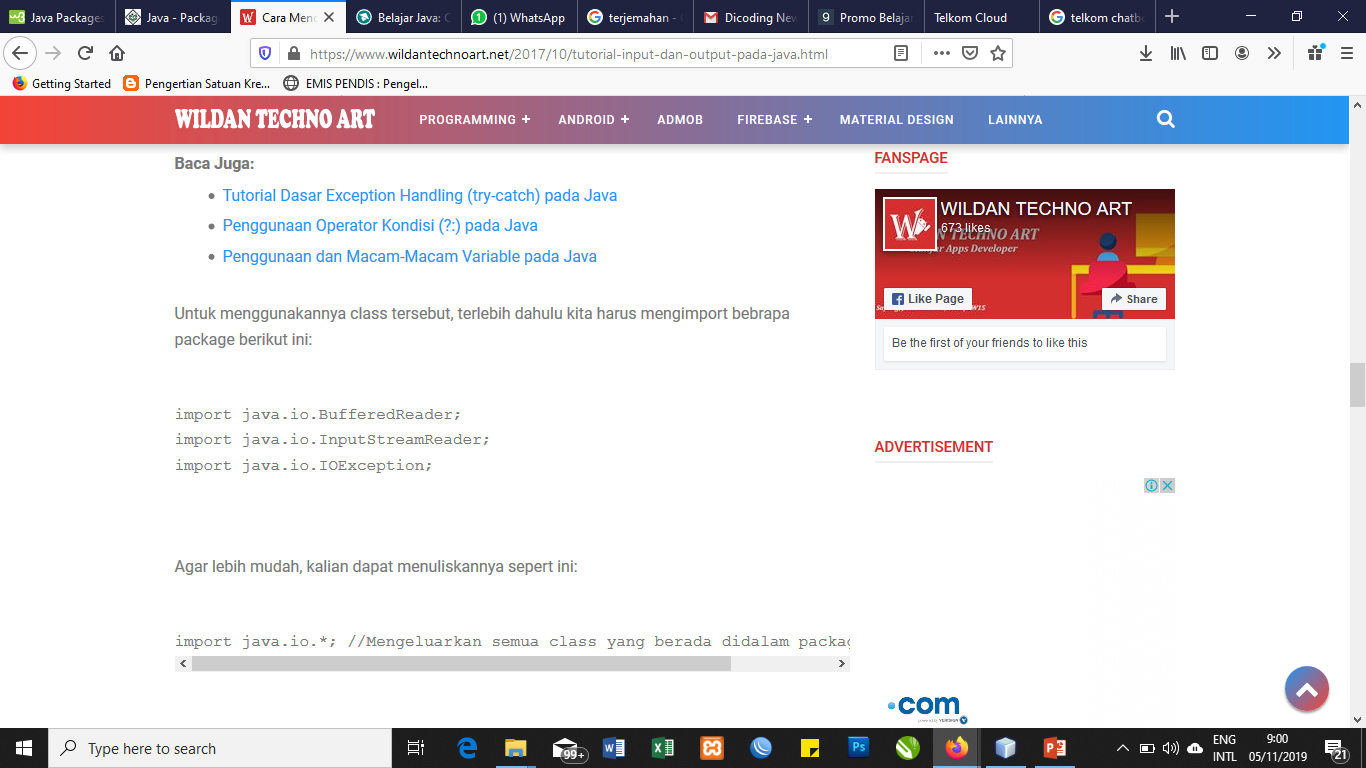
**nextByte** - Menerima bilangan bulat/Byte

**nextLong**  - Menerima bilangan bulat/Long

**nextShort**  - Menerima bilangan bulat/Short

* **Buffer Reader**

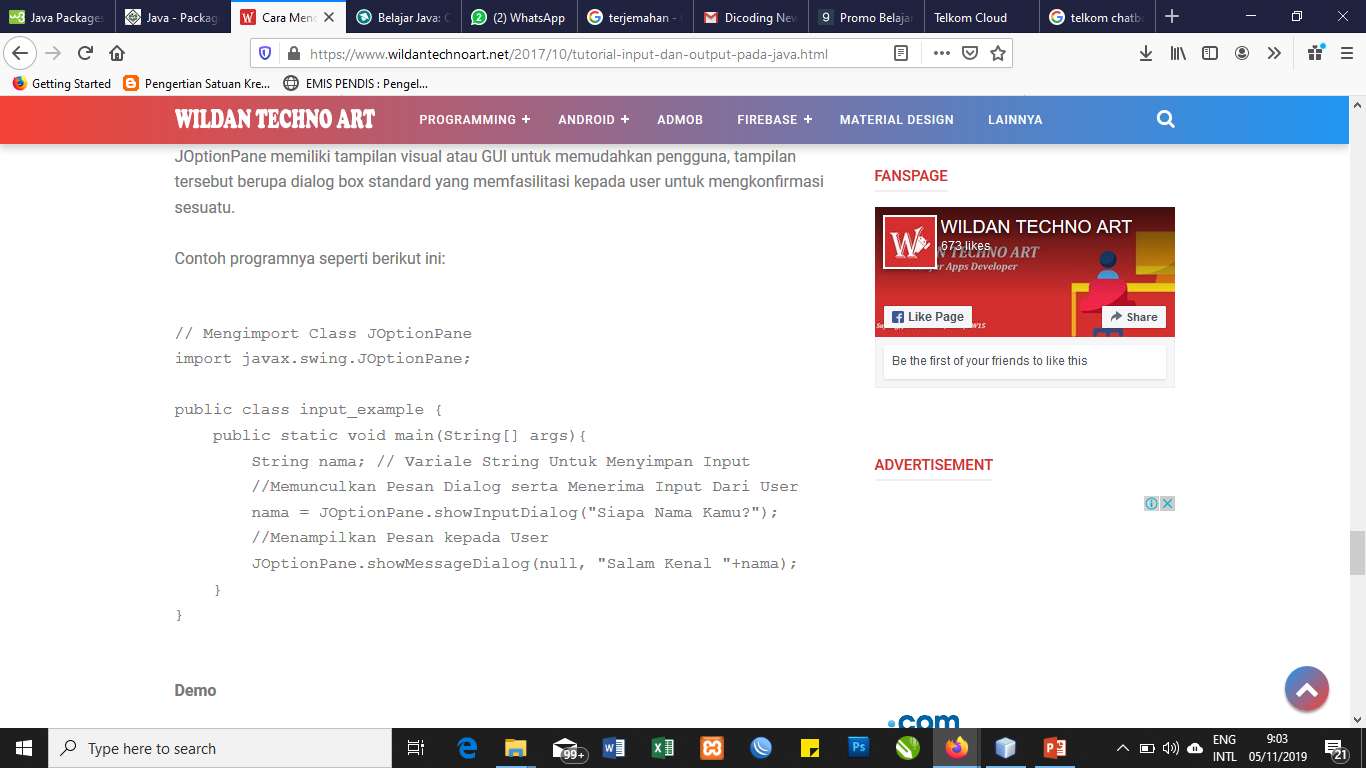
Pada bagian ini, kita akan menggunakan class BufferedReader, class tersebut berada di dalam package *java.io*, class ini dignakan untuk mendapatkan input dari user, hampir sama seperti Scanner, yang membedakan hanya cara penulisan syntaxnya saja.



* **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.

1. **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.

**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.

Contohnya:

|  |
| --- |
| public final class vehicle{  }  public class car extends vehicle {  public static void main (String args[]) {  }  } |

Output nya :

|  |
| --- |
| 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.

1. **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.

* 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 behaviournya masih abstrak dan belum diimplementasikan.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.

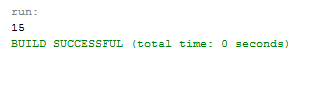
1. **Java Inner Classes :**

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:

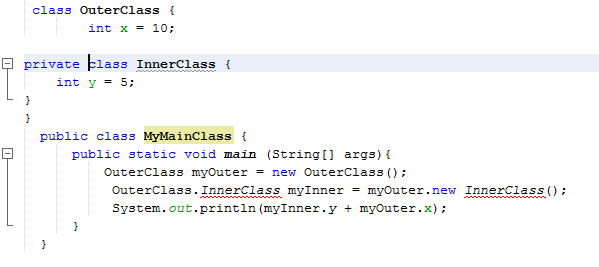


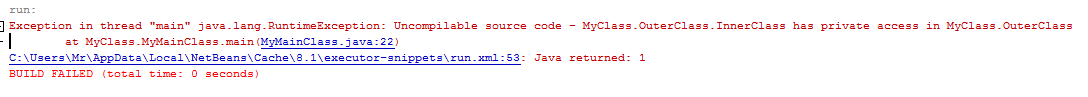
Akan didapatkan hasil running seperti ini :



**Private Inner Class**

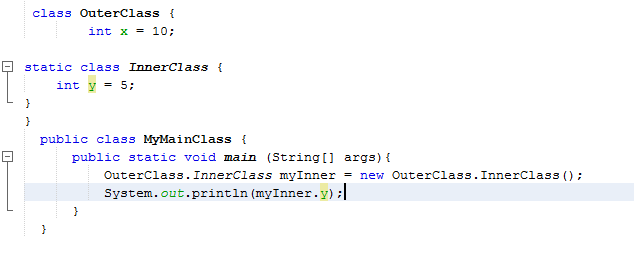
Tidak seperti kelas regular, kelas dalam bisa **private**  atau **protected.** Jika tidak ingin object luar mengakses kelas dalam, maka nyatakan dalam kelas sebagai **private**. Seperti contoh dibawah ini:



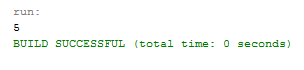
Jika dilihat dalam gambar, maka muncul garis merah pada syntax yang salah. Dan jika tetap ingin mengakses private inner class dari kelas luar (MyMainClass), maka terdapat kesalahan (error):

**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:



D akan memperoleh hasil running seperti gambar dibawah ini:



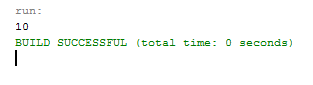
Perlu diingat bahwa, sama seperti **static** atribut dan metode, **static**  kelas dalam tidak memiliki akses ke anggota kelas kelas luar.

**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:



Akan didapatkan hasil dari running program tersebut:



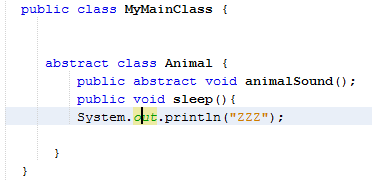
1. **Java Abstraction :**

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).

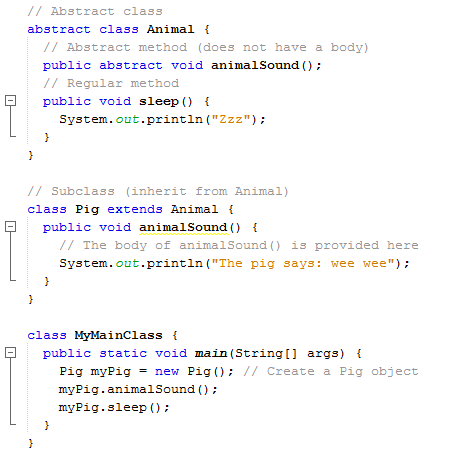
Kelas abstrak dapat memiliki metode abstrak dan regular, seperti gambar dibawah ini :



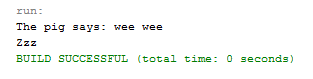
Dari contoh diatas, tidak mungkin membuat objek dari kelas Animal:



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.**



Hasil dari running diperoleh seperti gambar berikut:



1. **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 Animal {  public void animalSound(); // interface method (does not have a body)  public void sleep(); // interface method (does not have a body)  }  class Pig implements Animal {  public void animalSound() {  System.out.println("The pig says: wee wee");  }  public void sleep() {  System.out.println("Zzz");  }  }  class MyMainClass {  public static void main(String[] args) {  Pig myPig = new Pig();  myPig.animalSound();  myPig.sleep();  }  }  Result:  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).

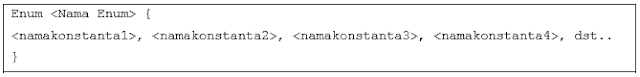
2) Java tidak mendukung "multiple inheritance" (sebuah kelas hanya dapat diwarisi dari satu superclass). Namun, ini dapat dicapai dengan interface, karena kelas dapat **mengimplementasikan** banyak interface. **Catatan:** Untuk mengimplementasikan beberapa interface, pisahkan dengan koma (lihat contoh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

|  |
| --- |
| interface FirstInterface {  public void myMethod(); // interface method  }  interface SecondInterface {  public void myOtherMethod(); // interface method  }  // DemoClass "implements" FirstInterface and SecondInterface  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();  }  }  Result:  Some text... Some other text... |

1. **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:

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

* Enum dalam pernyataan SWITCH-CASE

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 :

|  |
| --- |
| switch (value)  {  case constanta 1 : statement 1;  break;  case constanta 2 : statement 2;  break;  ……  ….etc  default : statement x;  } |

Keterangan:

* **Constanta 1, constanta 2,** dst adalah yang akan dicocokkan dengan isi value.
* **Statement** adalah pernyataan yang akan di kerjakan jika **value (bertipe Integer atau Karakter)**cocok dengan salah satu data dari **constanta 1 , constanta 2 atau yang lainnya**.
* **Break** adalah perintah untuk mengakhiri **statement**. Apabila tidak ada break, komputer akan mengeksekusi intruksi yang berada di bawahnya walaupun berada di case yang berbeda.
* **Default**, bersifat optional. dieksekusi jika **value** tidak cocok dengan salah satu **constanta**yang tersedia.
* **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 (EnumClass obj : EnumClass.values()){  System.out.println (obj); |

1. **Java User Input :**

##### Input Pengguna Java

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

## Jenis Input

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

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

1. **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 .

Sebagai 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:

|  |
| --- |
| *package date;*  *import java.time.LocalDate;* |

Pada bagian ini, kita melakukan impor Date (tanggal) yang dibutuhkan pada program.

Setelah kita mengimpor, lalu membuat kelas tanggal

|  |
| --- |
| *public class MyClass {*  *public static void main(String[] args) {*  *LocalDate myObj = LocalDate.now();* |

LocalDate myObj digunakan Untuk menambahkan objek tanggal dengan menggunakan metode Now yang artinya tanggal saat ini .

*System.out.println(myObj);*

}

}

Selanjutnya , kita memanggil objek atau menampilkan hasil outputannya .

* **Menampilkan Waktu saat ini**

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

Contoh:

*import java.time.LocalTime;*

Pada bagian ini, kita melakukan impor Time (waktu) yang dibutuhkan pada program. Setelah itu, kita membuat kelas waktu

|  |
| --- |
| *public class MyClass {*  *public static void main(String[] args) {*  *LocalTime myObj = LocalTime.now();* |

LocalTime myObj digunakan Untuk menambahkan objek waktu dengan menggunakan metode Now yang artinya waktu saat ini .

*System.out.println(myObj);*

*}*

*}*

Selanjutnya , kita memanggil objek atau menampilkan hasil outputannya .

* **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;*

pada bagian ini kita melakukan impor Date dan Time yang dibutuhkan pada program. Setelah itu, kita membuat kelas

|  |
| --- |
| *public class MyClass {*  *public static void main(String[] args) {*  *LocalDateTime myObj = LocalDateTime.now();* |

LocalDateTime myObj digunakan Untuk menambahkan objek tanggal dan waktu dengan menggunakan metode Now yang artinya tanggal dan waktu saat ini .

*System.out.println(myObj);*

*}*

*}*

.

**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;*

kita mengimpor tanggal dan waktu pada program

*import java.time.format.DateTimeFormatter;*

Selanjutnya, mengimpor format tanggal dan waktu yang akan kita atur

Setelah kita mengimpor lalu kita membuat kelas

|  |
| --- |
| *public class MyClass {*  *public static void main(String[] args) {*  *LocalDateTime myDateObj = LocalDateTime.now();* |

LocalDateTime myObj digunakan Untuk menambahkan objek tanggal dan waktu dengan menggunakan metode Now yang artinya tanggal dan waktu saat ini .

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

Ini adalah fungsi untuk menampilkan teks ke layar output-annya dengan menambahkan objek nya.

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

Selanjutnya kita mengubah format tanggal dan waktu , dengan format *ofPattern* yang menerima segala macam nilai .

*String formattedDate = myDateObj.format(myFormatObj);*

Setelah itu, menambahkan objek yang akan kita buat dalam memebrikan format yang berbeda pada tanggal dan waktu.

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

Kita memanggil hasil outputan yang dudah di edit pada format yang di ubah.

}

}

The 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 java.time.format.DateTimeFormatter;*

*public class MyClass1 {*

*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);*

*}*

*}*

Hasil Output

Before Formatting: 2019-12-22T22:01:31.570548  
After Formatting: 22-12-2019 22:01:31

2. dd/MM/yyyy

*import java.time.format.DateTimeFormatter;*

*public class MyClass2 {*

*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);*

*}*

*}*

Hasil Output

Before Formatting: 2019-12-22T21:46:15.420247  
After Formatting: 22/12/2019 21:46:15

1. dd-MMM-yyyy

*import java.time.LocalDateTime;*

*import java.time.format.DateTimeFormatter;*

*public class MyClass3 {*

*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);*

*} }*

Hasil Output

Before Formatting: 2019-12-22T21:52:55.657506  
After Formatting: 22-Dec-2019 21:52:55

1. E, MMM dd yyyy

*import java.time.LocalDateTime;*

*import java.time.format.DateTimeFormatter;*

*public class MyClass4 {*

*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);*

*}*

*}*

Hasil Output

Before Formatting: 2019-12-22T21:58:40.868365  
After Formatting: Sun, Dec 22 2019 21:58:4

1. **Java Array List :**

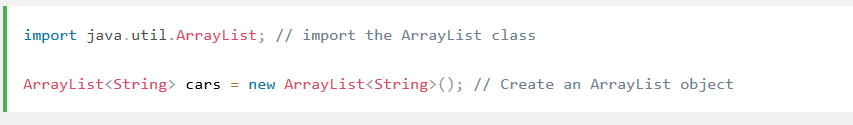
**Java Array List**

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:

****

* **Tambahkan Item**

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

Contoh :

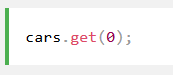
Hasil :

Run Add

1. **Akses suatu Item**

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

Contoh :





Hasil :

Run acess

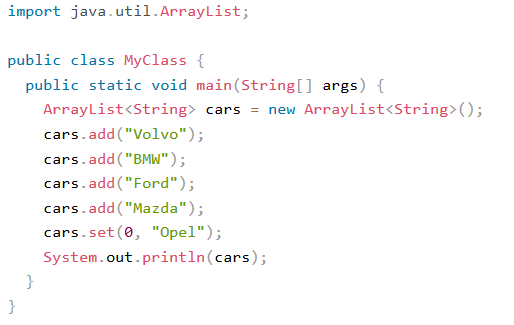
**Ingat:** Indeks array dimulai dengan 0: [0] adalah elemen pertama. [1] adalah elemen kedua, dll.

* **Ubah Item**

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

Contoh :





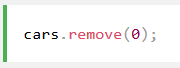
Hasil :

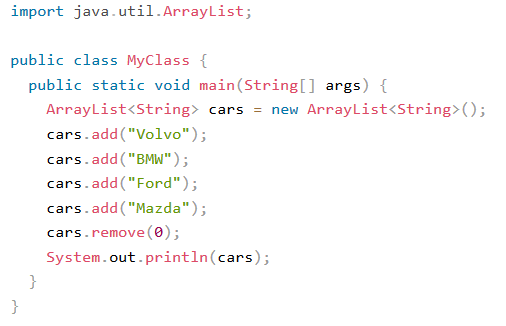
Run Change

* **Hapus Sebuah Item**

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

Contoh :

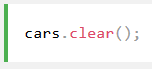




Hasil :

Run Remove

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





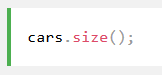
Hasil :

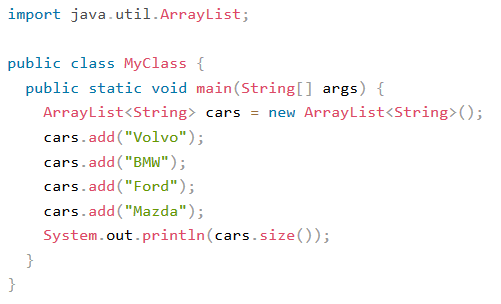
Run Clear

* **Ukuran ArrayList**

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

Contoh :





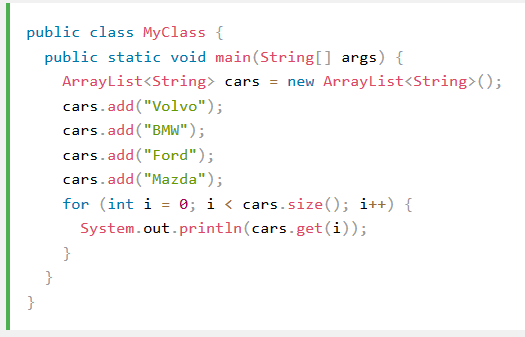
Hasil :

Run size

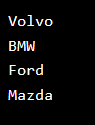
## Loop Through ArrayList

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

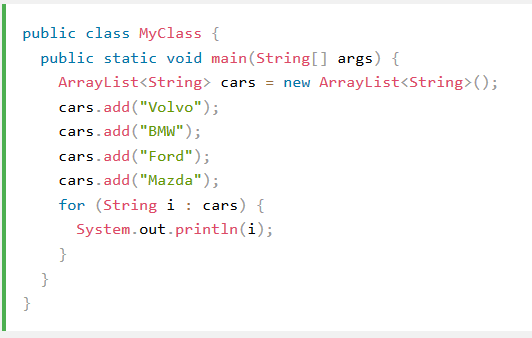
### Contoh



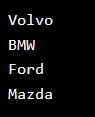
Hasil :



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



Hasil :



## Tipe yang lain

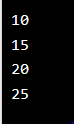
Elemen dalam ArrayList sebenarnya adalah objek. Dalam contoh di atas, kami membuat elemen (objek) dari tipe "String". Ingatlah bahwa String di Java adalah objek (bukan tipe primitif). Untuk menggunakan tipe lain, seperti int, Anda harus menentukan [kelas pembungkus yang](https://translate.googleusercontent.com/translate_c?depth=1&hl=id&prev=search&rurl=translate.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 ):



Hasil :

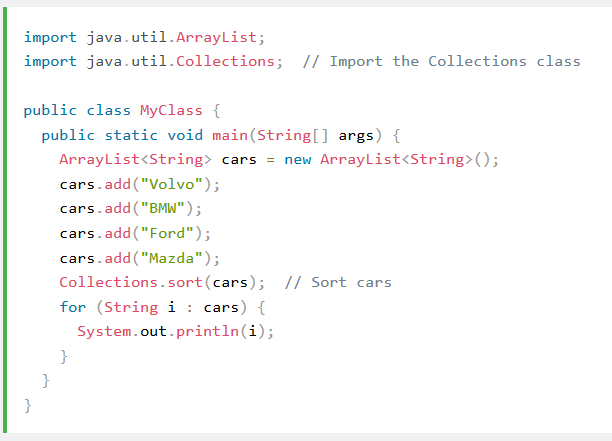


## Sortir Daftar Array

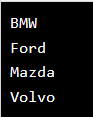
Kelas lain yang berguna dalam paket java.util adalah kelas Collections , yang mencakup metode sort() untuk menyortir daftar berdasarkan abjad atau angka:

### Contoh

Sortir ArrayList of Strings:



Hasil :

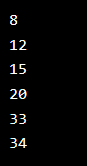


### Contoh

Sortir ArrayList of Integer:



Hasil :



1. **Java HashMap :**

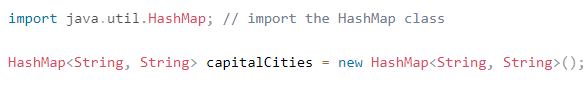
* **Java HashMap**

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 :



* **Add Item**

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



## Access an Item

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

Contoh

C:\Users\S a m s u n g\Pictures\hashmap\3.PNG

## Remove an Item

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

Contoh

C:\Users\S a m s u n g\Pictures\hashmap\4.PNG

Untuk menghapus semua item, gunakan metode clear() :

Contoh

C:\Users\S a m s u n g\Pictures\hashmap\5.PNG

## Ukuran HashMap

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

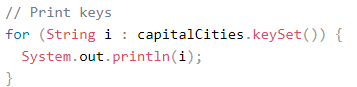
C:\Users\S a m s u n g\Pictures\hashmap\6.PNG

## Loop Through HashMap

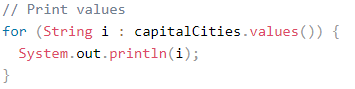
Ulangi item-item dari HashMap dengan **setiap** loop.

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

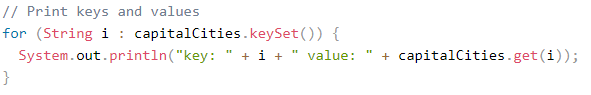
**Contoh**



Contoh



Contoh

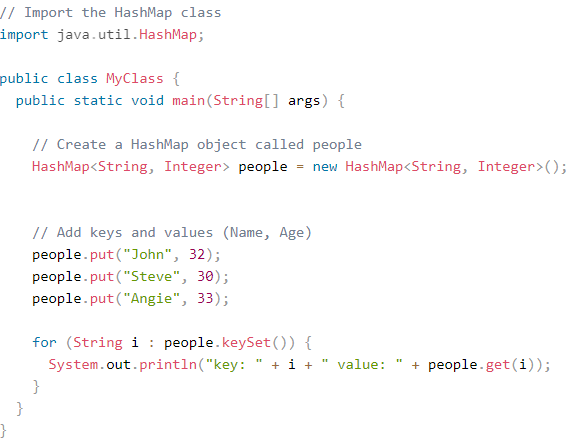


* **Tipe yang lain**

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 :



1. **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):

|  |
| --- |
| **MyClass.java**    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:

|  |
| --- |
| **MyClass.java**    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:

|  |
| --- |
| MyClass.java    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.

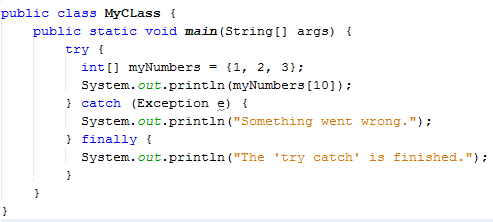
Dalam contoh berikut ini, kami mengonversi Integer ke String , dan menggunakan metode length() dari kelas String untuk menampilkan panjang "string":

|  |
| --- |
| MyClass.java  public class MyClass {  public static void main(String[] args) {  Integer myInt = 100;  String myString = myInt.toString();  System.out.println(myString.length());  }  }  Result:  3 |

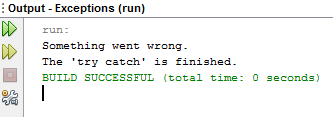
1. **Java Exceptions :**

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

Contoh :



Outputan:



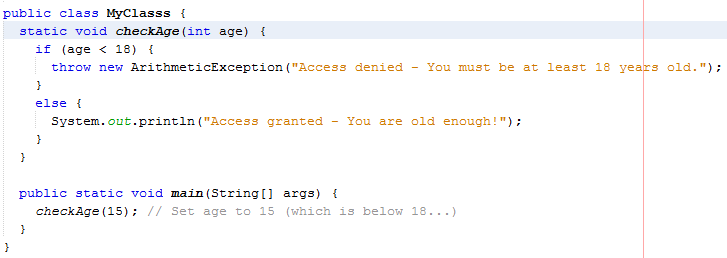
Kata kunci lemparan

Pernyataan **throw** memungkinkan saya untuk membuat kesalahan khusus.

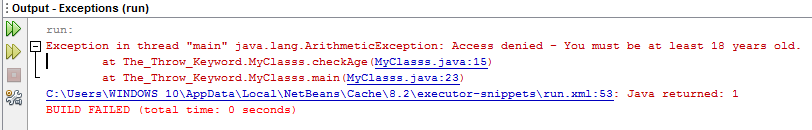
Pernyataan **throw** digunakan bersama dengan **tipe pengecualian** . Ada banyak jenis pengecualian yang tersedia di Jawa: ArithmeticException , FileNotFoundException , ArrayIndexOutOfBoundsException , SecurityException , dll:

Contoh :

Lempar pengecualian jika usia di abwah 18 (cetak “Akses ditolak”). Jika usian 18 tahun atau lebih cetak “Akses diberika”



Outputan :

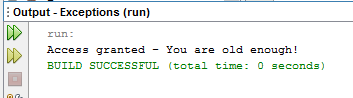


Jika usia 20 tahun, saya  tidak akan mendapatkan pengecualian:

Contoh :



Outputan:



**BAB III**

**HASIL PERCOBAAN**

1. **Java Class Attributes**

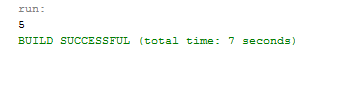
java class attribbutes dibagi beberapa kelas yaitu:

* Accessing Attributes

Contoh :

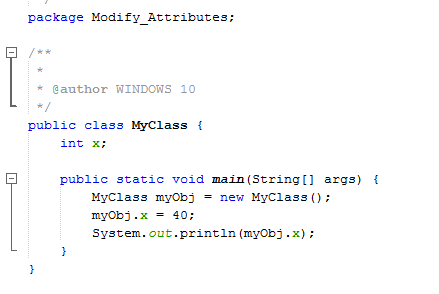


Hasil Output

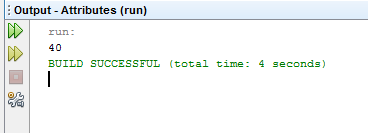


* Modify Attributes

Contoh :

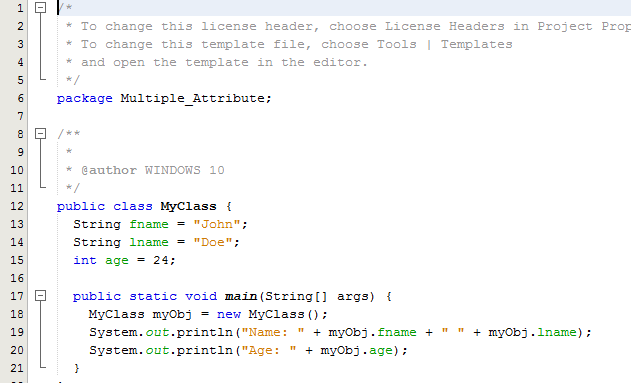


Hasil output

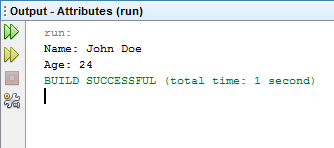


* Multiple Attribute

Contoh :

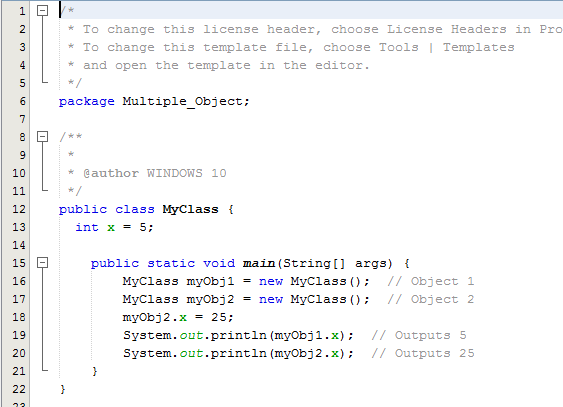


Hasil Output

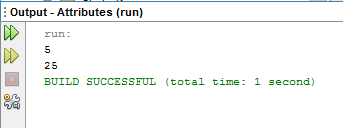


* Multiple Object

Contoh :



Hasil Output

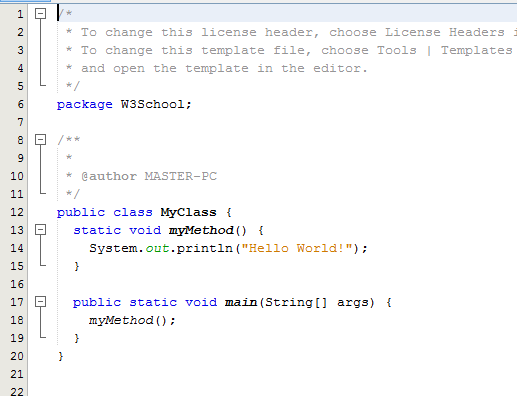


1. **Java Class Methods**

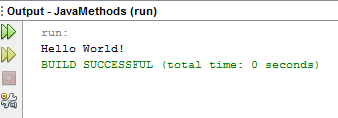
java class methods dibagi 2 class yaitu :

* MyClass

Contoh :

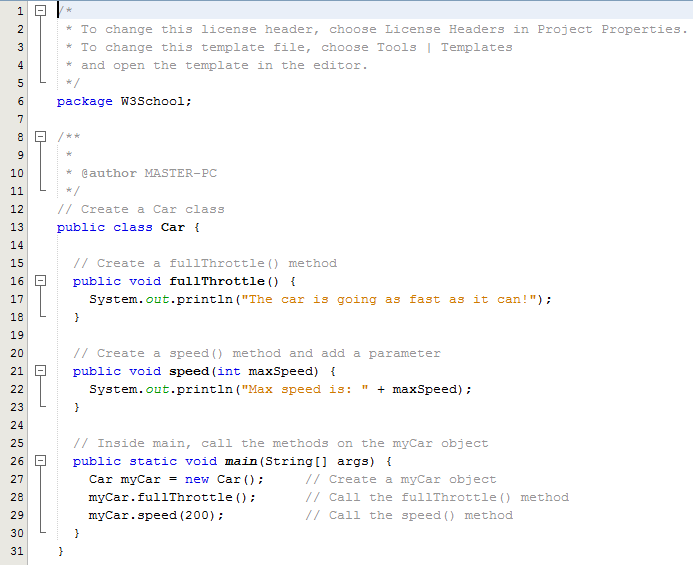


Hasil Output

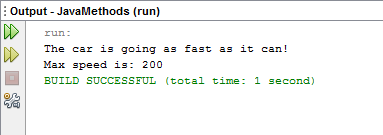


* Car

Contoh :



Hasil Output :

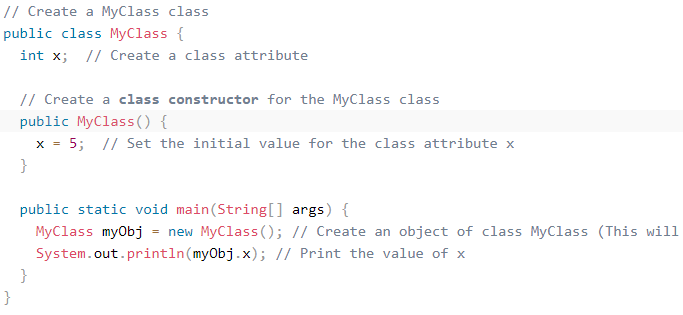


1. **Java Constructors**

Java Constructors dibagi 2 kelas yaitu:

* Java Constructors

Contoh :

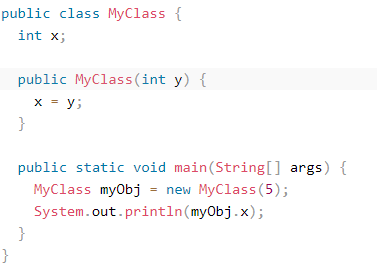


Hasil Output

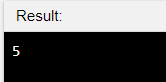


* Parameter Constructors

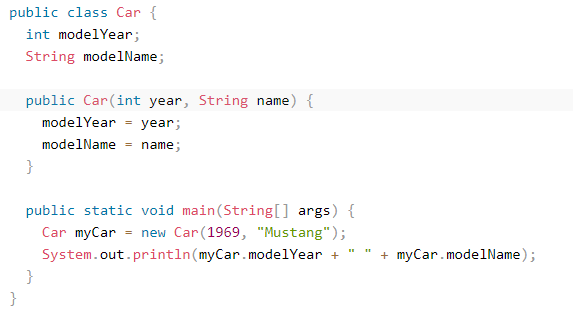
Contoh 1 :



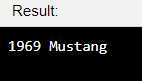
Hasil Output



Contoh 2 :



Hasil Output

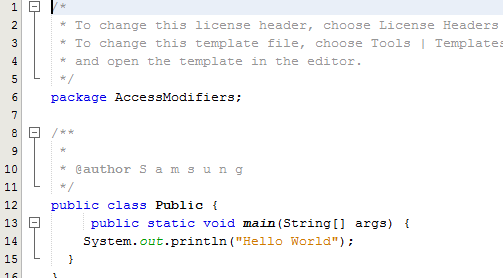


1. **Java Modifiers**

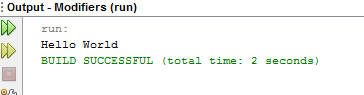
Java Modifiers dibagi menjadi beberapa kelas yaitu :

* Access Modifiers

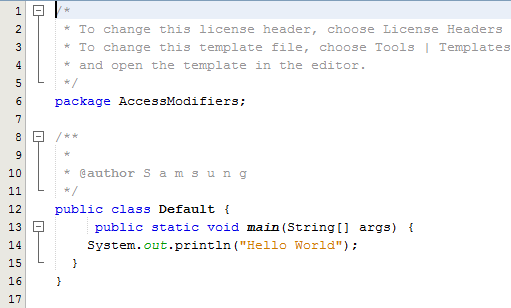
Contoh : public 1



Hasil Output



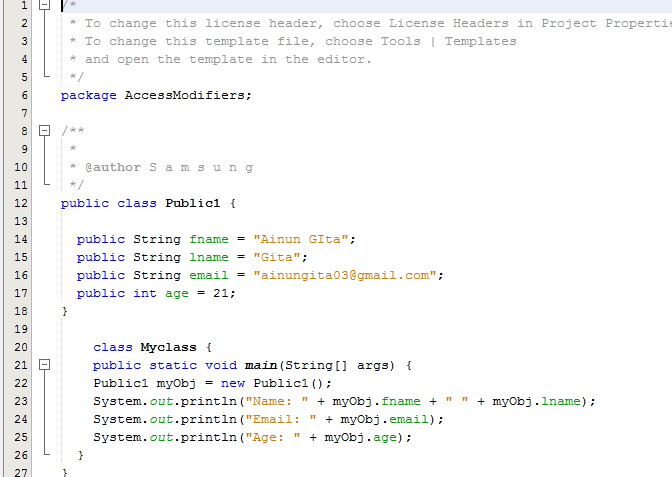
Contoh : default 1



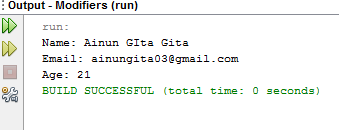
Hasil Output



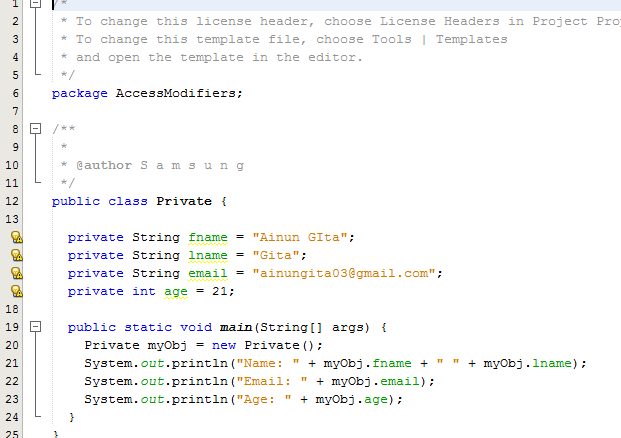
Contoh :public2



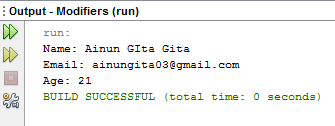
Hasil output



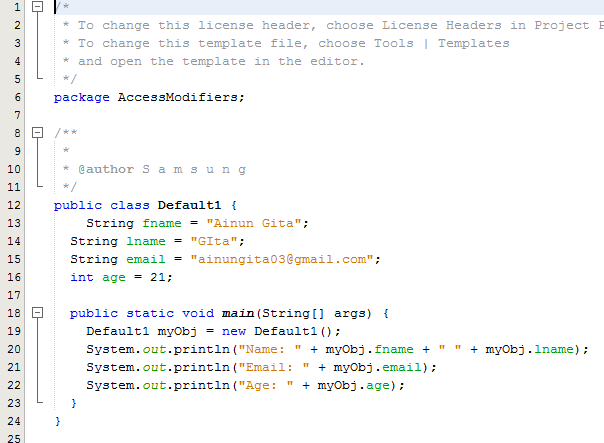
Contoh : private



Hasil output



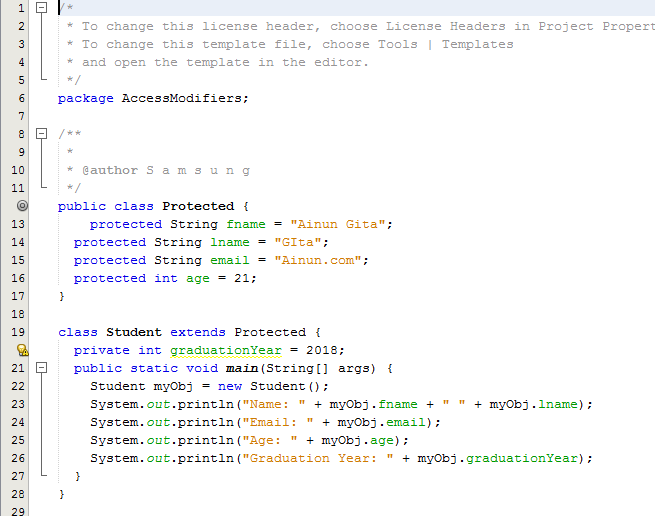
Contoh : default 2



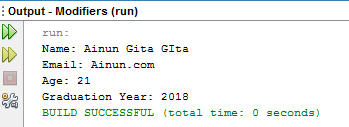
Hasil output



Contoh : protected

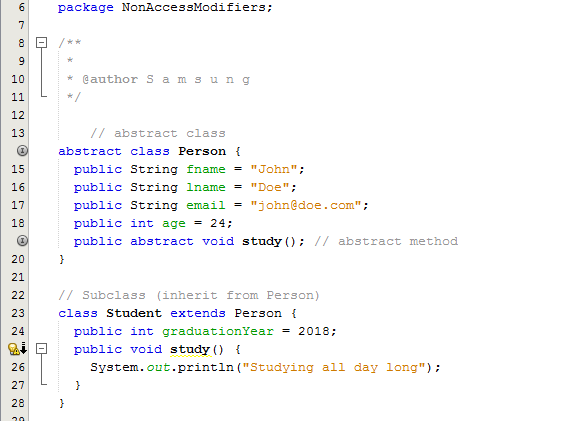


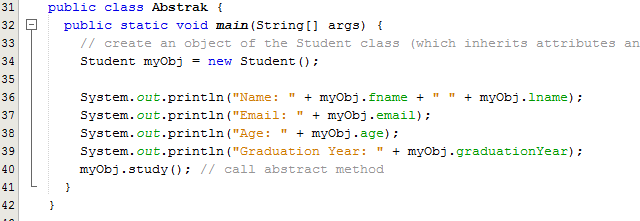
Hasil Output



* Non Access Modifiers

Contoh : Abstrak

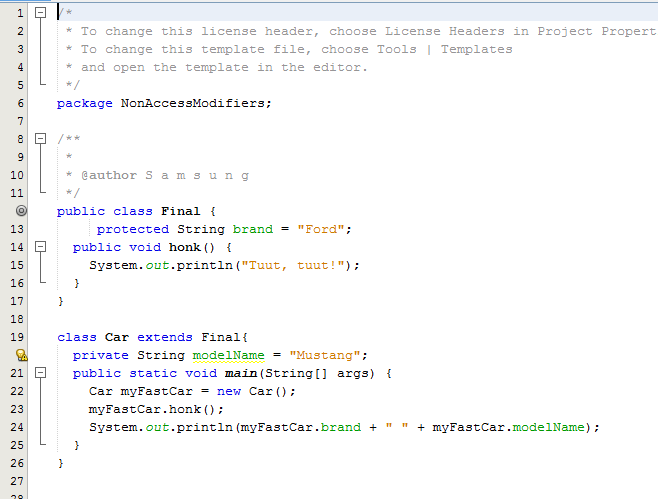




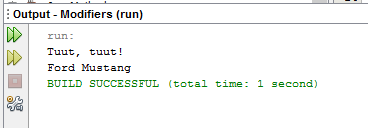
Hasil Output



Contoh : final



Hasil Output

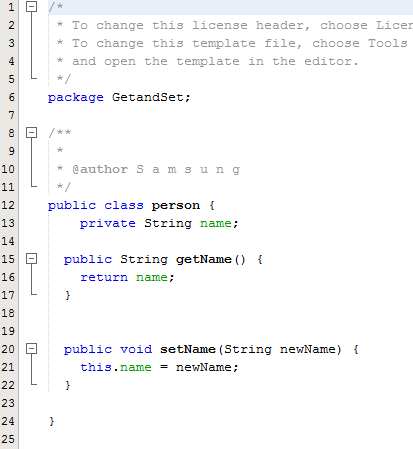


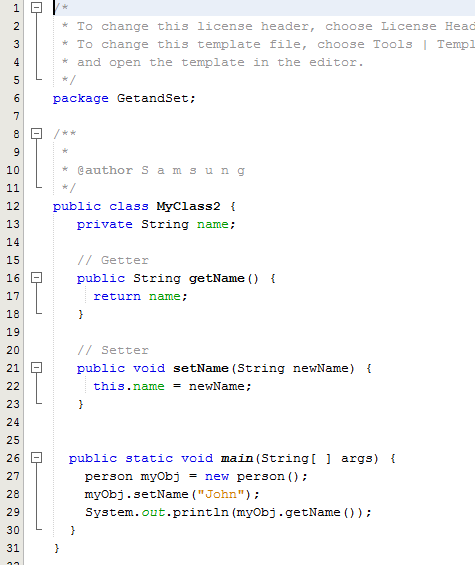
1. **Java Encapsulation**

Java Encapsulation dibagi dua kelas yaitu :

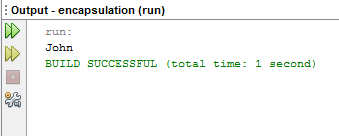
* MyClass

Contoh :





Hasil Output

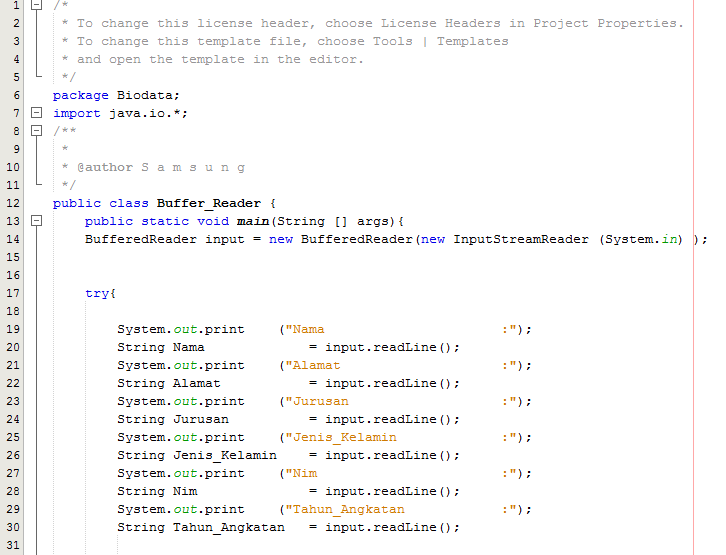


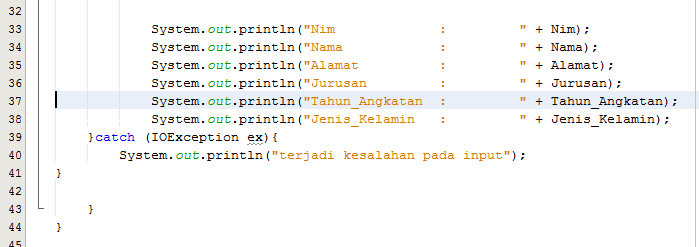
1. **Java Packages/API**

Java Packages/API dibagi menjadi dua kelas yaitu :

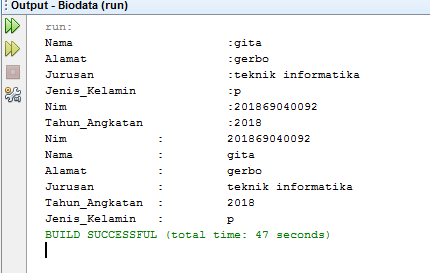
* Buffer Reader

Contoh :



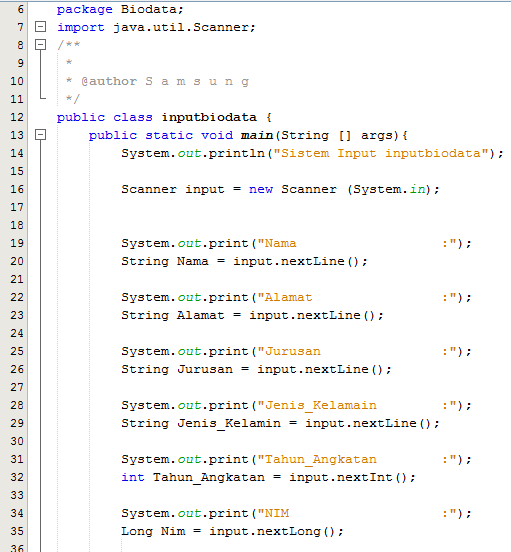


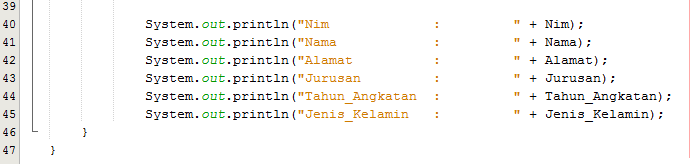
Hasil output



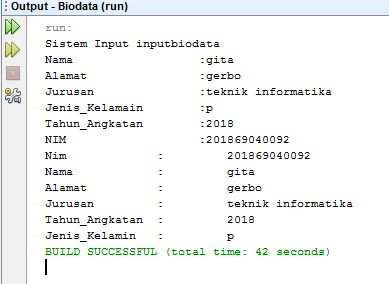
* Input Biodata

Contoh :





Hasil Outputan



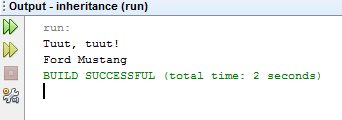
1. **Java Inheritance**

* Vehicle

Contoh :



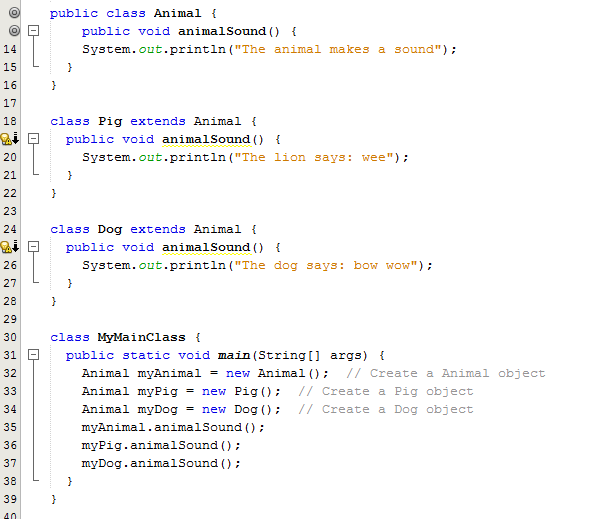
Hasil Output



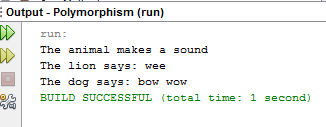
1. **Java Polymorishm**

* Animal

Contoh :



Hasil Output

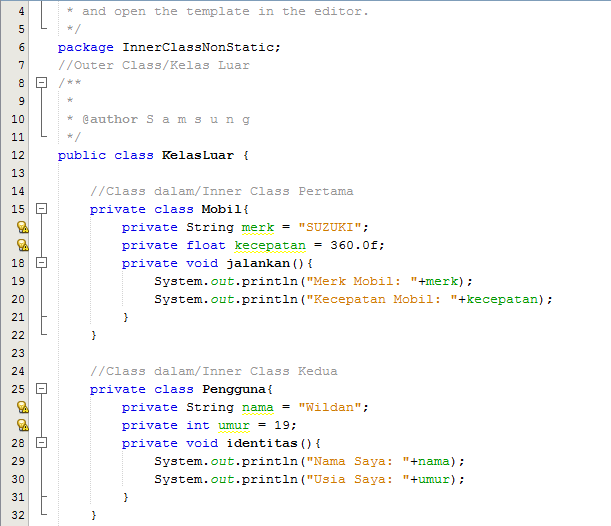


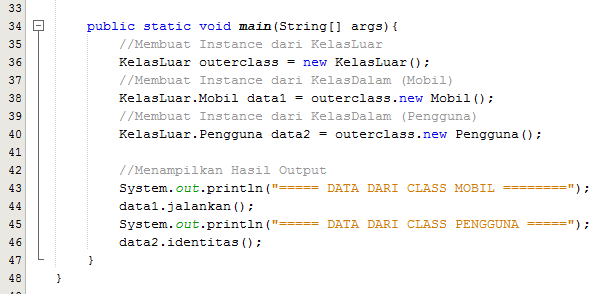
1. **Java Inner Classes**

Java Inner Classes dibagi dua kelas yaitu :

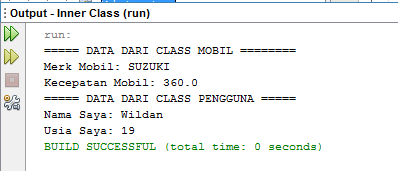
* Inner Class Non Static

Contoh :



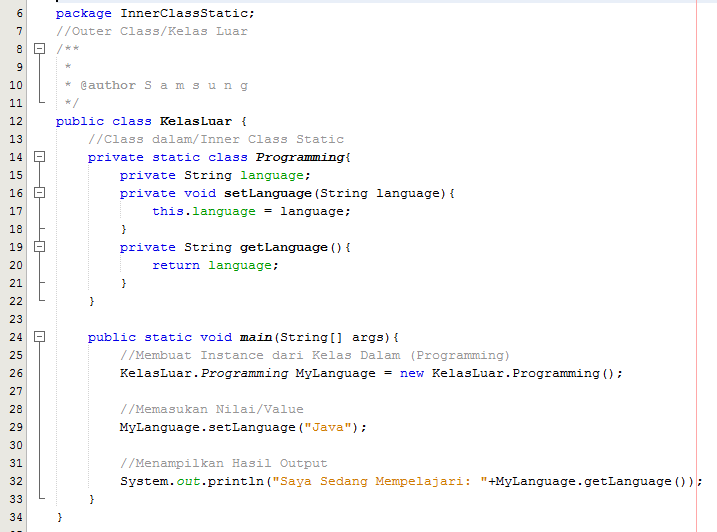


Hasil Output

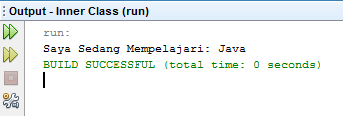


* Inner Class Static

Contoh :



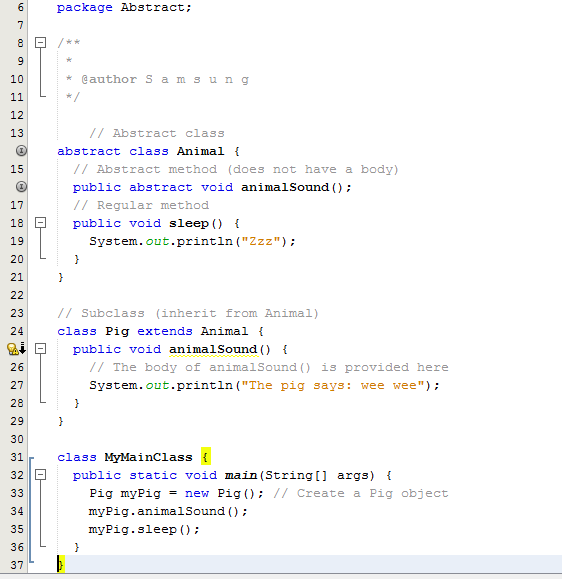
Hasil Output



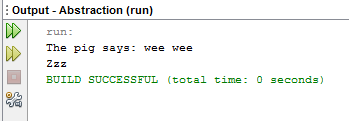
1. **Java Abstraction**

* **Abstract**

Contoh :



Hasil Output

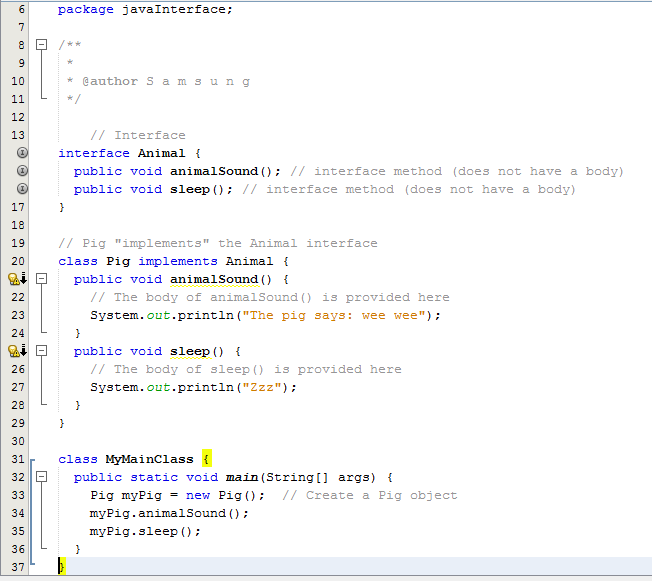


1. **Java Interface**

Java Interface dibagi menjadi dua kelas yaitu :

* Animal

Contoh :

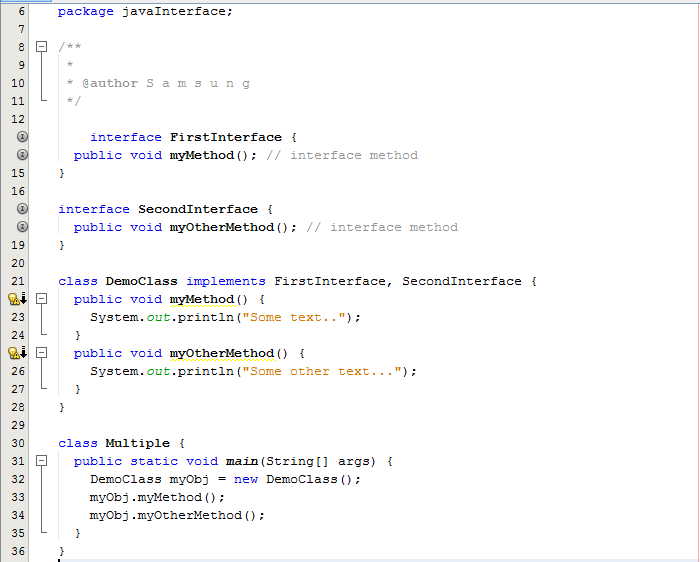


Hasil Output



* Multiple

Contoh :



Hasil Output

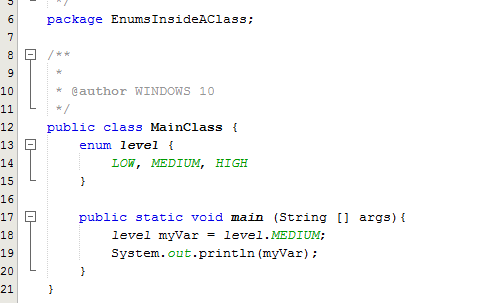


1. **Java Enums**

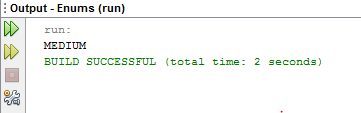
Java Enums dibagi menjadi beberapa kelas yaitu :

* Enum Inside a Class

Contoh :



Hasil Output

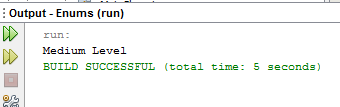


* EnumInASwithStatement

Contoh :



Hasil Output



* Loop Through an Enum

Contoh :



Hasil Output

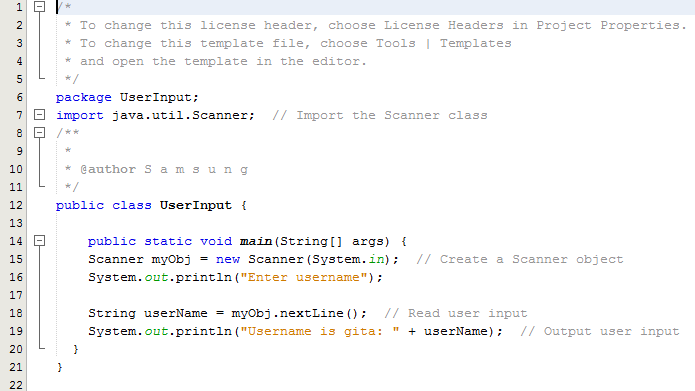


1. **User Input**

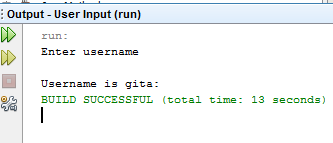
User Input dibagi menjadi dua kelas yaitu :

* Java User Input

Contoh :

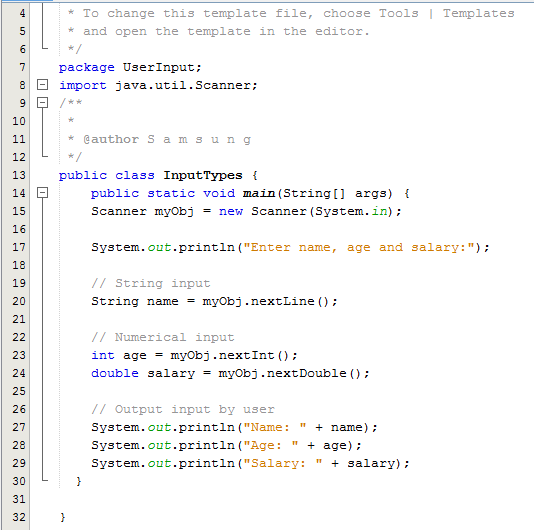


Hasil Output

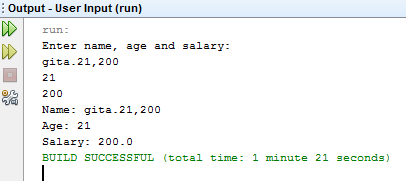


* Input Types

Contoh :



Hasil Output

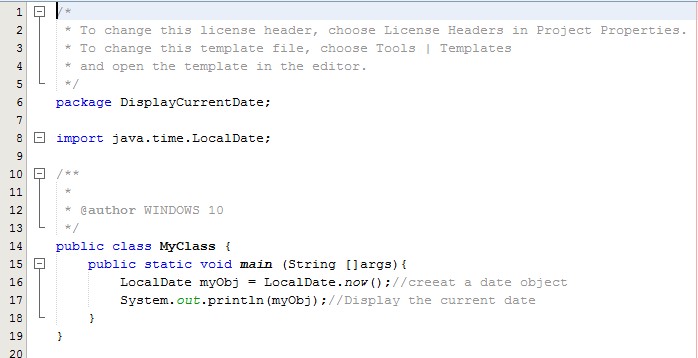


1. **Java Date**

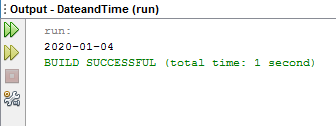
Java Date dibagi menjadi beberapa kelas yaitu :

* Display Current Date

Contoh :

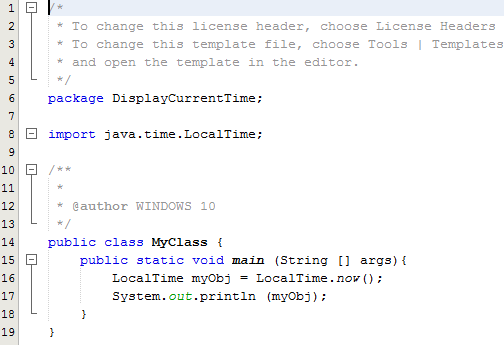


Hasil Output

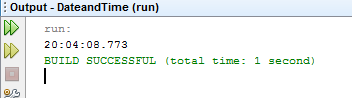


* Display Current Time

Contoh :

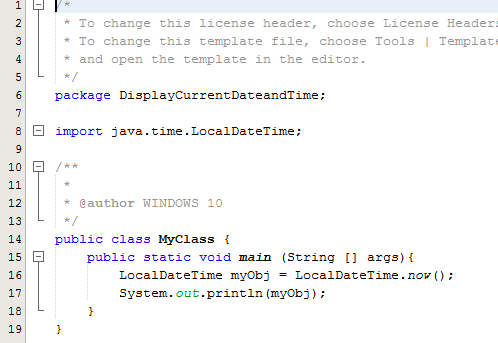


Hasil Output

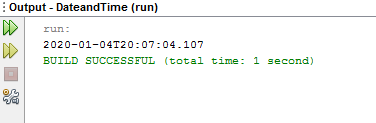


* Display Current Date and Time

Contoh :

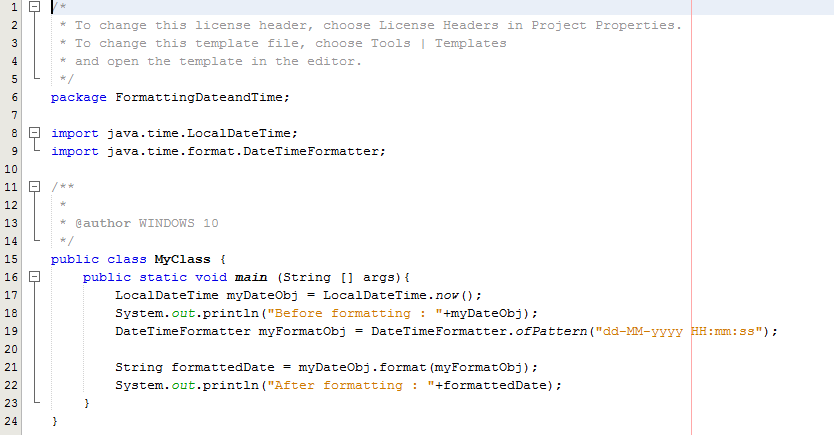


Hasil Output



* Formatting Date and Time

Contoh :



Hasil Output

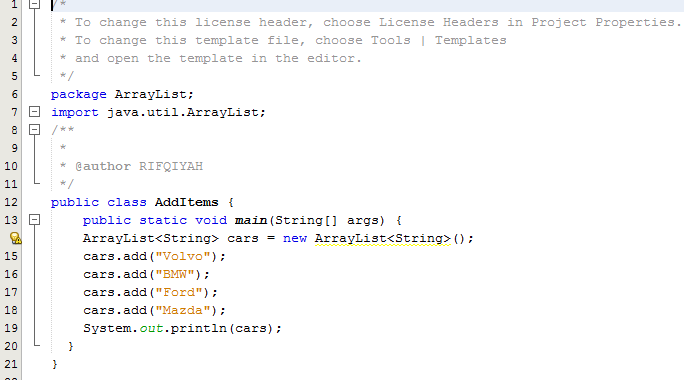


1. **Java ArrayList**

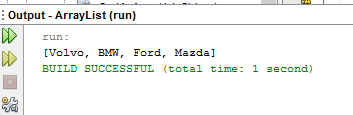
Java ArrayList dibagi menjadi beberapa kelas yaitu :

* Add Items

Contoh :

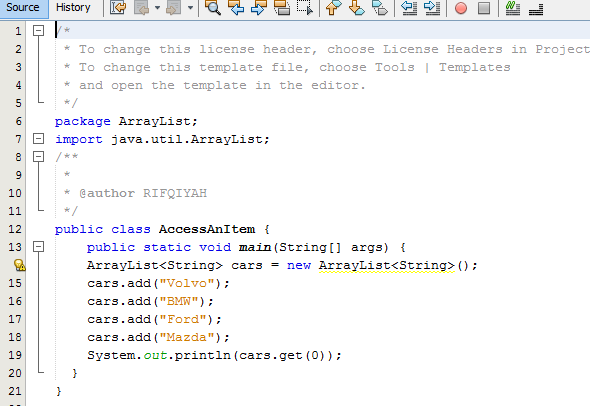


Hasil Output

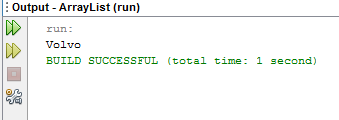


* Access an Item

Contoh :

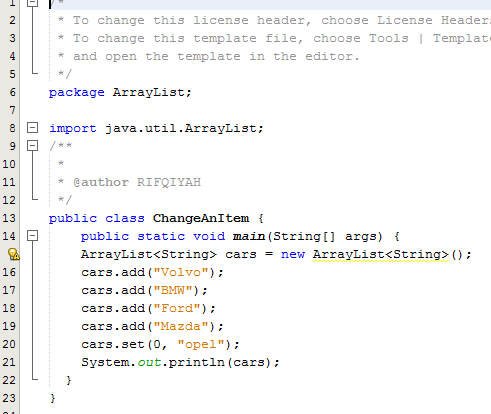


Hasil Output

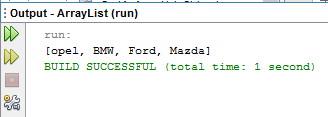


* Change an Item

Contoh :

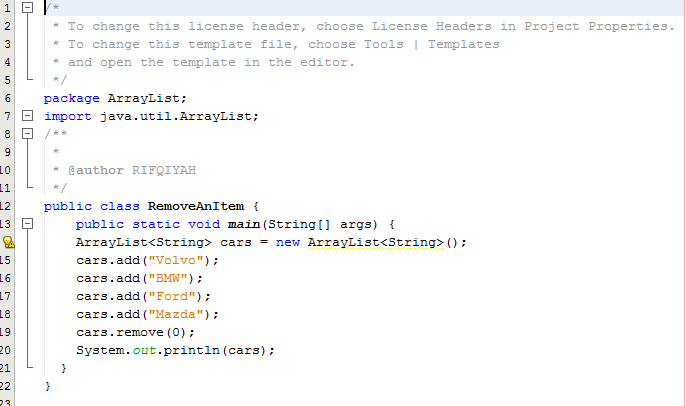


Hasil Output

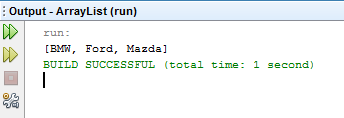


* Remove an Item

Contoh :

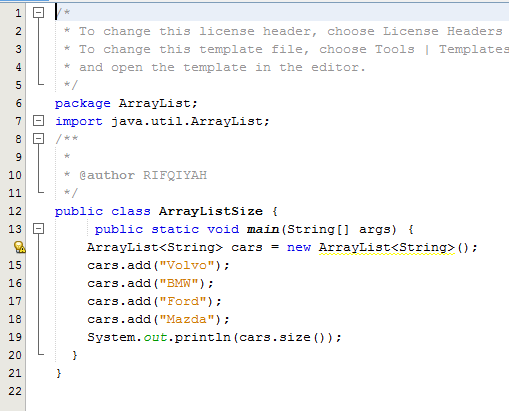


Hasil Output

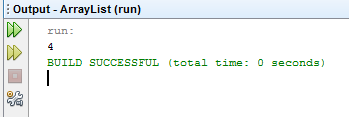


* ArrayList Size

Contoh :

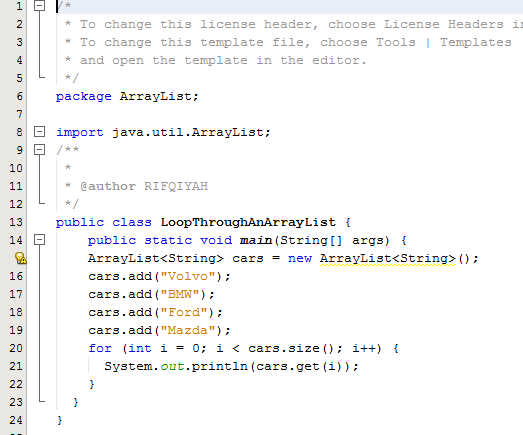


Hasil Output

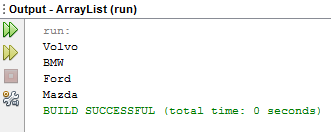


* Loop Through an ArrayList

Contoh :

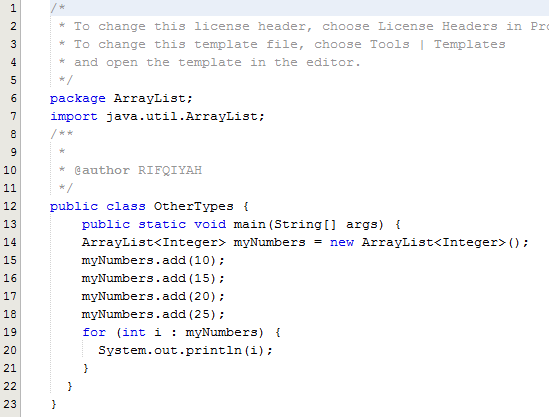


Hasil Output

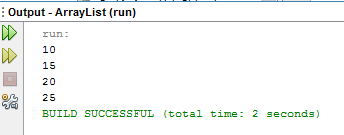


* Other Types

Contoh :

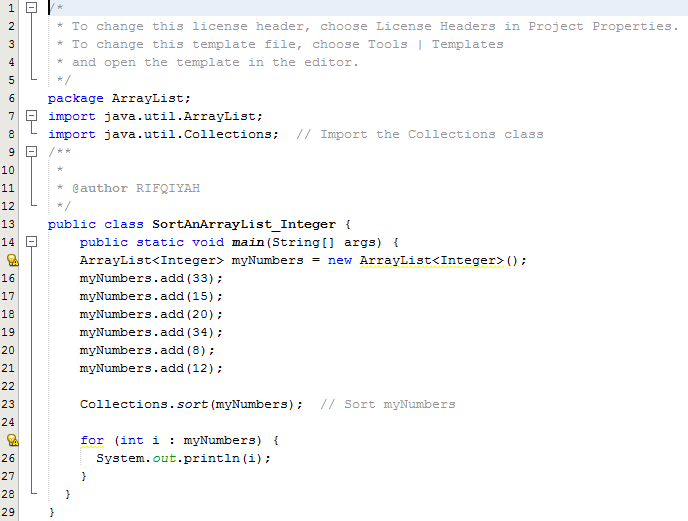


Hasil Output

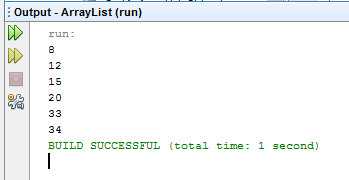


* Sort an ArrayList

Contoh :



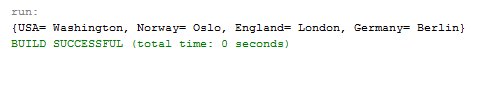
Hasil output



1. **Java HashMap**

* **Menambahkan Item**

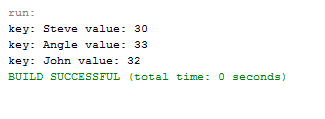
Output dari proses running:



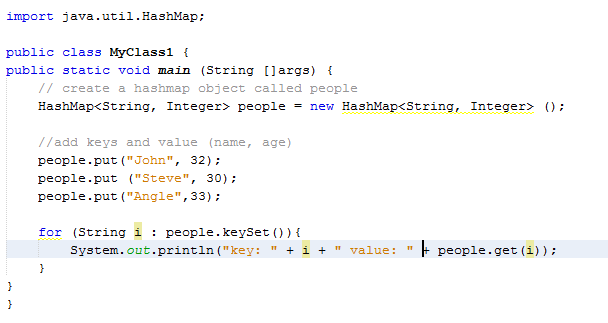
Source code: code program java pada java HashMap untuk menambahkan item dengan kata kunci string dan nilai string



Output dari proses running:



Source code: code program java pada java HashMap untuuk menambahkan item dengan kunci String, dan nilai Integer

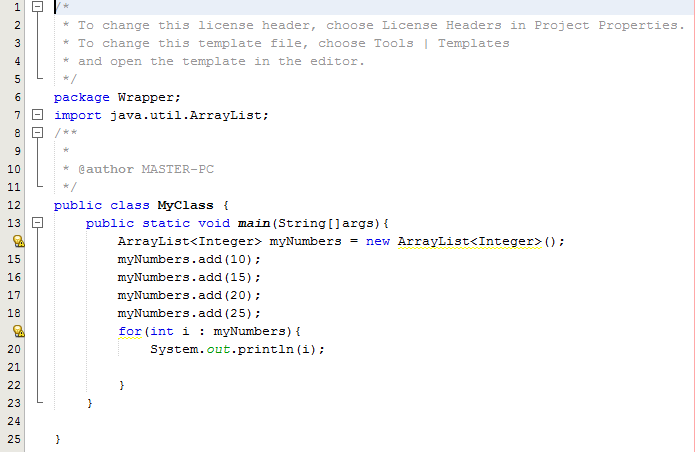


1. **Java Wrapper Classes**

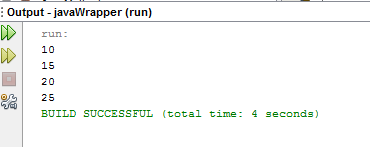
Java Wrapper Classes dibagi menjadi beberapa kelas yaitu:

* Java Wraapper Classes

Contoh :

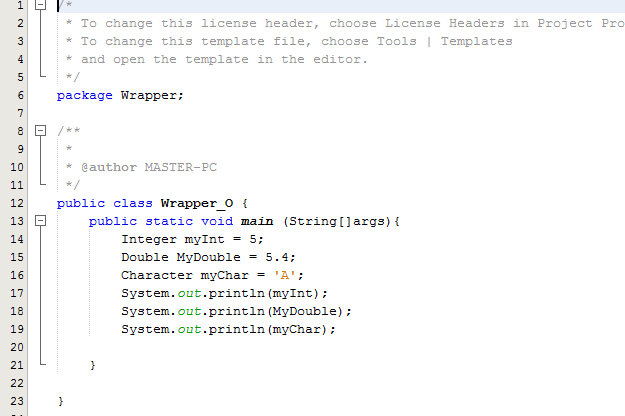


Hasil Output

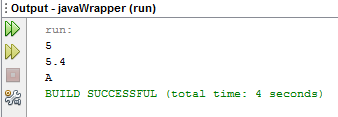


* Wrapper

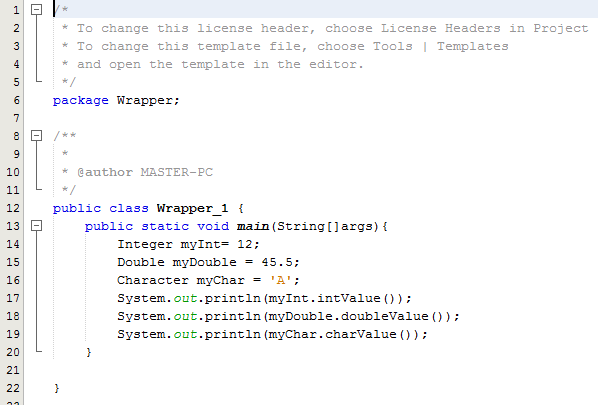
Contoh : 0



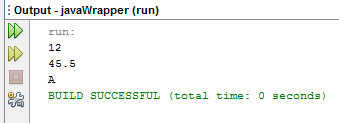
Hasil Output



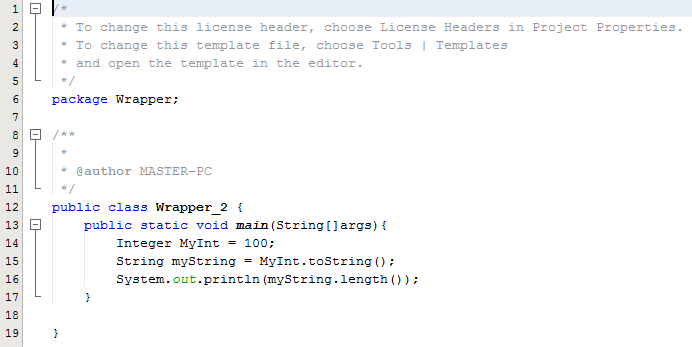
Contoh : 1



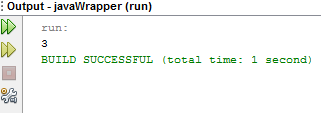
Hasil Output



Contoh : 2



Hasil Output

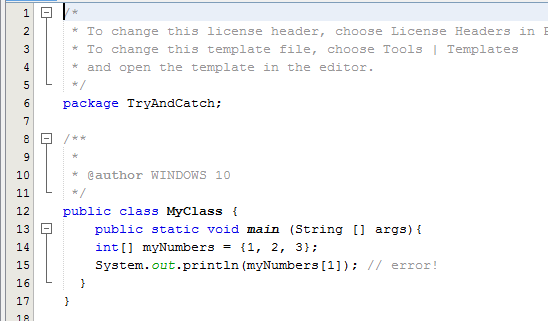


1. **Java Exceptions**

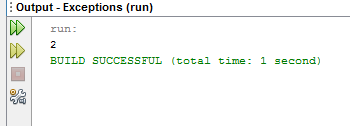
Java Exceptions dibagi menjadi beberapa kelas yaitu :

* Java try and catch

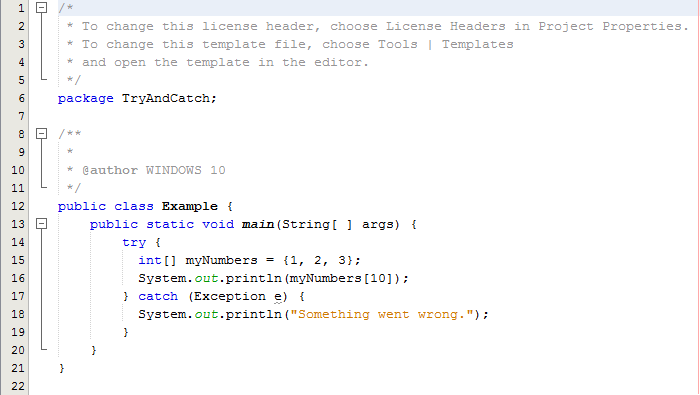
Contoh : 1



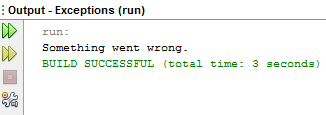
Hasil Output



Contoh : 2

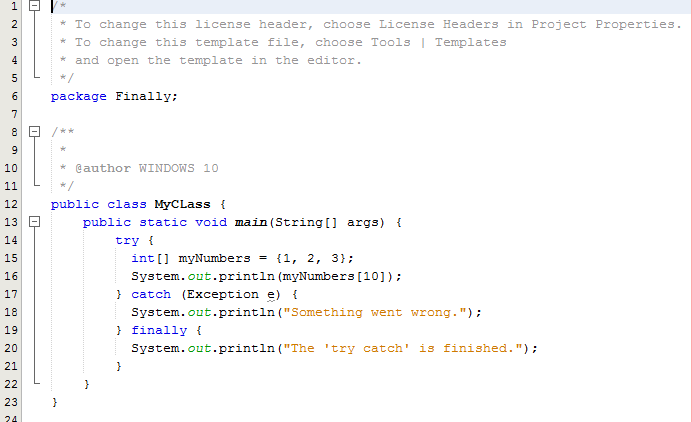


Hasil Output

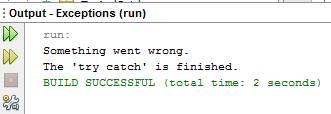


* Finally

Contoh :

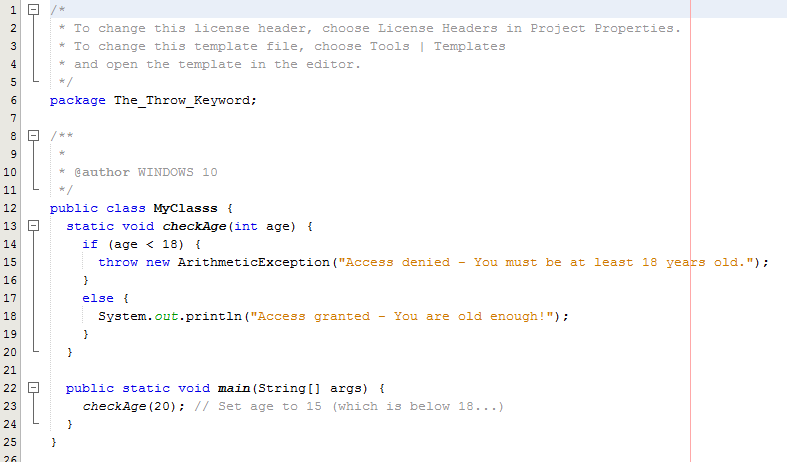


Hasil Output

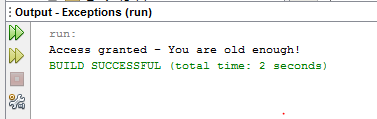


* The throw keyword

Contoh :



Hasil Output



**BAB IV**

**Analisis Study Kasus**

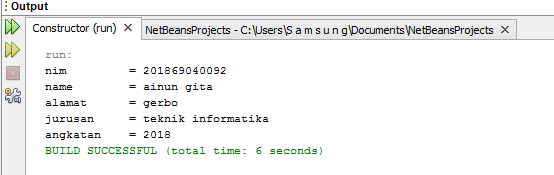
1. **Java Constructor**

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. Didalam construktor ada non parameter dan parameter berikut contohnya

* Non Parameter



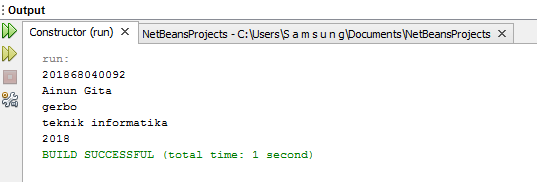
Hasil Output



* Parameter



Hasil Output

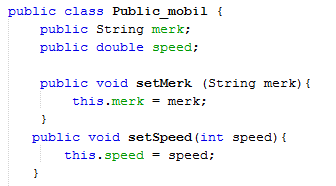


1. **Java Modifiers**

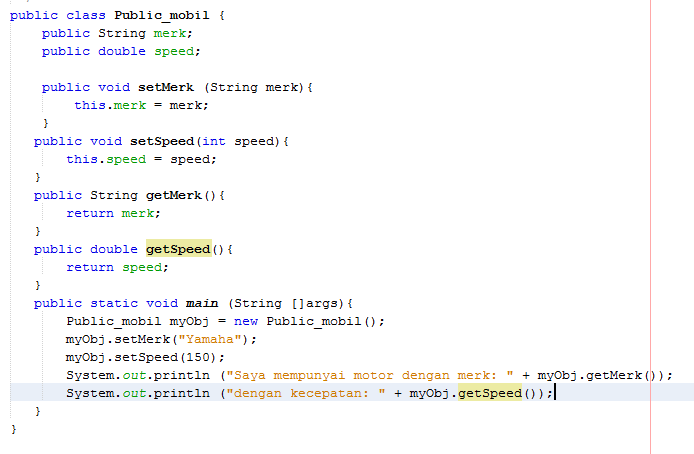
* Public

Java modifier public dapat diakses dari mana saja. Pada studi kasus kali ini, hanya dibuat satu class saja agar memudahkan untuk dipahami. Jika seandainya dibuat 2 class yang berbeda dengan package yang berbeda juga bisa, dikarenakan menggunakan modifiers public.

Contoh source code untuk modifiers public:

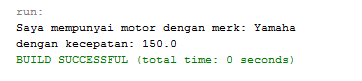


Telah dibuat class bernama public\_mobil dengan memiliki 2 variabel masing-masing data bertipe String dan Integer. Berikut ini adalah source code lengkapnya:



Untuk memanggil atributnya, maka menggunakan method dari getter dan setter, juga telah dibuat object baru dengan nama **myObj.**

Tampilan output:



1. **Encapsulation**

Untuk studi kasus pada java encapsulation,digunakan kode untuk name yang sudah ditetapkan. Telah dibuat dua class dengan nama Name dan Name1. Pada class Name diisikan dengan variabel dan method get and set agar dapat dipanggil dalam class Name1.

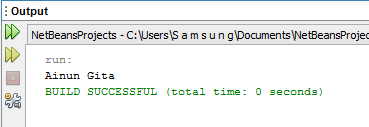
* Contoh 1



* Contoh 2



Hasil Output



1. **Packages /API**

* Class Scanner

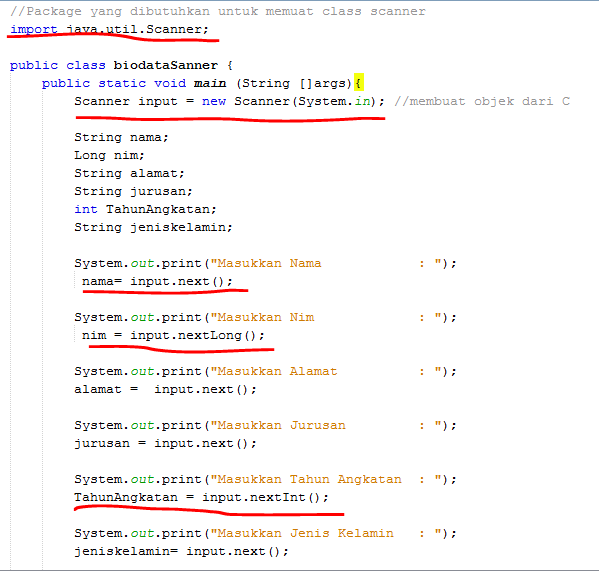
Pada studi kasus java package/API class scanner, akan dibuat contoh menginputkan biodata mahasiswa. Untuk membuat class scanner diperlukan mengimport dulu.



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:

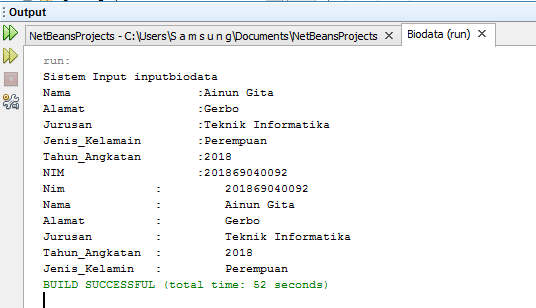




Perlu diketahui tentang code yang diberi garis merah dibwahnya. Untuk class scanner di haruskan untuk mengimport scanner terlebih dahulu, dan juga membuat object scanner yang berguna untuk membuat object. Untuk menginputkan data pada class scanner terbagi menjadi beberapa tipe data, yaitu untuk tipe 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();**

Tampilan output

Hasil Output



* **Buffer Reader**

BufferedReaderadalah 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;**

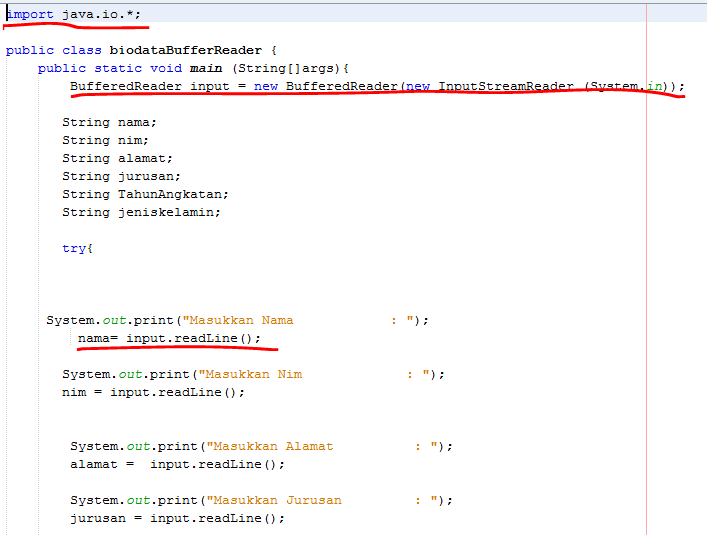
atau untuk lebih mempersingkat pengkodean dapat menggunakan langsung dari clas io, karena dapat memanggil 3 sekaligus class pada java paket io. Untuk membuat java package/API dalam buffer reader diperlukan untuk mengimport terlebih dahulu pada package.

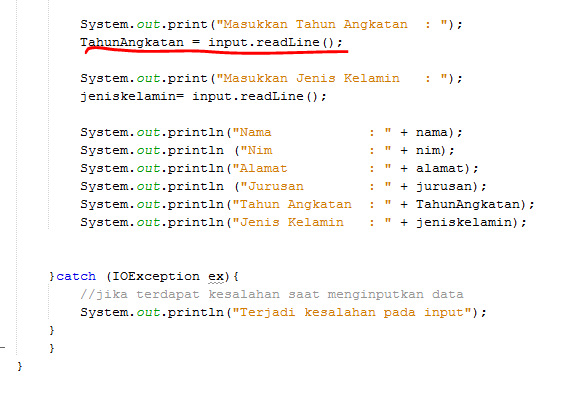


Dan membuat object baru pada class agar dapat memuat object yang nantinya akan di inputkkan.



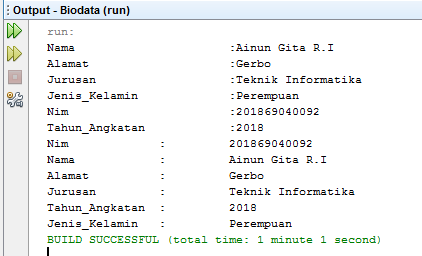
Untuk lebih jelasnya mengetahui perbedaan java package/API pada class scanner, buffer reader, console, dan JOption Pane. Perhatikan source code yan digaris bawahi.





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 Output



1. **Inheritance**

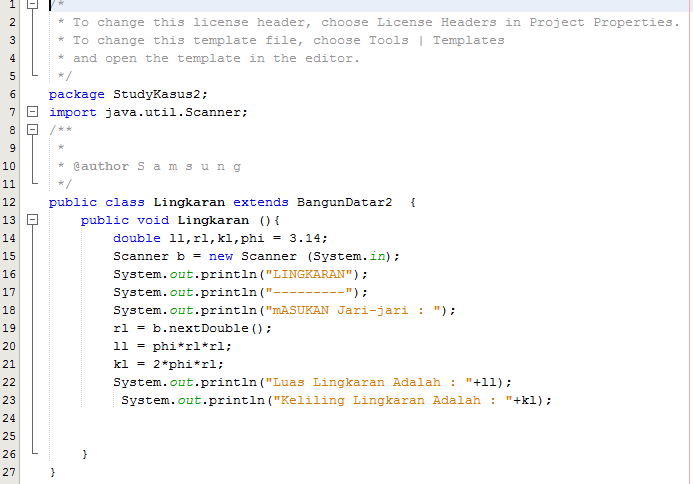
Program java yang dibuat untuk studi kasus java inheritance ini adalah perhitungan luas bangun datar persegi,segitiga, lingkaran, dan persegi panjang dengan menginputkan data sendiri, maka dari itu harus mengimport class scanner. Buat sebuah project dalam satu package bernama **inheritance** dan class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

File: inheritance/Lingkaran.java

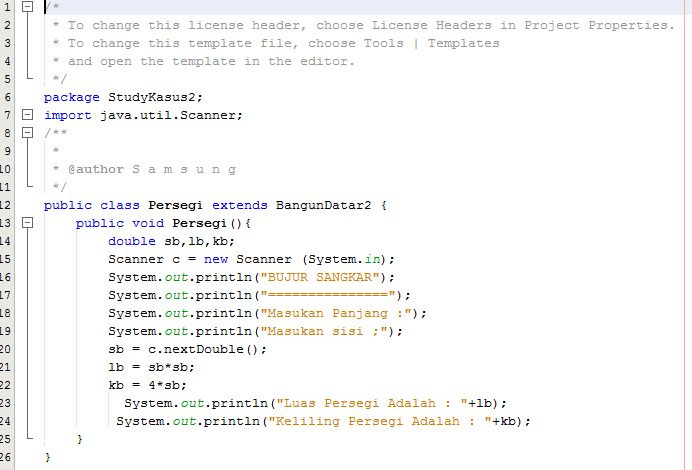
* Java Bangun Datar



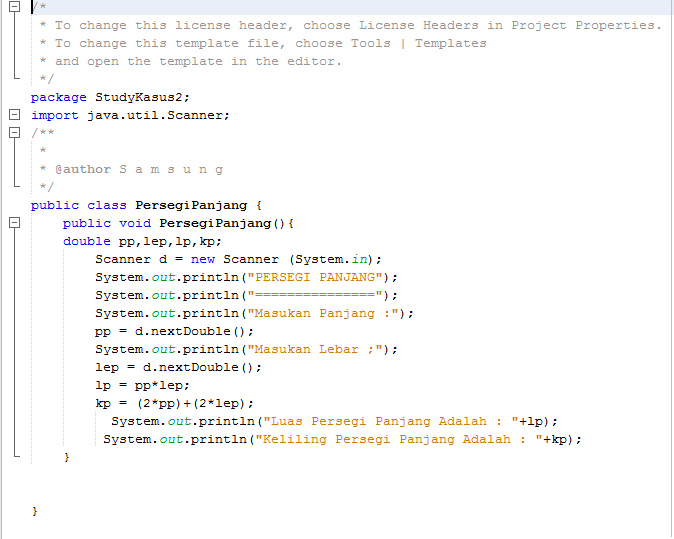
* Java Lingkaran



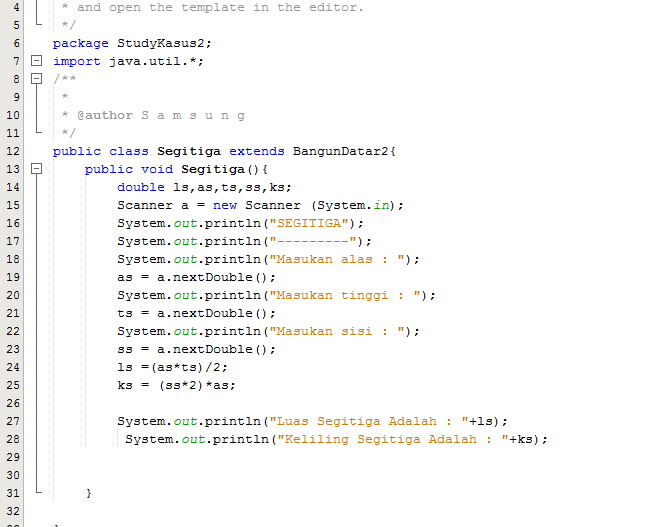
* Java Persegi



* Java Persegi Panjang



* Java Segitiga



Hasil Output

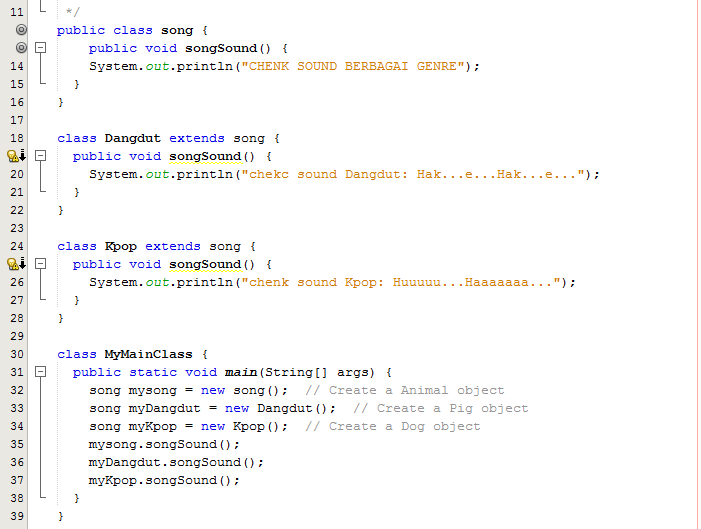


Catatan untuk hasil Output saya contohkan 1 karena hasil Outputan semua dari masing masing kelas sama aja tinggal pilih menu apa yang kita cari contoh saya pilih nomer 2 yaitu kelas segitiga maka akan muncul perintah masukan sisi dan tinggi lalu tekan enter maka akan muncul hasilnya begitu masing masing class semuanya sama seperti itu.

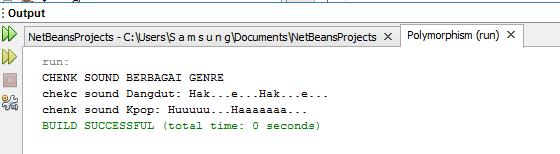
1. **Polymorphism**

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

* Song



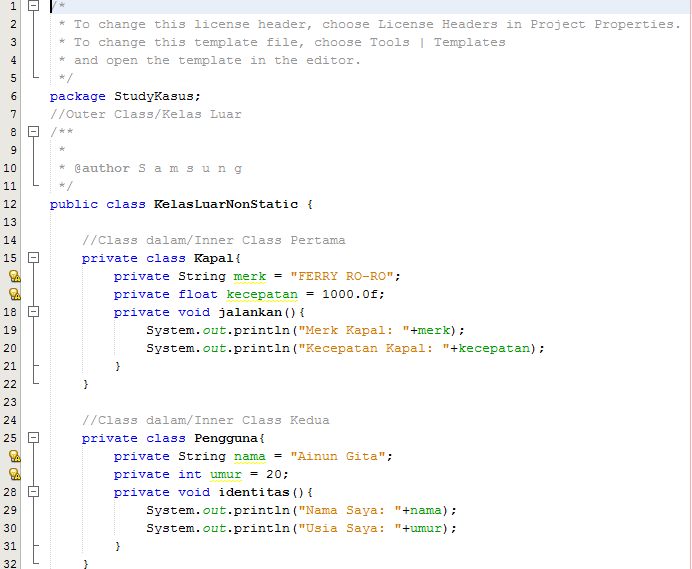
Hasil Output



1. **Java Inner Classes**

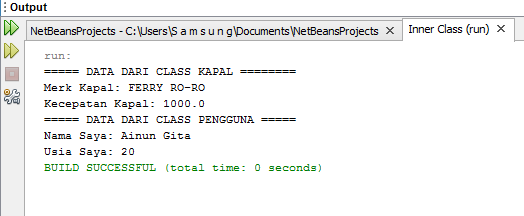
* Kelas Luar Non Static

Untuk mengakses variabel atau method pada class luar, perlu membuat instance/objek class luar di dalam inner class. Contoh seperti saya kasih Pada simbol tanda panah biru



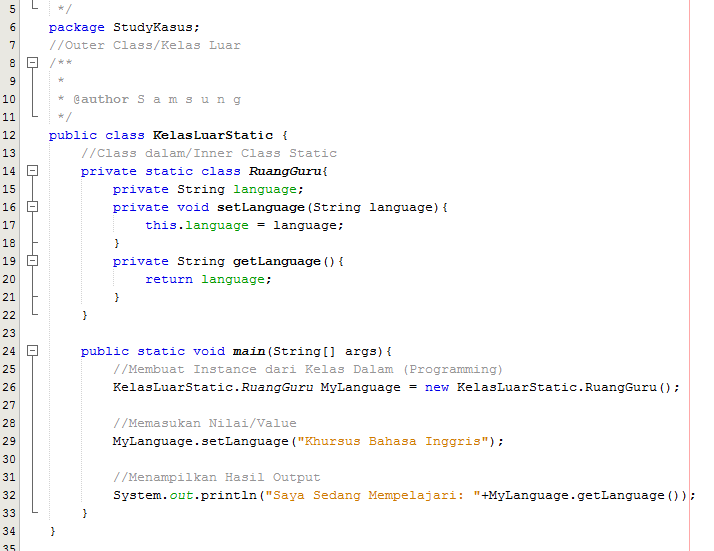


Hasil Output

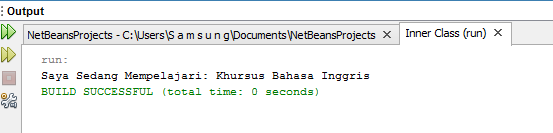


* Kelas Luar Static

Untuk mengubah sebuah Inner Class menjadi Static, hanya perlu menambahkan kata kunci static di belakang class. Contoh seperti simbol tanda panah biri



Hasil Output



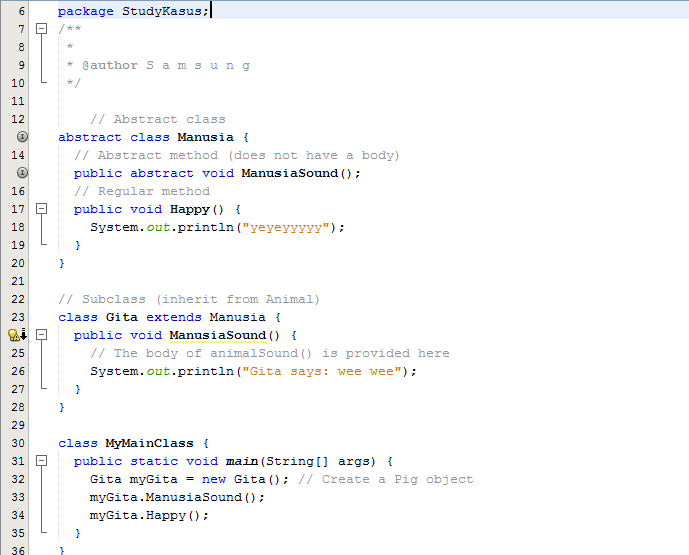
1. **Abstraction**

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

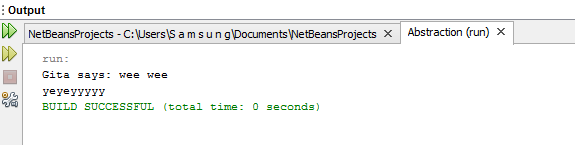


Pada abstract class tidak memiliki body atau method didalamnya.

Berikut adalah code lengkapnya:

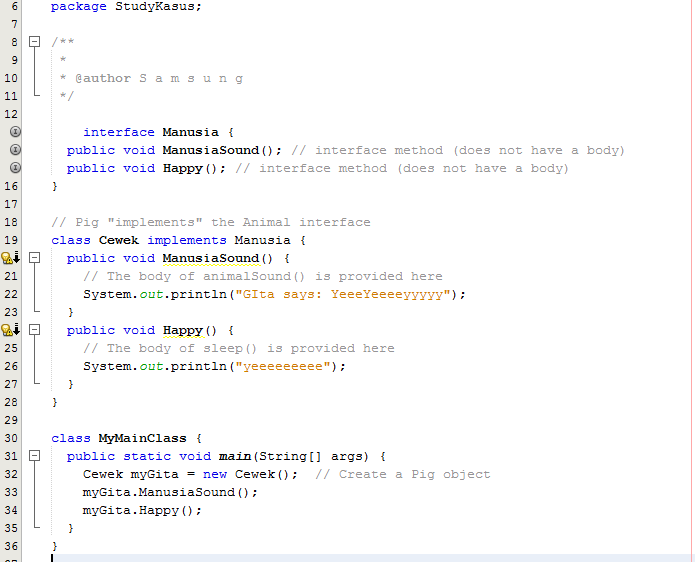


Hasil Output



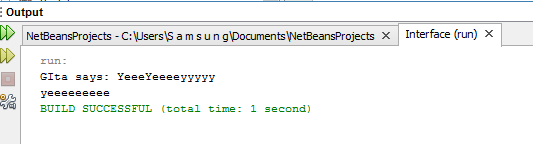
1. **Interface**

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



Class MyMainClass 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

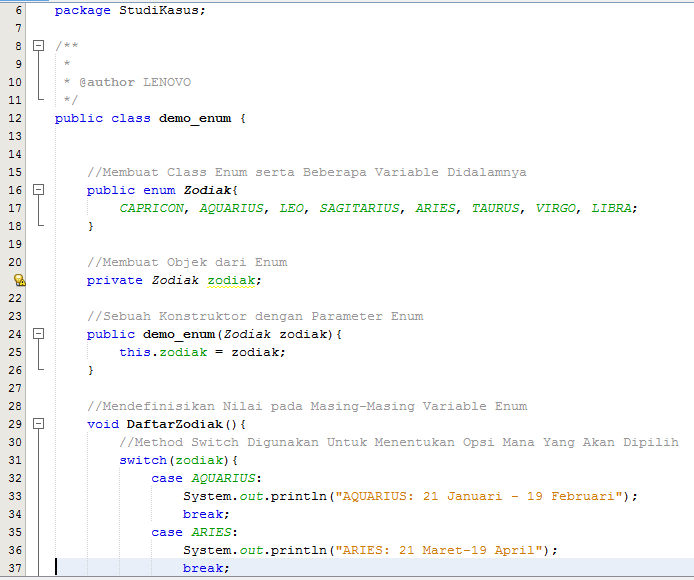


1. **Enums**

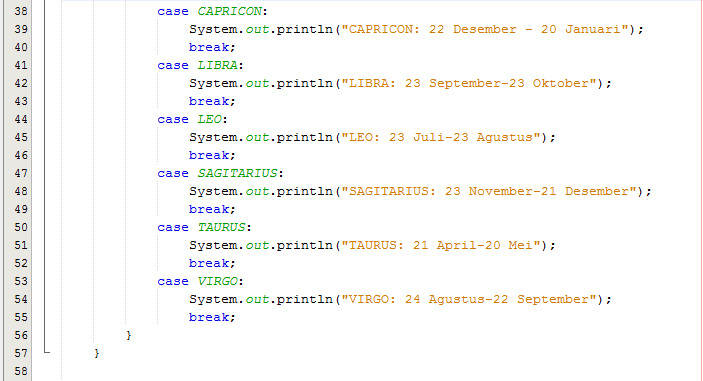
Enum merupakan salah satu class yang dapat digunakan untuk memuat kumpulan variabel yang telah didefinisikan menjadi tipe data konstanta. Setiap objek baru yang didefinisikan berdasarkan tipe data enum, objek tersebut dapat mengakses atribut atau method (Enum) lalu mengaplikasikan nilai atau value yang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

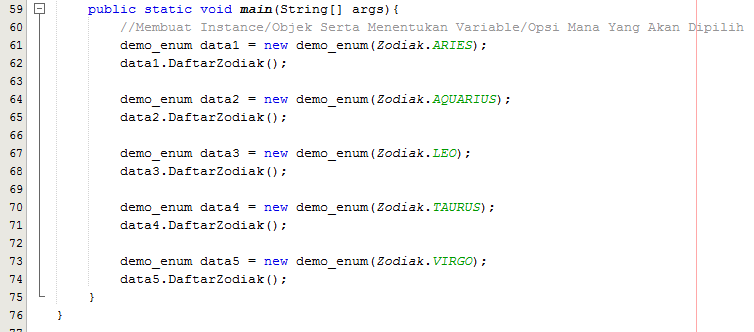
Pada baris code, dibuat variabel refernsi zodiac, lalu membuat constructor milik Main Class.



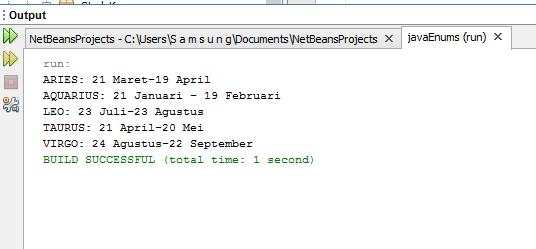
Untuk menentukan nilai output, berdasarkan variabel yang dideklarasikan pada enum zodiac, disini menggunakan switch case yang didefinisikan di dalam method yang bernama DaftarZodiak



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



Hasil Output



1. **User Input (Scanner)**

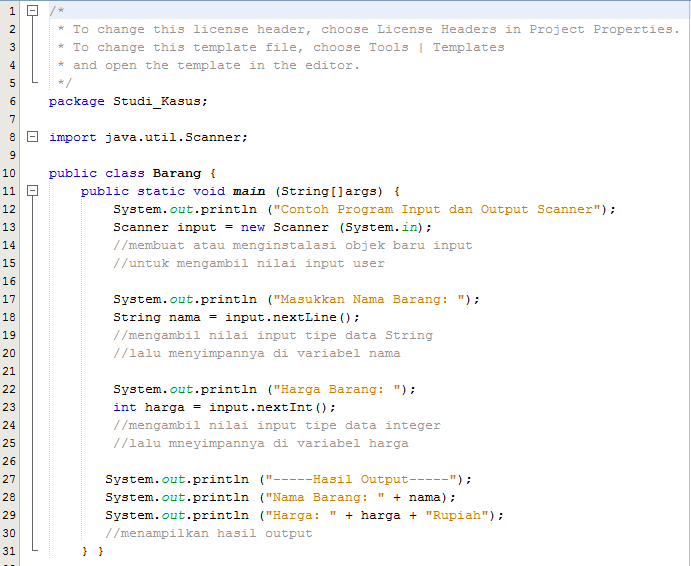
Pada studi kasus java user input, pertama akan dibuat atau menginstalasi objek baru kedalam class scanner yaitu input.



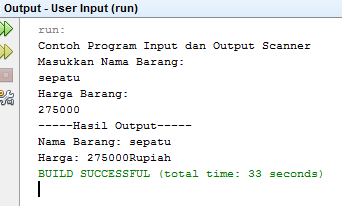
Disini, digunakan system.in agar user atau pengguna dapat menginputkan nilai lewat keyboard. Pada code in, merupakan objek dari class InputStream yang didefinisikan pada class System.

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



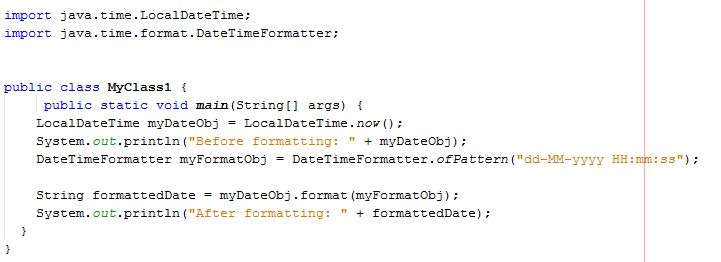
Hasil Output



1. **Date**

* 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.

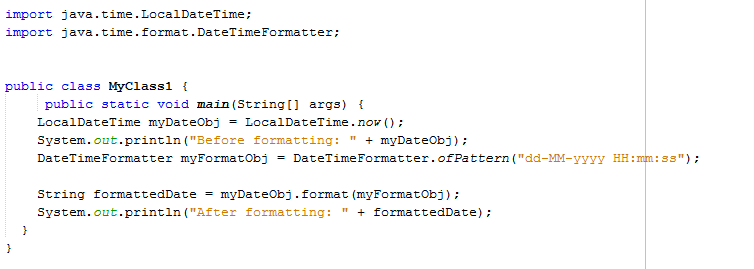


Source code membuat object baru untuk memformat tanggal dan waktu.

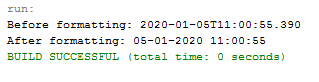


Source code untuk memanggil object baru yang telah di format.

Jadi, source codelengkapnya yaitu:



Hasil output:



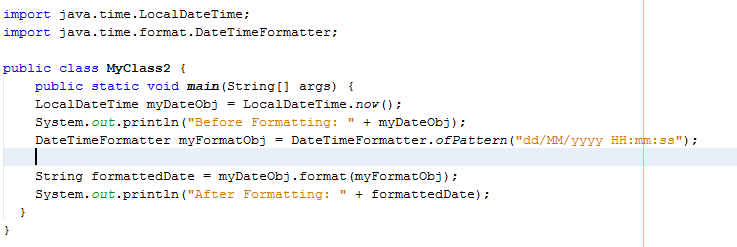
* Tanggal: dd/MM/yyyy

Waktu: HH:mm:ss

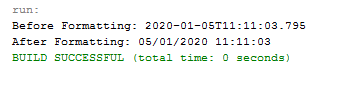
Source code untuk menampilkan format tanggal dan waktu saat ini hampir sama dengan sebelumnya,namun ada perbedaan sedikit pada source code untuk format tanggal dan waktunya.



Source code lengkapnya:



Hasil output:



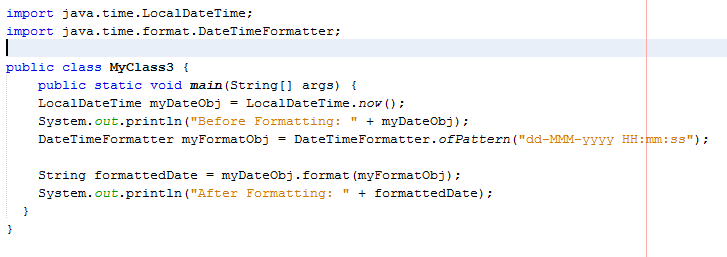
* Tanggal: dd-MMM-yyyy

Waktu: HH:mm:ss

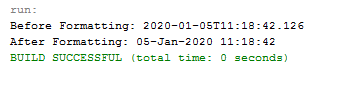
Source code untuk memformat tanggal dan waktu



Source code lengkapnya yaitu:



Hasil output:



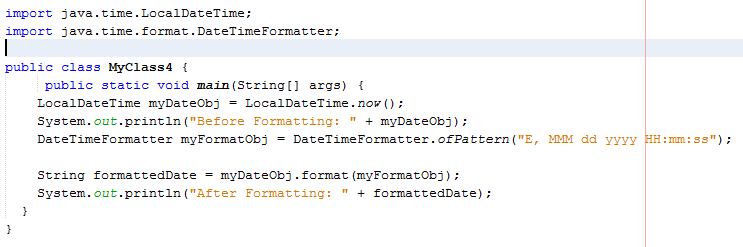
* Tanggal: E, MMM dd yyyy

Waktu: HH:mm:ss

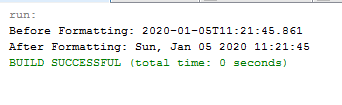
Source code untuk memformat tanggal dan waktu sesuai yang dituliskan.



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



Hasil output:



1. **Java ArrayList**

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

Untuk menggunakan ArrayList, harus mengimpor terlebih dahulu.



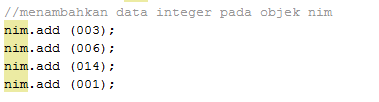
Kemudian, membuat object sesuai dengan tipe data.



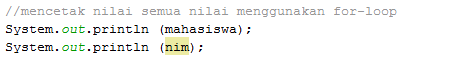
Source code diatas untuk membuat object baru dengan tipe data String. Gambar dibawah ini merupakan source code untuk membuat object dengan tipe data Integer.



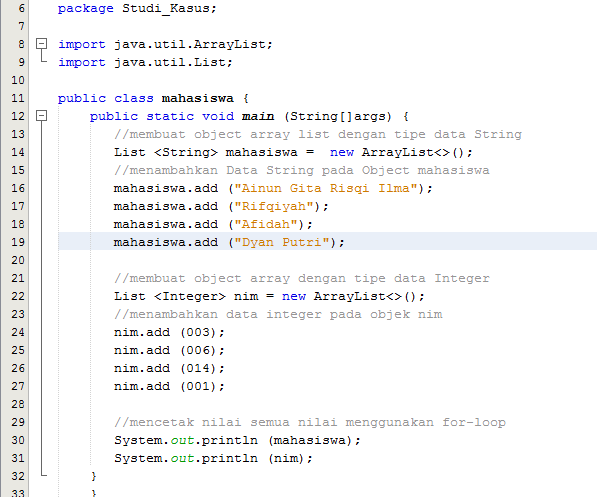
Source code untuk menambahkan data pada tipe data String menggunakan method put().



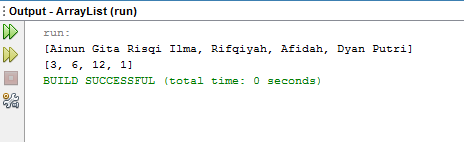
Source code untuk menambahkan data pada tipe data Integer menggunakan method put(). Setelah menambahkan data pada masing-masing object, kemudian panggil semua object menggunakan method for-loop.



Source code lengkapnya yaitu:



Hasil output:



1. **Java HashMap**

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

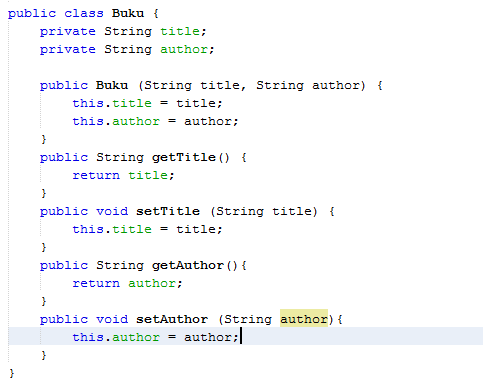


Unntuk menggunakan HashMap, diharuskan untuk membuat objeknya terlebih dahulu. Object dari hashMap dapat dibuat dengan kata kunci **new.**

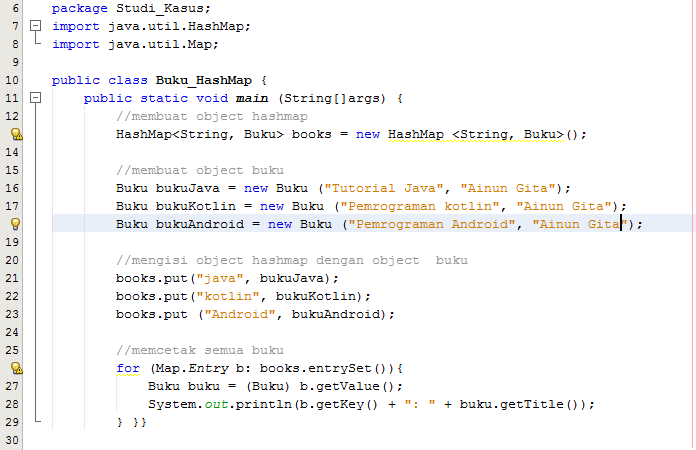


Pada source code diatas, telah dibuat object HashMap bernama books. Object ini dapat digunakan untuk menyimpan koleksi data. Untuk studi kasus java HashMap,dibuat dua class, yaitu Buku.java dan Buku\_HashMap.java dalam satu package.

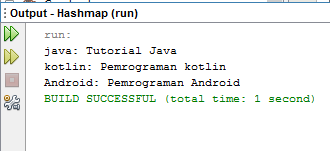
Berikut isi dari class Buku.java



Berikut isi class dari Buku\_HashMap.java



Hasil Output



1. **Java Wrapper**

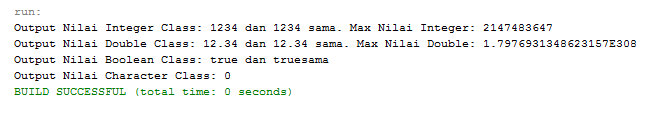
Source code untuk program java wrapper class yaitu:



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.

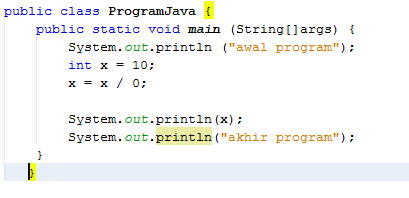
Hasil output:



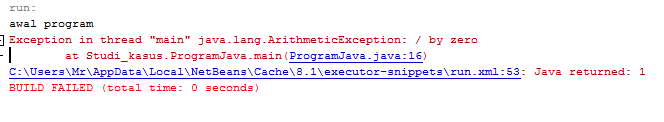
1. **Exceptions**

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.

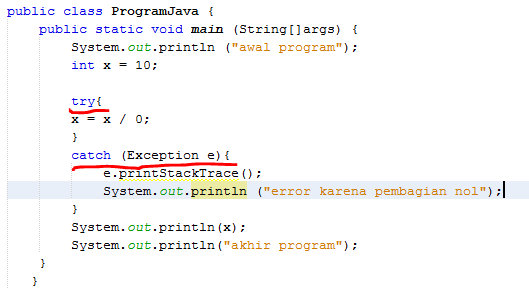
Untuk java exception, dibuat sebuah studi kasus yang mengandung error ketika dijalankan. Berikut adalah source code nya:



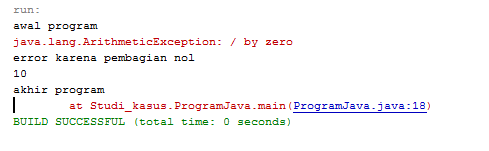
Ketika dijalankan maka akan mendapat hasil output sebagai berikut:



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

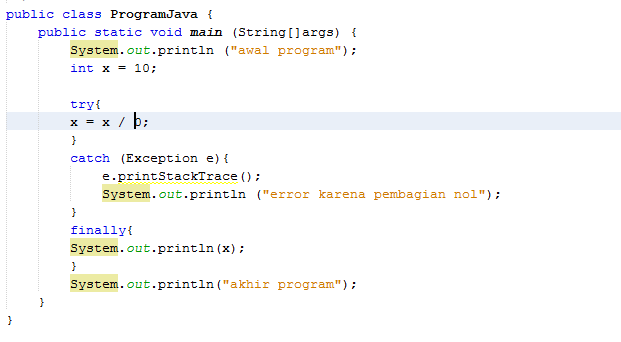


Hasil output:

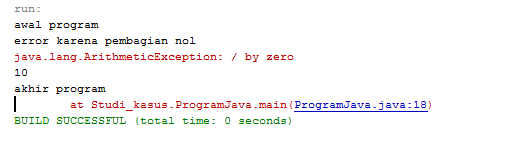


Ketika program di eksekusi kembali maka program akan dijalankan bagian akhir program walaupun terjadi error di tengah-tengah eksekusi.

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 di kembangkan lebih lanjut sebagai pedoman dalam mempelajari sebuah pemrograman objek dalam java. Tidak hanya mempelajarinya sebatas teori tetapi juga bisa mempraktikkannya. Hal ini di himbau agar perkembangan dalam memajukan Pendidikan teknologi di Indonesia lebih maju lagi.

**DAFTAR PUSTAKA**

W3Schools. 2019. *Java*.

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