**LAPORAN PRAKTIKUM**

**“Pemrograman Berbasis Objek”**



Laporan Untuk Melengkapi Tugas Terstruktur Mata Kuliah Pemrograman Berorientasi Object

Dosen Pengajar :

**Bapak Mohammad Imron Rosadi,M.Kom**

Disusun Oleh :

**Dafina Agustina (201869040014)**

TEKNIK INFORMATIKA-3A

**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.

Pruwosari, 05 desember 2019

Penulis

Dafina Agustina

**Daftar Isi**

**HALAMAN JUDUL i**

**KATA PENGANTAR ii**

**DAFTAR ISI iii**

**BAB I PENDAHULUAN 1**

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

**BAB II LANDASAN TEORI 2**

A. Java OOP 2

B. Java Classes/Objects 2

C. Java Class Attributes 5

D. Java Class Methods 7

E. Java Constructors 12

F. Java Modifiers 14

G. Java Encapsulation 21

H. Java Package/API 22

I. Java Inheritance (Pewarisan) 27

J. Java Polymorphism 28

K. Java Inner Class 30

L. Java Abstraksi 32

M. Java Interface 34

N. Java Enum 36

O. Java User Input (Scanner) 38

P. Java Date 40

Q. Java ArrayList 43

R. Java HashMap 46

S. Java Wrapper Classes 50

T. Java Exceptions 51

**BAB III HASIL PERCOBAAN PRAKTIKUM 55**

1. Java Classes and Object 55

B. Java Class Attributes 56

C. Java Class Methods 58

D. Java Constructor 59

E. Java Modifiers 60

F. Java Encapsulations 62

G. Java Package / API 63

H. Java Inheritance 65

I. Java Polymorphism 65

J. Java Inner Class 66

K. Java Abstraksi 68

L. Java Interface 68

M. Java Enum 69

N. Java User Input 70

O. Java Date 71

P. Java ArrayList 75

Q. Java HashMap 77

R. Java Wrapper 78

S. Java Exceptions 79

**BAB IV STUDI KASUS 80**

1. Java Class and Objects 80
2. Java Class Attributes 81
3. Java Class Methods 82
4. Java Constructors 83
5. Java Modifiers 84
6. Java Encapsulation 87
7. Java Package / API 89
8. Java Inheritance 92
9. Java Polymorphism 97
10. Java Inner Class 98
11. Java Abstraksi 101
12. Java Interface 102
13. Java Enum 104
14. Java User Input (Scanner) 107
15. Java Date 108
16. Java ArrayList 111
17. Java HashMap 114
18. Java Wrapper 115
19. Java Exception 116

**BAB V PENUTUP 118**

1. Kesimpulan 118
2. Saran 118

DAFTAR PUSTAKA 119

**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. **Java OOP**

OOP adalah singkatan dari Object Oriented Programming (Pemrograman Berorientasi Objek).

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

Pemrograman berorientasi objek memiliki beberapa keunggulan dibandingkan pemrograman procedural:

1. OOP lebih cepat dan lebih mudah untuk dieksekusi
2. OOP menyediakan struktur yang jelas untuk program-program tersebut
3. OOP membantu menjaga kode java KERING “Don’t Repeat Yourself”, dan membuat kode lebih mudah untuk mempertahankan, memodifikasi dan men-debug
4. OOP memungkinkan untuk membuat aplikasi yang dapat digunakan kembali secara penuh dengan kode yang kebih sedikit dan waktu pengembangan yang lebih singkat

Secara garis besar, suatu bahasa pemrograman dapat dikatakan sebagai bahasa pemrograman berorientasi objek apabila ia mendukung konsep abtraksi (abstraction), enkapsulasi (encapsulation), polimorfisme (polymorphism), dan pewarisan (inheritance). Selain konsep – konsep ini, ada beberapa konsep fundamental lainnya seperti, kelas, objek dan message.

1. **Java Classes / Objects**

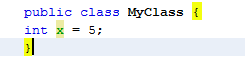
Java adalah bahasa pemrograman berorientasi objek. Segala sesuatu di java dikaitkan dengan kels dari objek, bersama dengan atribut dan metodenya. Contoh , di dalam kehidupan nyata, mobil adalah objek. Mobil memiliki atribut, seperti berat, dan warna, dan metode nya, seperti drive,dan rem.

* **Class**

Class mendefinisikan karakteristik-karakteristik abstrak dari sesuatu (objek), termasuk karakteristik dan perilaku dari objek itu sendiri. Class dapat dilustrasikan sebagai sebuah cetak biru (blueprint), prototype, atau pabrik (factory) yang berfungsi untuk menghasilkan objek-objek.



Untuk membuat class, gunakan kata kunci **class**. Dibawah ini merupakan contoh membuat class baru bernama “**MyClass**” dengan variabel x.

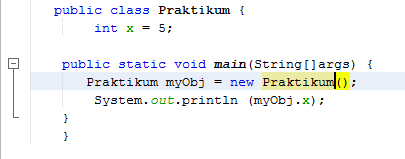


* **Object**
* **Buat Object**

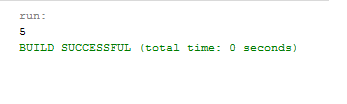
Di java, object dibuat dari class, dalam contoh ini telah dibuat class bernama **Praktikum,** jadi sekarang bisa digunakan untuk membuat object. Untuk membuat object **Praktikum,** tentukan nama class, diikuti nama object, dan gunakan kata kunci **new**:

Contoh:

Buat object yang disebut “**myObj”** dan cetak nilai x:

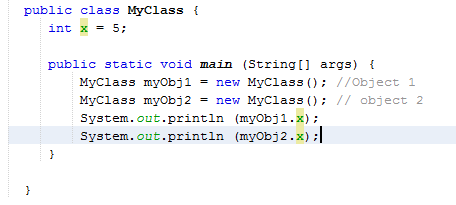


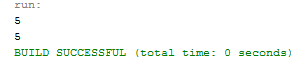
Akan didapat hasil running seperti ini:



* **Banyak Object**

Ketika membuat sebuah program, dapat membuat beberapa objek dari satu kelas. Berikut adalah salah satu contoh program yang membuat banyak objek dalam satu kelas:

 Jika menjalankan proses running, akan didapat hasil seperti gambar berikut:



* **Menggunakan Multiple Objects**

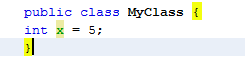
Ketika membuat sebuah objek, disini bisa juga untuk membuat objek kelas dan mengaksesnya di kelas lain. Ini sering digunakan untuk organisasi kelas yang lebih baik (satu kelas memiliki semua atribut dan metode, sedangkan kelas lainnya untuk **main( )** metode (kode yang akan diekseskusi)).

Perlu diingat bahwa nama file java harus sesuai dengan nama kelas. Dalam contoh ini, telah dibuat dua file direktori/pada folder yang sama:

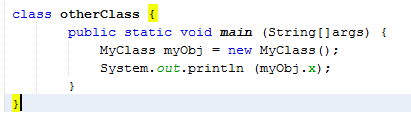
1. MyClass.java
2. otherClass.java

Source code : code membuat object dari banyak class

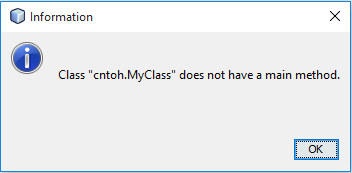
1. MyClass.java



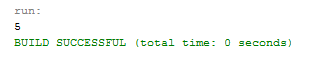
1. otherClass.java



Ketika dilakukan proses running pada class MyClass.java, maka akan mucul kotak dialog seperti gambar dibawah ini:



Munculnya kotak dialog tersebut, dikarenakan pada class MyClass.java tidak mempunyai main( ) metode. Namun, berbeda ketika melakukan proses running pada class otherClass.java. berikut adalah gambar dari hasil running class otherClass.java:

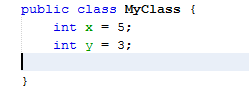


1. **Java Class Attributes**

* **Java Class Attributes**

Pada bab Java Classes and Objects, digunakan istilah variabel untuk x dalam contoh seperti gambar dibawah. Ini sebenarnya adalah atribut dari class atau bisa dikatakan bahwa atribut kelas adalah variabel dalam class.

Untuk membuat class attribute, dibuat class yang disebut MyClass dengan dua atribut x dan y:

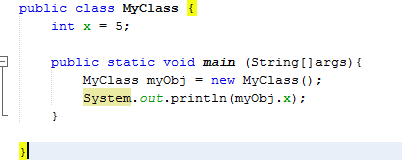


* **Accessing Attributes**

Untuk mengakses atribut, bisa juga dengan membuat object class, dan dengan menggunakan sintaks dot (.):

Contoh berikut akan dibuat object MyClass class, dengan nama myObj. Digunakan “x” atribut pada object untuk mencetak nilaiya:

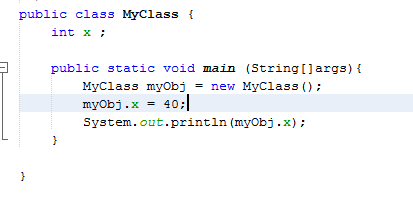
Berikut adalah program untuk membuat object “myObj” dan mencetak nilai x:



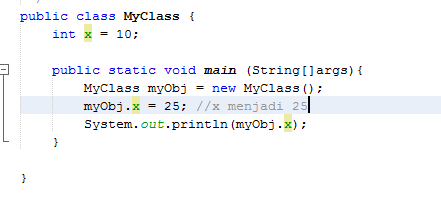
* **Modify Attributes**

Untuk mengubah nilai attribute, bisa juga melihat source code pada contoh dibawah ni:

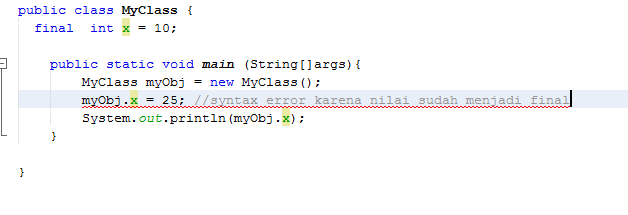
Nilai x = 40



Untuk menimpa nilai yang ada dengan mengubah nilai x menjadi 25:



Jika tidak ingin menimpa nilai yang ada, maka nyatakan atribut sebagai final:



* **Multiple Objects**

Ketika akan membuat beberapa object dari satu class, bisa juga mengubah nilai atribut di satu object, tanpa memengaruhi nilai atribut yang lain:

Berikut adalah mengubah nilai x ke 25 di myobj2, dan meninggalkan x di myObj1 berubah menjadi:



1. **Java Class Methods**

* **Java Class Method**

Method merupakan suatu operasi berupa fungsi-fungsi yang dapat dikerjakan oleh suatu object. Method didefinisikan pada class akan tetapi dipanggil melalui object. Metode menentukan perilaku objek, yakni apa yang terjadi ketika objek itu dibuat serta berbagai operasi yang dapat dilakukan objek sepanjang hidupnya. Ada 4 (Empat) bagian dasar yang dimiliki metode antara lain:

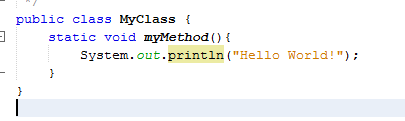
1. Nama Method
2. Tipe Objek atau Tipe primitive yang di kembalikan ke objek
3. Daftar Parameter
4. Badan atau Isi Method

Tiga bagian pertama mengindikasikan informasi penting tentang metode itu sendiri. Dengan kata lain, nama metode tersebut=metode lain dalam program. Dalam java kita dapat memiliki metode-metode berbeda yang memiliki nama sama tetapi berbeda tipe kembalian atau daftar argumennya, sehingga bagian-bagian definisi metode ini menjadi penting. Ini disebut overloading metode(proses yang berlebihan dalam suatu mtehod).

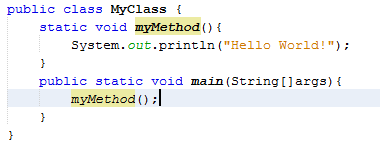
Untuk menjalankan program yang memiliki sifat polymorphism tersebut, diperlukan suatu kemampuan overloading, yaitu suatu kemampuan untuk menentukan fungsi yang mana yang harus digunakan atau dijalankan jika terdapat nama fungsi yang sama. Polimorfisme bisa diartikan seperti kemampuan suatu variable untuk mengubah perangai sesuai dengan objek hasil instansiasi yang digunakan. Polimorfisme membiarkan lebih dari 1 objek dari sub class sub class dan diperlakukan sebagai objek dari super class tunggal.

Metode harus dideklarasikan di dalam kelas. Itu didefinisikan dengan nama metode, diikuti oleh tanda kurung () . Java menyediakan beberapa metode yang telah ditentukan, seperti System.out.println() , tetapi Anda juga dapat membuat metode Anda sendiri untuk melakukan tindakan tertentu:

Contoh:



myMethod() mencetak teks (tindakan), ketika dipanggil. Untuk memanggil metode, bisa dituliskan nama metode diikuti oleh dua tanda kurung() dan tanda titik koma ; pada akhir metode. Jika menggunakan **main**  maka metode dipanggil pada class **main**



* **Static dan Non Static**

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

Method static adalah method yang dapat digunakan tanpa harus membuat objek dari class.tidak seperti public, yang hanya dapat diakses oleh objek:

Contoh perbedaan static dan public

Apabila sebuah variabel didefinisikan static didalam sebuah class, maka untuk memanggil variabel tersebut kita tidak perlu membuat objek dari class tersebut, namun langsung bisa memanggil variabel tersebut dari nama class dimana dia dideklarasikan.

Contoh program :

|  |
| --- |
| *public class Siswa { public static int umur = 18; }* |

Bila kita ingin memanggil variabel umur yang dideklarasikan static tersebut maka kita cukup memanggil seperti berikut ini :

|  |
| --- |
| *int umurAndi = Siswa.umur;* |

Disini Siswa adalah nama class, dan bukan merupakan variabel sebuah objek.  
Apabila kita mengubah variabel tersebut menjadi non-static (bukan static), contoh nya:

|  |
| --- |
| *public class Siswa { public int umur = 18; }* |

Maka untuk memanggilnya kita harus membuat objek siswa terlebih dahulu :

|  |
| --- |
| *Siswa siswaAndi = new Siswa(); int umurAndi = siswaAndi.umur;* |

**Contoh:**

|  |
| --- |
| public class MyClass {  static void myMethod() {  System.out.println("Hello World!");  }  public static void main(String[] args) {  myMethod();  }  } |

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

**Contoh 2:**

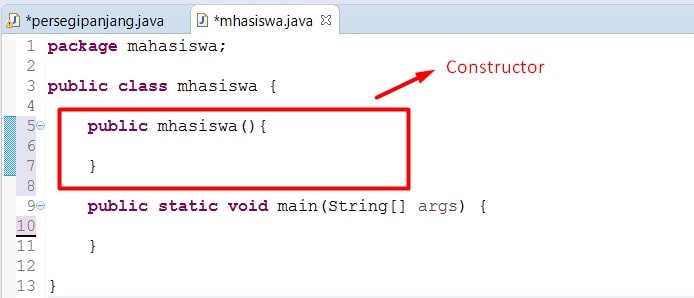
|  |
| --- |
| **//** Create a Car class  public class Car {    // Create a fullThrottle() method  public void fullThrottle() {  System.out.println("The car is going as fast as it can!");  }  // Create a speed() method and add a parameter  public void speed(int maxSpeed) {  System.out.println("Max speed 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  }  } |

Penjelasan dari contoh di atas :

* Kami menciptakan kelas Car khusus dengan kata kunci class .
* Kami menciptakan metode fullThrottle() dan speed() di kelas Car .
* Metode fullThrottle() dan metode speed() akan mencetak beberapa teks, ketika mereka dipanggil.
* Metode speed() menerima parameter int disebut maxSpeed - kita akan menggunakannya dalam **8)** .
* Untuk menggunakan kelas Car dan metodenya, kita perlu membuat **objek** dari Kelas Car .
* Lalu, buka metode main() , yang Anda tahu sekarang adalah metode Java bawaan yang menjalankan program Anda (kode apa pun di dalam main dieksekusi).
* Dengan menggunakan kata kunci new kami membuat objek Car dengan nama myCar .
* 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() .

1. **Java Constructors**

Java konstruktors adalah **metode khusus** yang digunakan untuk menginisialisai objek. Konstrutor dipanggil ketika objek kelas dibuat. Construktors berfungsi untuk memberikan nilai awal pada sebuah class ketika class tersebut dibuat dalam bentuk objek pada class lain.



* Konstruktor mempunyai ciri yaitu :  
  - mempunyai nama yang sama dengan nama class  
  - tidak mempunyai return type (seperti void, int, double dll
* Tipe Constructor
* Default constructor (tanpa argumen)
* Constructor berparameter
* **Perbedaan Construktor adan Methods**

|  |  |
| --- | --- |
| **Construktor** | **Methods** |
| Constructor digunakan untuk menginisialisasi status sebuah objek. | Metode digunakan untuk mengekspos perilaku objek. |
| Konstruktor tidak harus memiliki jenis kembali | Metode harus memiliki jenis kembali. |
| Konstruktor dipanggil implisit | Metode dipanggil secara eksplisit. |
| Kompiler Java menyediakan konstruktor default jika Anda tidak memiliki konstruktor | Metode ini tidak disediakan oleh kompilator dalam hal apapun. |
| Nama konstruktor harus sama dengan nama kelas | Nama metode mungkin atau mungkin tidak sama dengan nama kelas. |

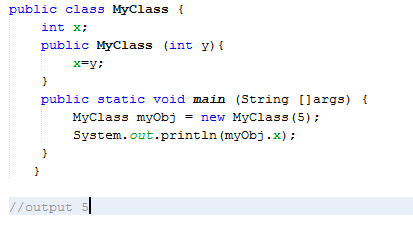
|  |
| --- |
| // Create a MyClass class  public class MyClass {  int x; // Create a class attribute  // Create a **class constructor** for the MyClass class  public MyClass() {  x = 5; // Set the initial value for the class attribute x  }  public static void main(String[] args) {  MyClass myObj = new MyClass(); // Create an object of class MyClass (This will **call the constructor**)  System.out.println(myObj.x); // Print the value of x  }  } |

Perhatikan bahwa nama konstruktor harus **sesuai dengan nama kelas** , dan tidak dapat memiliki **jenis kembali** (seperti void).Juga mencatat bahwa konstruktor dipanggil saat objek dibuat.Semua kelas memiliki konstruktor secara default: jika Anda tidak membuat konstruktor kelas sendiri, Java menciptakan satu untuk Anda. Namun, maka Anda tidak dapat mengatur nilai awal untuk atribut objek

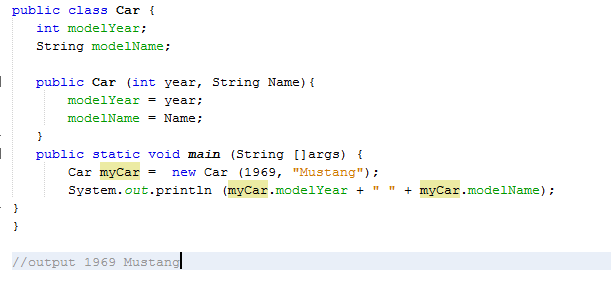
* **Constructors Parameter**

Constructors biasanya digunakan untuk initialize (menyiapkan) data untuk class. Untuk membuat ini, terlebih dahulu membuat parameter sebagai inputan constructor. Contoh berikut menambahkan “int y” parameter ke constructor. Di dalam constructoratur x ke y (x = y). ketika memanggil constructor, dapat juga meneruskan parameter ke constructor (5), yang akan menetapkan nilai x ke 5.

Berikut adalah contoh constructor parameter dengan satu parameter:



Contoh constructors dengan banyak parameter:



1. **Java Modifier**

Modifier adalah Java keyword yang digunakan untuk memberikan "Sesuatu yang berbeda" pada kelas, method, ataupun property yang menggunakan modifier tersebut. dengan kata kunci public yang muncul hampir pada semua program java,seperti contoh:

|  |
| --- |
| **public** class MyClass |

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

* Java modifiers dapat dibagi menjadi dua kelompok :

1. Access modifier : untuk mengontrol level akses atau sebuah *“hak akses”* yang diberikan kepada variabel, method atau class yang bertujuan untuk menjaga integritas dari data ketika ingin diakses oleh object lain. Hak akses tersebut diberikan oleh si pembuat program. Dengan adanya Access Modifier, kita dapat membatasi resource-resource mana saja yang

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| No Modifier | Y (Dapat Diakses) | Y (Dapat Diakses) | N (Tidak Dapat Diakses) | N (Tidak Dapat Diakses) |
| Public | Y (Dapat Diakses) | Y (Dapat Diakses) | Y (Dapat Diakses) | Y (Dapat Diakses) |
| protected | Y (Dapat Diakses) | Y (Dapat Diakses) | Y (Dapat Diakses) | N (Tidak Dapat Diakses) |
| Private | Y (Dapat Diakses) | N (Tidak Dapat Diakses) | N (Tidak Dapat Diakses) | N (Tidak Dapat Diakses) |

dapat diakses oleh object tertentu, turunannya, ataupun oleh method tertentu.

* Ada 4 Macam Acces Modifier di Java sesuai gambar di atas, yaitu:
* ***public***  
  Kelas, method maupun atribut yang mempunyai access modifier public dapat diakses dari manapun dan oleh kelas apapun. Di bawah ini adalah sintak penulisan public access modifier di Java :

|  |
| --- |
| public class Person{  public String name;  public String getName(){  return name;  }  } |

* ***protected***  
  Atribut dan method yang mempunyai access modifier protected, artinya bahwa atribut dan method tersebut dapat diakses oleh kelas yang sama, package yang sama, dan kelas turunannya (*subclass*). Anda tidak boleh memberikan akses protected pada kelas, dan interface. Anda juga tidak boleh memberikan akses protected pada method dan property pada interface.  
    
  Anda dapat memberikan *keyword* **protected** untuk memberikan akses protected pada method, dan property yang akan di berikan akses protected.  
  Di bawah ini adalah sintak penulisan protected access modifier di Java:

|  |
| --- |
| public class Person{  protected String name;  protected String getName(){  return name;  }  } |

Contohnya :

|  |
| --- |
| class Person {  protected String fname = "John";  protected String lname = "Doe";  protected String email = "john@doe.com";  protected int age = 24;  }  class Student extends Person {  private int graduationYear = 2018;  public static void main(String[] args) {  Student myObj = new Student();  System.out.println("Name: " + myObj.fname + " " + myObj.lname);  System.out.println("Email: " + myObj.email);  System.out.println("Age: " + myObj.age);  System.out.println("Graduation Year: " + myObj.graduationYear);  }  } |

* ***default***

Atribut dan method yang mempunyai access modifier default, artinya Kode ini hanya dapat diakses dalam paket yang sama. Ini digunakan ketika Anda tidak menentukan modifier.

Contohnya :

|  |
| --- |
| class Person {  String fname = "John";  String lname = "Doe";  String email = "john@doe.com";  int age = 24;    public static void main(String[] args) {  Person myObj = new Person();  System.out.println("Name: " + myObj.fname + " " + myObj.lname);  System.out.println("Email: " + myObj.email);  System.out.println("Age: " + myObj.age);  }  } |

|  |
| --- |
| public class Person{  private String name;  private String getName(){  return name;  }  } |

* ***private***  
  Atribut dan method yang mempunyai access modifier default, artinya bahwa atribut dan method tersebut dapat diakses oleh kelas yang sama dan package yang sama. Anda dapat memberikan keyword private untuk memberikan akses private pada method atau property yang akan diberikan akses private.  
  Di bawah ini adalah sintak penulisan private access modifier di Java

Contohnya :

|  |
| --- |
| private String fname = "John";  private String lname = "Doe";  private String email = "john@doe.com";  private int age = 24;    public static void main(String[] args) {  Person myObj = new Person();  System.out.println("Name: " + myObj.fname + " " + myObj.lname);  System.out.println("Email: " + myObj.email);  System.out.println("Age: " + myObj.age);  } |

1. Non- Access Modifiers : tidak mengontrol level akses, tetapi menyediakan fungsionalitas lain. Maksudnya adalah modifier jenis ini tidak perlu dituliskan pada variable/method/class, jika menggunakan modifier jenis ini, variable/method/class dapat diakses oleh objek atau class lain asalkan didalam satu package yang sama.

* Beberapa macam Non-Access Modifier, yaitu:
* ***Final***  
  Final adalah salah satu modifier yang digunakan agar suatu atribut atau method bersifat final atau Atribut dan metode tidak dapat ditimpa/dimodifikasi dan tidak bisa diubah nilainya. Modifier ini digunakan untuk membuat konstanta di Java.

Contohnya :

|  |
| --- |
| public class MyClass {  final int x = 10;  final double PI = 3.14;  public static void main(String[] args) {  MyClass myObj = new MyClass();  myObj.x = 50; // will generate an error  myObj.PI = 25; // will generate an error  System.out.println(myObj.x);  }  } |

* ***Static***  
  Static adalah salah satu Atribut dan metode memiliki kelas, bukan objek maksunya suatu atribut atau pun method dapat diakses oleh kelas atau objek tanpa harus melakukan instansiasi terhadap kelas tersebut.  
    
  Method main adalah salah satu contoh method yang mempunyai modifier static.

Contohnya :

|  |
| --- |
| 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  }  } |

* ***Abstract***  
  Abstract adalah modifier yang digunakan untuk membuat kelas dan method abstrak. dan hanya dapat digunakan pada metode. Metode ini tidak memiliki tubuh, misalnya abstrak void Run ();. Tubuh disediakan oleh subclass (diwarisi dari).
* ***Transient***  
  Modifier ini digunakan agar suatu variable tidak bisa di serialisasi. Serialization adalah konsep dimana sebuah objek dapat ditransfer dari suatu aplikasi ke aplikasi lainnya atau dari suatu workstation ke workstation lainnya. Konsep ini sangat diperlukan ketika membuat aplikasi client server. Salah satu tujuan serialization adalah bahwa tidak boleh ada perubahan terhadap atribut pada saat objek di transformasikan menjadi stream.
* ***Synchronized***  
  Synchronized adalah modifier yang digunakan dalam aplikasi Java berbasis thread. Modifier ini menspesifikasikan bahwa method merupakan thread safe. Artinya bahwa hanya ada satu jalur atau satu waktu eksekusi pada method yang menggunakan modifier jenis ini dan memaksa thread thread lain menunggu giliran.
* ***Volatile***

Nilai atribut tidak di-cache thread-lokal, dan selalu membaca dari "memori utama "

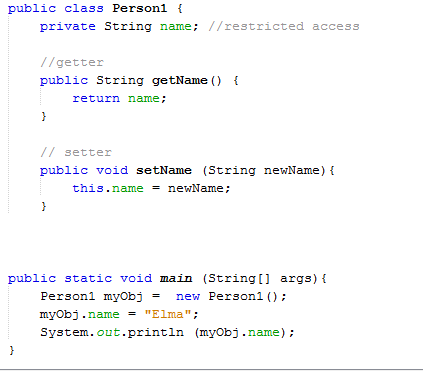
1. **Java Encapsulation**

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 :

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

* **Get and 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 **syntaks** untuk keduanya adalah bahwa dimulai dengan salah satu **get** atau **set** , diikuti oleh nama variabel , dengan huruf pertama menggunakan huruf kapital. Berikut adalah contoh penggunaan dari **get** and **set:**



Penjelasan dari contoh yaitu method **get** berfungsi untuk mengembalikan nilai variabel **name.** Sedangkan, the **set** method berfungsi untuk mengambil parameter (**newName**) dan penerima ke **name**  variabel. Kata kunci “**this”** digunakan untuk merujuk ke objek saat ini. Namun, ketika variabel **name**  di deklarasikan sebagai **private** , maka tidak dapat diakses dari luar class. Jika tetap mencoba untuk mengakses variabel **private,** maka akan terjadi kesalahan. Sebagai gantinya, maka digunakan method **getName( )** dan **setName ( )**  untuk mengakses dan memeprbarui variable. 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 Packages/API**

Sebuah package di java digunakan untuk kelompok terkait kelas. Anggap itu sebagai folder dalam direktori file. Kami menggunakan paket untuk menghindari konflik nama, dan untuk menulis kode yang lebih baik maintainable. Atau bisa juga merupakan komponen-komponen dan kelas JAVA yang sudah jadi, yang memiliki berbagai kemampuan. Kemampuan untuk menangani objek, string, angka, dsb.  Package dibagi menjadi dua Kategori:

* Paket Built-in (paket dari Java API)
* User-defined Packages (buat paket Anda sendiri)
* Paket Built-in (paket bawaan)

Adalah Paket Java. Sebuah paket Java adalah sekelompok serupa jenis kelas, antarmuka dan sub-paket. Paket di Java dapat dikategorikan dalam dua bentuk, built-in paket dan paket yang ditetapkan pengguna. Ada banyak built-in paket seperti Java, lang, AWT, javax, ayunan, net, IO, util, SQL dll.

|  |  |
| --- | --- |
| Paket (package) | Keterangan |
| javax.swing | Paket ini berisi kelas-kelas dan interface untuk komponen GUI swing Java yang menyediakan dukungan untuk tampilan grafis yang portabel. |
| javax.swing.event | Paket ini berisi kelas-kelas dan interfes yang memperbolehkan penanganan event untuk komponent grafis yang terletak di paket javax.swing. |
| java.lang | Paket ini berisi kelas-kelas dan interfes yang diperlukan oleh banyak program Java. Paket ini diimpor oleh kompiler ke semua program Java secara otomatis. |
| java.applet | Paket ini berisi kelas-kelas Applet dan juga beberapa interfes yang memperbolehkan interaksi applet dan browser serta untuk memainkan klip audio. Kelas javax.swing.JApplet digunakan untuk menetapkan applet yang menggunakan komponen GUI swing. |
| java.text | Paket ini berisi kelas-kelas dan interfes yang memperbolehkan program Java untuk memanipulasi angka, tanggal, karakter dan juga string. |
| java.net | Paket ini berisi kelas yang memperbolehkan program untuk berkomunikasi melalui jaringan. |

Perpustakaan berisi komponen untuk mengelola input, pemrograman basis data, dan banyak lagi lainnya. Daftar lengkap dapat di temukan 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 import kata kunci:

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

## Impor Kelas

|  |
| --- |
| import java.util.Scanner; |

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

Dalam contoh diatas, java.util adalah paket, sedangkan scanner kelas dari java.util paket

Untuk menggunakan scanner kelas, buat objek kelas dan gunakan salah satu metode yang tersedia dan ditemukan dalam scanner dokumentasi kelas dalam contoh kita akan menggunakan nextLine() metode, yang di gunakan untuk membaca baris lengakap.

* **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

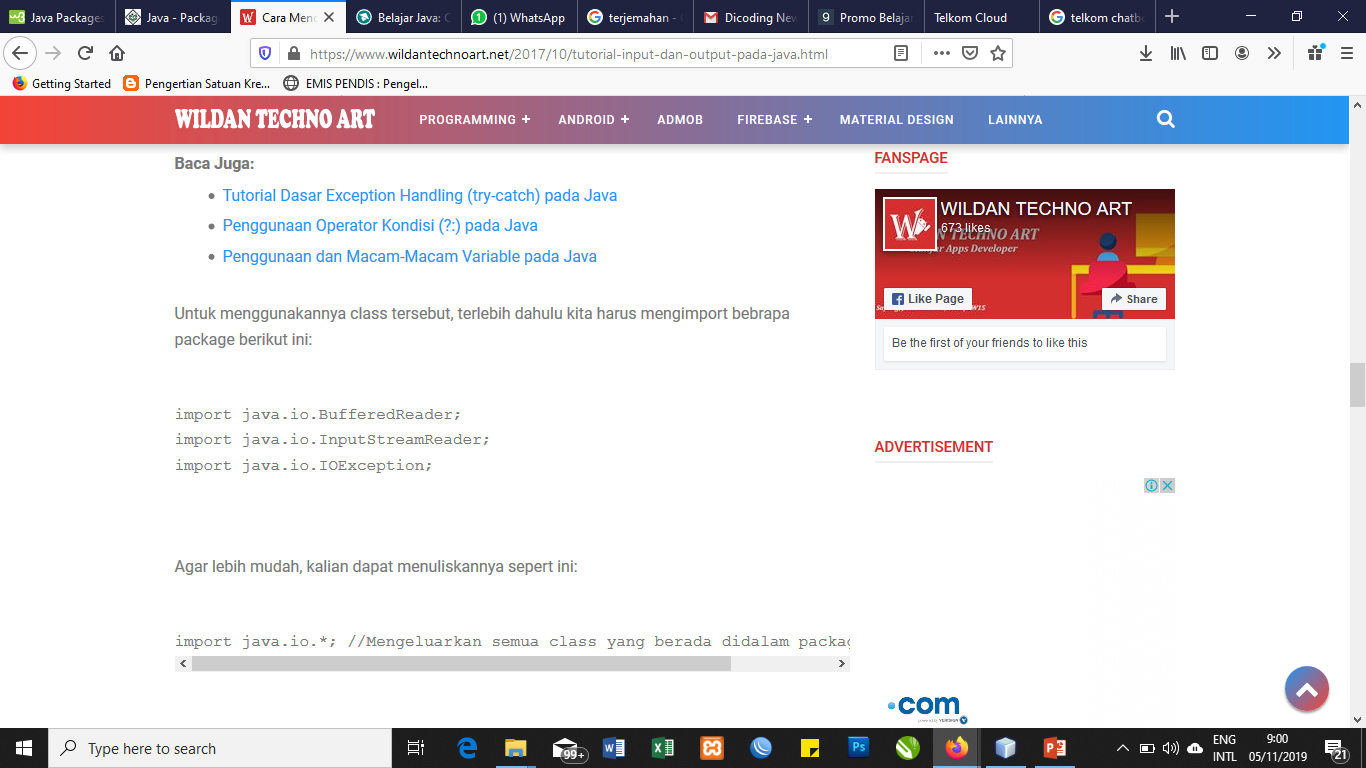
**- nextShort**  - Menerima bilangan bulat/Short

Hasil percobaan :

|  |
| --- |
| import java.util.Scanner; // import the Scanner class  class MyClass {  public static void main(String[] args) {  Scanner myObj = new Scanner(System.in);  String userName;    // Enter username and press Enter  System.out.println("Enter username");  userName = myObj.nextLine();    System.out.println("Username is: " + userName);  }  } |

* **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.



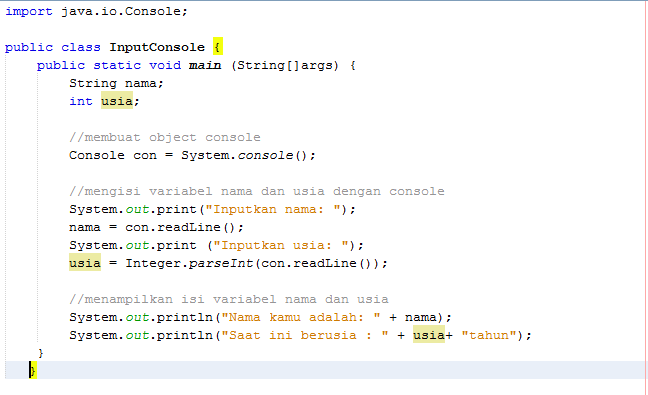
* **Class JOptionPane**

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

* **Class Console**

Class console hampir sama dengan BufferReader. Class console 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 pada NetBeans. Maka dari itu, harus dikompilasi secara manual.untuk menggunakan class ini, diperlukan untuk mngimpornya terlebih dahulu.

Contoh: ketik printah javac input Console.java untuk melakukan kompilasi



* **Import Package**

Terdapat banyak package untuk dipilih. Pada contoh sebelumnya, percobaan menggunakan class **Scanner**  dari **java.util** package. Package ini berisi fasilitas tanggal dan waktu, generator nomor acak, dan manfaat class lainnya.

Untuk mengimpor seluruh package, maka akhiri kalimat dengan tanda bintang (\*). Contoh berikut akan mengimpor semua class dalam **java.util** package.



* **Daftar Package pada Program Java**

Berikut ini adalah beberapa daftar package java yang sering digunakan pada program java:

1. javax.swing

paket ini berisi kelas-kelas dari interface untuk komponen GUI swing java yang menyediakan dukungan untuk tampilan grafis yang portable.

1. Javax.swing.event

Paket ini berisi kelas-kelas dari interface yang memperbolehkan penanganan event untuk komponen grafis yang terletak di paket javax.swing

1. Java.lang

Paket ini berisi kelas-kelas dari interface yang diperlukan olehbanyak program java. Paket ini diimpor oleh compiler ke semua program java secara otomatis.

1. Java.applet

Paket ini berisi kelas-kelas applet dan juga beberapa interface yang memperbolehkan interakse applet dan browser serta untuk memainkan klip audio. Kelas javax.swing.JApplet digunakan untuk menetapkan applet yang menggunaka komponen java GUI swing.

1. Java.text

Paket ini berisi kelas-kelas dari interface yang memperbolehkan program java untuk memanipuasi angka, tanngal, karakterdan juga string.

1. Java.net

Paket ini berisi kelas yang memperbolehkan program untuk berkomunikasi melalui jaringan.

* **Java API terdiri dari tiga bagian utama:**
* **Java Standard Edition (SE)**, sebuah standar API untuk merancang aplikasi desktop dan*applets*dengan bahasa dasar yang mendukung grafis, keamanan, konektivitas basis data dan jaringan.
* **Java Enterprose Edition (EE)**, sebuah inisiatif API untuk merancang aplikasi serverdengan mendukung untuk basis data.
* **Java Macro Edition (ME)**, sebuah API untuk merancang aplikasi yang jalan pada alat kecil seperti telepon genggam, komputer genggam dan pager.

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.

**KATA KUNCI 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.

Ketika suatu class dideklarasikan sebagai final maka class tersebut tidak bisa diwariskan.

* **Contoh**

|  |
| --- |
| class Vehicle {  protected String brand = "Ford"; // Vehicle attribute  public void honk() { // Vehicle method  System.out.println("Tuut, tuut!");  }  }  class Car extends Vehicle {  private String modelName = "Mustang"; // Car attribute  public static void main(String[] args) {  // membuat sebuah objek car  Car myCar = new Car();  // Panggil metode Honk () (dari kelas kendaraan) pada objek mobil akseshalamanprofil  myCar.honk();  // Menampilkan nilai dari atribut merek (dari kelas kendaraan) dan nilai modelName dari kelas mobil  System.out.println(myCar.brand + " " + myCar.modelName);  }  } |

1. **Java Polymorphism**

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.

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

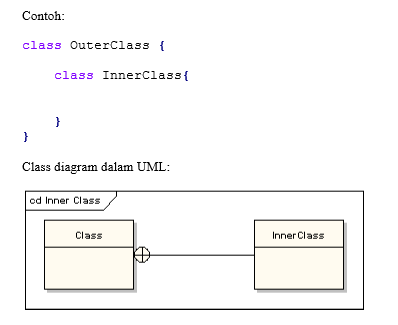
Contoh :

|  |
| --- |
| class Animal {  public void animalSound() {  System.out.println("The animal makes a sound");  }  }  class Pig extends Animal {  public void animalSound() {  System.out.println("The pig says: wee wee");  }  }  class Dog extends Animal {  public void animalSound() {  System.out.println("The dog says: bow wow");  }  }  class MyMainClass {  public static void main(String[] args) {  Animal myAnimal = new Animal(); // Create a Animal object  Animal myPig = new Pig(); // Create a Pig object  Animal myDog = new Dog(); // Create a Dog object  myAnimal.animalSound();  myPig.animalSound();  myDog.animalSound();  }  } |

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.

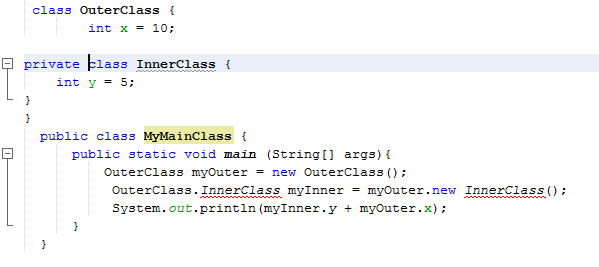
Untuk mengakses inner class, buatlah object dari kelas luar, dan kemudian buat objek dari kelas dalam.  sama halnya seperti variable dan method. Hal ini mengakibatkan instance dari Inner Class dapat mengakses /berbagi dengan instance dari outer Class-nya. Instance dari Inner Class dapat mengakses semua member dari outer Class-nya, bahkan yang private.

[](https://zegyjib.files.wordpress.com/2013/04/in1.png)

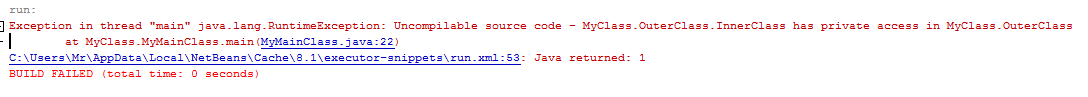
* **Private Inner Class**

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

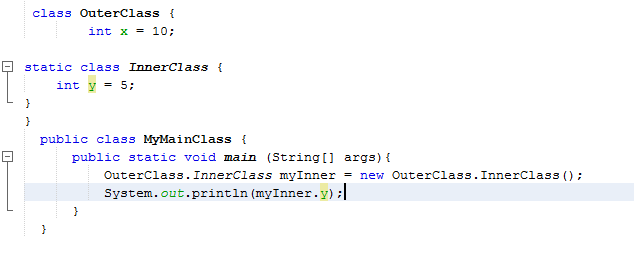
Seperti contoh dibawah ini:



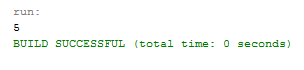
Jika dilihat dalam gambar, maka muncul garis merah pada syntax yang salah. Dan jika tetap ingin mengakses private inner class dari kelas luar (MyMainClass), maka terdapat kesalahan (error):



* **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:

Dan akan memperoleh hasil running seperti gambar dibawah ini:



* **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:



1. **Java Abstraction**

Kelas abstrak (*abstract class*) adalah kelas yang mengandung komsep abstrak, dan tidak alan pernah bisa diinstansiasi. Kelas abstrak didefinisikan dengan tujuan untuk digunakan dan diperluas oleh kelas lain. Dengan demikian, kelas ini merupakan cikal bakal dari superkelas.

Abstraction Class adalah sebuah kelas abstrak yang memiliki metod dan atribut. Abstraction class adalah hirarki tertinggi dari sub-sub classnya. Kelas abstrak digunakan untuk membuat sebuah kelas yang memiliki method yang belum jelas implementasinya.

Berikut ini aturannya:

1. Jika sebuah kelas memiliki method abstrak maka kels itu harus menjadi kelas abstrak.
2. Sebuah kelas abstrak dapat saja memiliki method yang tidak abstrak.
3. Jika sebuah kelas abstrak diturunkan menjadi kelas konkret, maka semua method abstrak dari kelas abstrak haruslah ditulis ulang/ dibuat ulang di kelas konkretnya dan diberi detail dari methodnya.
4. Jika method abstrak di turunkan dan kelas turunannya adalah kelas abstrak, maka tidak perlu menulis ulang method abstraknya.

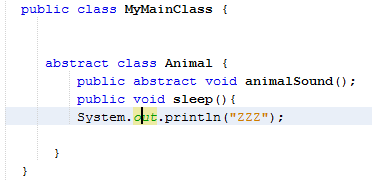
* **Java Abstract Classes dan Methods**

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:

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

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

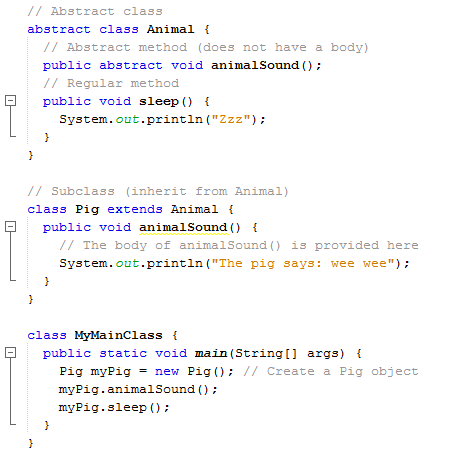


Dari contoh diatas, tidak mungkin membuat objek dari kelas Animal:



Untuk mengakses kelas 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.**

**Contohnya :**



1. **Java Interface**

Interface merupakan kumpulan deklarasi fungsi tanpa implementasi yang mendefinisikan aturan pemanggilan fungsi oleh sembarang class lain. Interface mendefinisikan sebuah cara standar dan umum dalam menetapkan sifat-sifat dari class-class. Interface memungkinkan pemrogram untuk berbagi konstanta atau menentukan bentuk metode yang dapat digunakan oleh sejumlah class.

Interface digunakan jika kita ingin class yang tidak berhubungan mengimplementasikan method yang sama. Melalui interface-interface, kita dapat menangkap kemiripan diantara class yang tidak berhubungan tanpa membuatnya seolah-olah class yang berhubungan. Ciri-ciri dari interface:

* Method interface tidak punya tubuh, sebuah interface hanya dapat mendefinisikan konstanta dan interface tidak langsung mewariskan hubungan dengan class istimewa lainnya, mereka didefinisikan secara independent.
* Tidak bisa membuat instance atau objek baru dari sebuah interface.
* Hirarki antara class interface dan class yang mengimplementasinya adalah independen atau tidak saling berelasi
* Interface diimplementasikan ke dalam suatu class dengan menggunakan kata kunci implements, dan sebuah kelas dapat mengimplementasikan lebih dari satu interface.
* Ciri umum lain adalah baik interface maupun class dapat mendefinisikan method. Interface tidak punya sebuah kode implementasi sedangkan class abstrak memiliki salah satunya.

Cara lain untuk mencapai abstraksi di Java, adalah dengan interface. Interface adalah "kelas abstrak" yang sepenuhnya digunakan untuk mengelompokkan metode terkait dengan benda kosong.

Contoh :

|  |
| --- |
| // 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 implement (alih-alih meluas). Isi metode interface disediakan oleh kelas "implement".

Contoh :

|  |
| --- |
| interface Animal{  public void AnimalSound();  public void Sleep();    }  class pig implements Animal{  public void AnimalSound(){  System.out.println ("the pig a 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();  }  } |

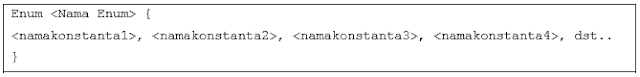
* Catatan tentang Interface:
* Seperti kelas abstrak, interface tidak dapat digunakan untuk membuat objek (dalam contoh di atas, tidak mungkin membuat objek "Hewan" di MyMainClass)
* Metode interface tidak memiliki body, body disediakan oleh kelas "implement"
* Pada implementasi interface, kita harus mengganti semua metodenya
* Metode interface secara default abstrak dan public
* Atribut interface secara default publik, statis dan final
* Interface tidak dapat berisi konstruktor (karena tidak dapat digunakan untuk membuat objek)

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;  } |

Keteterangan:

* **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**

*Scanner* merupakan *class* yang menyediakan fungsi-fungsi untuk mengambil input dari keyboard.

Agar kita bisa menggunakan Scanner, kita perlu mengimpornya ke dalam kode:

import java.util.Scanner;

Kelas **Scanner** digunakan untuk mendapatkan input pengguna, dan ditemukan dalam paket java.util .

Untuk menggunakan kelas **Scanner ,** buat objek kelas dan gunakan salah satu metode yang tersedia yang ditemukan dalam dokumentasi kelas **Scanner** . Dalam contoh kita, kita akan menggunakan metode **nextLine()** , yang digunakan untuk membaca 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 | Keterangan |
| nextBoolean() | Membaca nilai Boolean dari pengguna |
| nextByte() | Membaca nilai byte dari pengguna |
| nextDouble() | Membaca nilai ganda dari pengguna |
| nextFloat() | Membaca nilai Float dari pengguna |
| nextInt() | Membaca nilai int dari pengguna |
| nextLine() | Membaca nilai string dari pengguna |
| nextLong() | Membaca nilai panjang dari pengguna |
| nextShort() | Membaca nilai singkat dari pengguna |

## Contoh:

|  |
| --- |
| import java.util.Scanner;  class MyClass {  public static void main(String[] args) {  Scanner myObj = new Scanner(System.in);  System.out.println("Enter name, age and salary:");  // String input  String name = myObj.nextLine();  // Numerical input  int age = myObj.nextInt();  double salary = myObj.nextDouble();  // Output input by user  System.out.println("Name: " + name);  System.out.println("Age: " + age);  System.out.println("Salary: " + salary);  }  } |

**Catatan:**  Jika Anda memasukkan input yang salah (mis. Teks dalam input numerik), Anda akan mendapatkan pesan pengecualian / kesalahan (seperti "InputMismatchException").

## Java Date

* **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().

Contohnya :

*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(). Contohnya hampir sama dengan tampilan yang diatas hanya saja perbedaannya pada script import java.time.LocalDate;(pada tanggal)dan *import java.time.LocalDate;*(pada waktu).

Contohnya :

*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().

Contohnya :

*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);*

Selanjutnya , kita memanggil objek atau menampilkan hasil outputannya .

}

}

* **Memformat 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 :

Contohnya :

*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 sudah di edit pada format yang di ubah.

}

}

Outputnya :

Before Formatting: 2020-01-02T21:33:03.906715  
After Formatting: 02-01-2020 21:33:03

1. **Java ArrayList**

ArrayList pada bahasa pemrograman Java, merupakan salah satu *collection*yang bisa kalian gunakan untuk menampilkan daftar atau list nilai/value, yang bersifat dinamis dan juga dapat dimodifikasi, ukuran pada ArrayList dapat kita ubah dengan cara menambahkan (add) atau menghapusnya (remove). Tidak seperti Array pada umumnya, pada ArrayList kita tidak dapat menggunakan tipe data primitive, seperti int, char, boolean, float, dsb. Yang membedakan Array dengan ArrayList lainnya adalah kita dapat mendefinisikan nilai berapapun yang kita mau, berdasarkan method dari objek yang kita gunakan, karena ArrayList memunyai sifat yang dinamis. Pada meteri kali ini, kita akan belajar cara menggunakan ArrayList serta perbedaannya dengan Array pada program Java.

beberapa perbedaan lainnya dari ArrayList dengan Array, yaitu seperti berikut ini:

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Tidak bisa menggunakan generic | Bisa menggunakan generics untuk menjaga keamanan tipe data dalam Arrays |
| Bersifat statis ukuran data nya tidak bisa berubah sesuai dengan saat pertama kali dibuat/definisikan. | Bersifat dinamis, ukuran datanya dapat di ubah dengan menambahkan atau hapus |
| Dapat menggunakan tipe data primitive, seperti init, boolean, char, float,dsb. | Tidak dapat menggunakan tipe data primitive |

Untuk mendefinisikan ArrayList pada Java, sintaxnya seperti berikut ini:

List<TipeData> objek = new ArrayList<>();

* + Beberapa operasi dapat Anda lakukan terhadap ArrayList seperti berikut:
* **size()**, untuk mngetahui berapa banyak elemen yang dimiliki  *ArrayList,* gunakan size pada metode ini: cars.size();
* **add()**, untuk menambah elemen baru pada ArrayList gunakan add() pada index : cars()
* **get()**, untuk mengakses elemen di ArrayList gunakan get() pada indeks : *cars.get(0);*
* **isEmpty()**, untuk memeriksa apakah *ArrayList* kosong atau tidak
* **indexOf()**, untuk mengetahui indeks dari suatu nilai
* **contains()**, untuk memeriksa apakah suatu nilai ada dalam *ArrayList*
* **set()**, untuk memodifikasi elemen, gunakan set() metode dan merujuk kenomor index : *cars.set(0,”opel”);*
* **remove()**: untuk menghapus nilai/elemen gunakan remove() pada indeks tertentu: *cars.remove(0);*
* **clear()** : untuk menghapus semua elemen di ArrayList, gunakan clear() metode : cars.clear();
  + Loop Through ArrayList

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

* + Tipe yang lain

Elemen dalam ArrayList sebenarnya adalah objek. 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 primitive lainnya, gunakan: Boolean untuk boolean, Character untuk char, Double untuk double, dll:

* + 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.

Sort an ArrayList of Strings:

Contohnya:

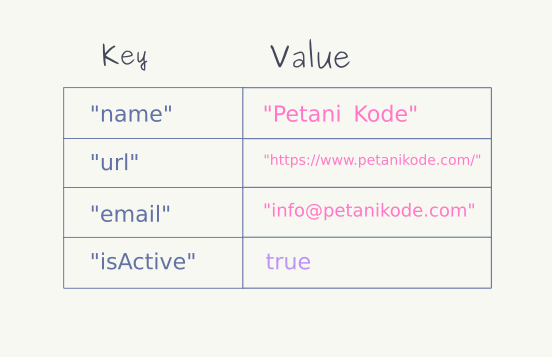
|  |
| --- |
| import java.util.ArrayList;  import java.util.Collections; // Import the Collections class  public class MyClass {  public static void main(String[] args) {  ArrayList<String> cars = new ArrayList<String>();  cars.add("Volvo");  cars.add("BMW");  cars.add("Ford");  cars.add("Mazda");  Collections.sort(cars); // Sort cars  for (String i : cars) {  System.out.println(i);  }  }  } |

Jika memakai integer/angka :

|  |
| --- |
| import java.util.ArrayList;  import java.util.Collections; // Import the Collections class  public class MyClass {  public static void main(String[] args) {  ArrayList<Integer> myNumbers = new ArrayList<Integer>();  myNumbers.add(33);  myNumbers.add(15);  myNumbers.add(20);  myNumbers.add(34);  myNumbers.add(8);  myNumbers.add(12);  Collections.sort(myNumbers); // Sort myNumbers  for (int i : myNumbers) {  System.out.println(i);  }  }  } |

## Java HashMap

## HashMap adalah sebuah class yang berisi sekumpulan pasangan nilai (value) dan kunci (key). Nilai(value) bisa dalam bentuk *string, integer, boolean, float, double*, dan *objek*. Sedangkan untuk kunci(key) biasanya dalam bentuk *string* dan *integer*.

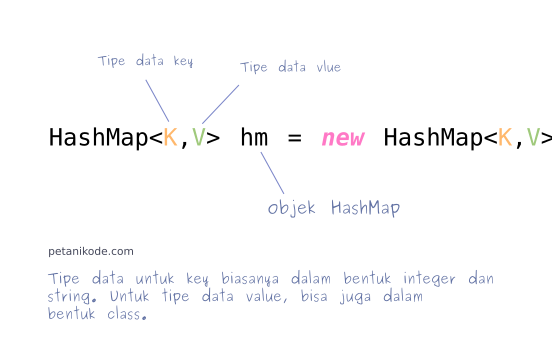
Coba perhatikan tabel berikut:

Pada tabel tersebut, **key**digunakan sebagai kunci untuk menampilkan atau mengakses value, key pada HashMap dapat kalian isi dengan bebas, ini yang membedakan HasMap dengan ArrayList, jika pada ArrayList untuk menampilkan nilai, kita perlu mengakses posisi indexnya, tetapi pada HashMap, untuk mengakses value/nilai tersebut, kita perlu memanggil keynya.Sebelum bisa menggunakan HashMap, kita harus mengimpornya terlebih dahulu:

import java.util.HashMap;

untuk menggunakan HashMap, kita harus membuat objeknya terlebih dahulu. Objek dari Hashmap dapat dibuat dengan kata kunci new.

Namun, ada sedikit tambahan untuk menentukan tipe data untuk key dan value yang akan disimpan.



* K (key : Digunakan sebagai Kunci Untuk Memanggil Value): Integer..
* V (value : Merupakan Isi Nilai Yang Akan Dipanggil Oleh Key): String

Selain tipe data String , kita juga dapat menggunakan Object lain ,seperti **List**atau **Set**. Untuk digunakan sebagai Valuenya.

* Tambahkan Item

Pada HashMap kelas emiliki banyak metode yang berguna misalnya, untuk menambahkan item ke dalamnaya gunakan put() pada metode ini:

Capitalcities.put();

* Akses suatu item

Untuk mengakses nilai dalam HashMap gunakan metode get() dan lihat kunciya:

Capitalcities.get();

* Hapus item

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

Capitalcities.remove();

Dan untuk menhapus semua item gunakan metode ini clear() dan kuncinya:

Capitalcities.clear();

* Ukuran HashMap

Untuk mengetahui berapa banyak item yang ada gunakan metode size dan kuncinya:

Capitalcities.size();

## Loop Through hashMap

## Loop melalui item HashMap dengan masing-masing untuk loop:

## Gunakan keySet() untuk mengambil semua key yang ada di dalam HashMap dan kuncinya : capitalcities.keyset();

## Gunakan values() untuk mengambil semua nilai yang ada di dalam HashMap, kuncinya: capitalcities.values();

## isEmpty() untuk mengecek apakah HashMap dalam keadaan kosong atau tidak

## clone() untuk menggandakan objek HashMa

## Contohnya:

|  |
| --- |
| // Import the HashMap class  import java.util.HashMap;  public class MyClass {  public static void main(String[] args) {  // Create a HashMap object called people  HashMap<String, Integer> people = new HashMap<String, Integer>();  // Add keys and values (Name, Age)  people.put("John", 32);  people.put("Steve", 30);  people.put("Angie", 33);  for (String i : people.keySet()) {  System.out.println("Name: " + i + " Age: " + people.get(i));  }  }  } |

## Java Wrapper Classes

## Di dalam bahasa pemrograman Java, *Wrapper Class* (Kelas Pembungkus) adalah suatu mekanisme yang digunakan untuk mengconvert atau mengubah suatu nilai yang di definisikan, dari tipe data primitif menjadi sebuah nilai dengan tipe data referensi (Objek). Selain itu, Wrapper Class mendukung method dari tipe data primitif, contohnya seperti Boolean, Character, Integer, Long, Float, dsb.

Pada bahasa pemrograman Java, terdapat 2 jenis tipe data, yaitu primitif dan referensi (objek), Tipe Data *Primitif*adalah tipe data yang tidak memiliki method, hanya memiliki data saja. Tipe data ini bukanlah sebuah object, sedangkan Tipe Data *Referensi*adalah tipe data yang digunakan untuk memegang referensi dari sebuah object ( instance dari class).

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 |

metode berikut ini digunakan untuk mendapatkan nilai yang terkait dengan objek pembungkus yang sesuai:

intValue() , byteValue() , shortValue() , longValue() , floatValue() , doubleValue() , charValue() , booleanValue() .

metode lain yang bermanfaat adalah toString() metode yang digunakan untuk mengubah objek pembukus menjadi string.

Contohnya:

|  |
| --- |
| public class Wrapper\_O {  public static void main (String[]args){  Integer myInt = 5;  Double MyDouble = 5.4;  Character myChar = 'A';  System.out.println(myInt);  System.out.println(MyDouble);  System.out.println(myChar);    }    } |

1. **Java Exceptions**

Exceptions/Pengecualian (atau peristiwa yang luar biasa) adalah masalah yang muncul selama pelaksanaan program. Ketika pengecualian terjadi aliran normal program terganggu dan program/aplikasi berakhir abnormal, yang tidak disarankan, oleh karena itu, pengecualian ini harus ditangani.

Pengecualian dapat terjadi karena berbagai alasan. Berikut adalah beberapa skenario di mana pengecualian terjadi.

* Pengguna telah memasukkan data yang tidak valid.
* Berkas yang harus dibuka tidak ditemukan.
* Sambungan jaringan telah hilang di tengah komunikasi atau JVM telah kehabisan memori.

Beberapa pengecualian ini disebabkan oleh kesalahan pengguna, yang lain oleh programmer kesalahan, dan lain-lain dengan sumber daya fisik yang telah gagal dalam beberapa cara.

* + Berdasarkan jenisnya kesalahan dalam pemrograman terbagi menjadi 3, yaitu:
* **Syntax Error**

Adalah kesalahan yang disebabkan oleh kesalahan tata cara penulisan tanda baca, kesalahan pemakaian operator dan nilai. Kesalahan jenis ini akan dengan mudah dideteksi oleh compiler maupun interpreter.

* **Logical Error**

Adalah kesalahan yang disebabkan oleh kesalahan logika maupun model atau metode yang digunakan untuk pemrosesan data, sehingga menyebabkan informasi yang dihasilkan menjadi salah. Kesalahan ini tidak dapat dideteksi oleh compiler maupun interpreter, kesalahan ini disadari setelah melihat penyimpanan pada saat proses maupun hasil proses.

* **Runtime Error**

Adalah exception yang bisa saja tidak ditangani tanpa menyebabkan program berhenti kecuali jika propogasi exceptionnya samapai ke main, maka akan menyebabkan terminasi program secara subnormal. Checked exception adalah exception yang ditangani secara explicit didalam throws. Programmer harus mmembuat cath untuk menangani exception yang terjadi. Sedangkan error adalah kesalahan yang tidak unrecoverable exception artinya tidak bisa ditangani oleh catch.

Atau pengertian lain, Runtime error adalah kesalahan yang disebabkan oleh tidak tersedianya sumber daya atau kondisi yang normal bagi program untuk berjalan dengan baik, misalnya kekurangan memori computer, disk full, atau pintu drive tidak terkunci, dll.

* + Ada beberapa *keyword* penting dalam java :
* java try and catch

*Try* pernyataan memungkinkan anda untuk menentukan blok kode yang akan diujiuntuk kesalahan ketika sedang di jalankan.

*Catch* pernyataan memungkinkan anda untuk menentukan blok kode yang akan dieksekusi, jika terjadi kesalahan dalam blok try.

Kata kunci *try* dan *catch* berpasangan :

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

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

* Finally

*Finally* pernyataan memungkin kan anda mengeksekusi kode, setelah *try….catch* terlepas dari hasilnya:

|  |
| --- |
| try {  // Protected code  } catch (ExceptionType1 e1) {  // Catch block  } catch (ExceptionType2 e2) {  // Catch block  } catch (ExceptionType3 e3) {  // Catch block  }finally {  // The finally block always executes.  } |

* Throw keyword

Throw pernyataan memungknkan anda untuk membuat kesalah kustom.

Throw di gunakan bersama-sama dengan jenis pengecualian. Ada banyak jenis pengecualian tersedia di java : ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, dll.

Contohnya :

|  |
| --- |
| public class MyClass {  static void checkAge(int age) {  if (age < 18) {  throw new ArithmeticException("Access denied - You must be at least 18 years old.");  } else {  System.out.println("Access granted - You are old enough!");  }  }    public static void main(String[] args) {  checkAge(20);  }  } |

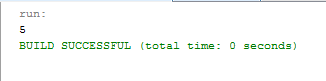
**BAB III**

**HASIL PERCOBAAN PRAKTIKUM**

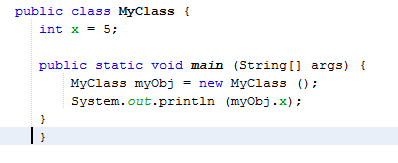
1. **Java Classes and Object**

* **Membuat Object**

Output dari proses running:

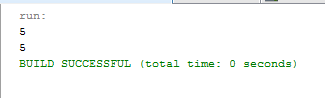


Source code : code program pada java classes and object untuk membuat object

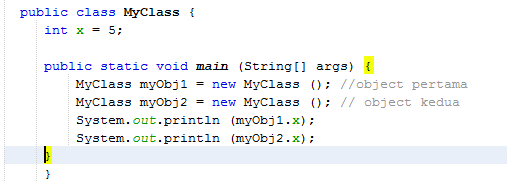


* **Membuat Banyak Object**

Output dari proses running:

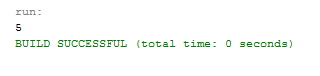


Source code : code program java classes and object untuk membuat banyak ibject dari satu kelas



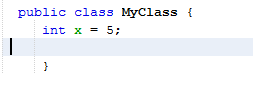
* **Membuat Banyak Kelas**

Output dari hasil kompilasi dua class untuk satu object

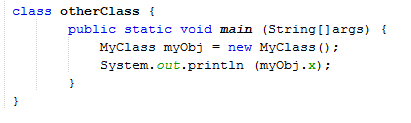


Source code : code membuat object dari banyak class

1. MyClass.java



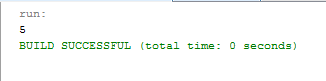
1. otherClass.java



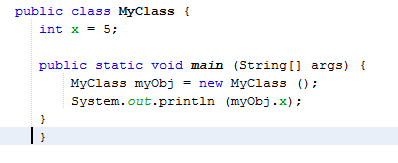
1. **Java Class Atributes**

* **Accessing Attributes**

Output dari proses running:



Source code : code program pada java class attributes untuk mengakses atribut

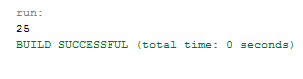


* **Modify Attributes**

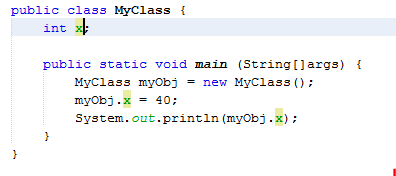
Output : nilai x = 40



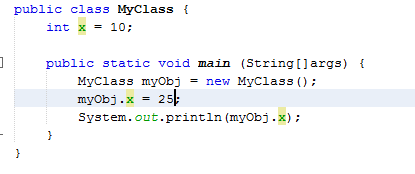
Output: nilai x = 25



Source code : code program java class attributes untuk mengubah nilai x menjadi 40

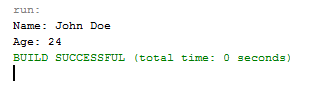


Source code : code program java class attributes untuk megubah nilai x menjadi 25

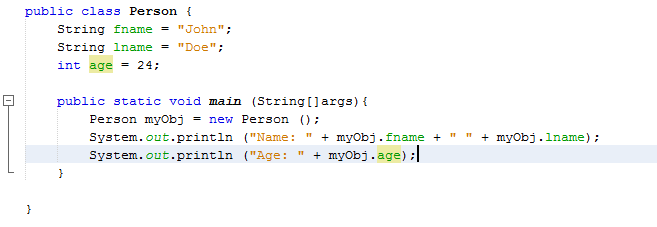


* **Multiple Objects**

Output:



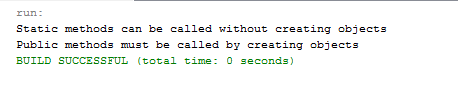
Source code: code program java attribute untuk memperbanyak attribute yang diiginkan

* 

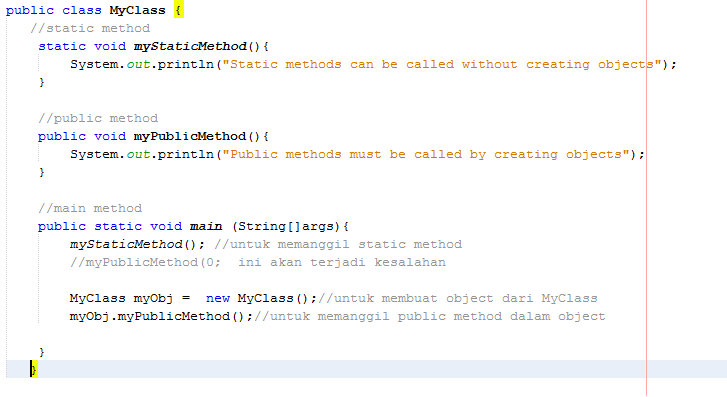
1. **Java Class Methods**

* **Static Vs Non-Static**

Output dari proses running:

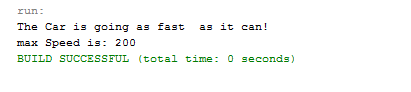


Source code: code program java pada java class methods untuk menunjukkan perbedaan static dan non-static

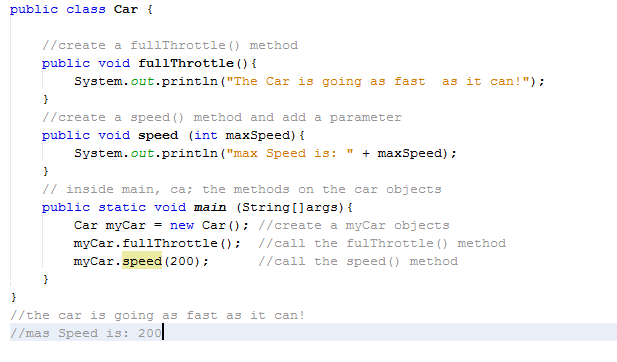


* **Access Method With Object**

Output dari proses running:



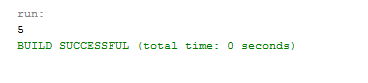
Source code: code program java pada java class methods untuk mengakses sebuah method dengan object



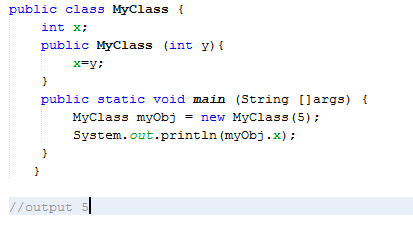
1. **Java Constructors**

* **Constructors Parameters**

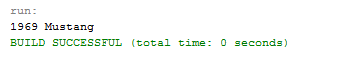
Output dari proses running:



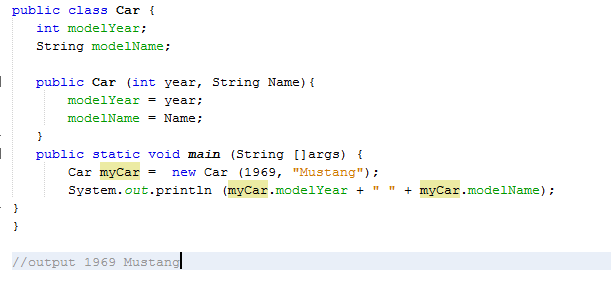
Source code: code program java pada java constructor untuk parameter dengan satu constructor



Output dari proses running:



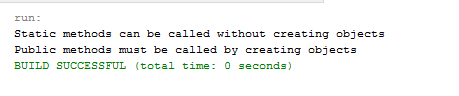
Source code: code program java pada java constructor untuk parameter dengan banyak constructor



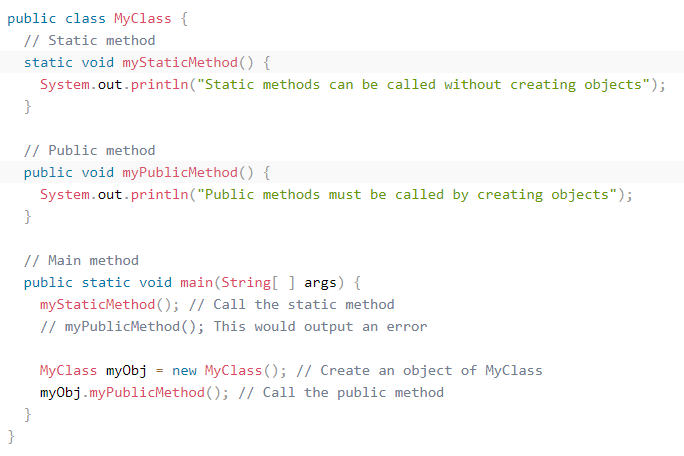
1. **Java Modifiers**

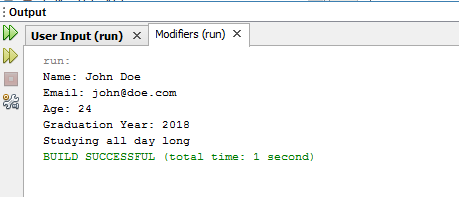
* **Static**

Output dari proses running:



Source code: code program java pada java modifiers untuk menunjukkan perbedaan antara method static dan public

* **Abstract**

Output dari proses running:

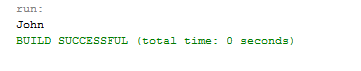
Source code: code program java pada java modifiers untuk sebuah abstract

|  |
| --- |
| abstract class Person {  public String fname = "John";  public int age = 24;  public abstract void study(); // abstract method  }  // Subclass (inherit from Person)  class Student extends Person {  public int graduationYear = 2018;  public void study() { // the body of the abstract method is provided here  System.out.println("Studying all day long");  }  }  // End code from filename: Person.java  // Code from filename: MyClass.java  class MyClass {  public static void main(String[] args) {  // create an object of the Student class (which inherits attributes and methods from Person)  Student myObj = new Student();  System.out.println("Name: " + myObj.fname);  System.out.println("Age: " + myObj.age);  System.out.println("Graduation Year:"+ myObj.graduationYear);  myObj.study(); // call abstract method  } |

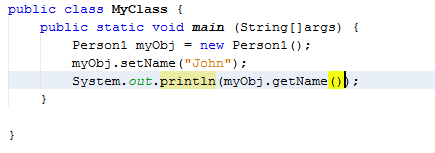
1. **Java Encapsulation**

* **Get and Set**

Output dari proses running:



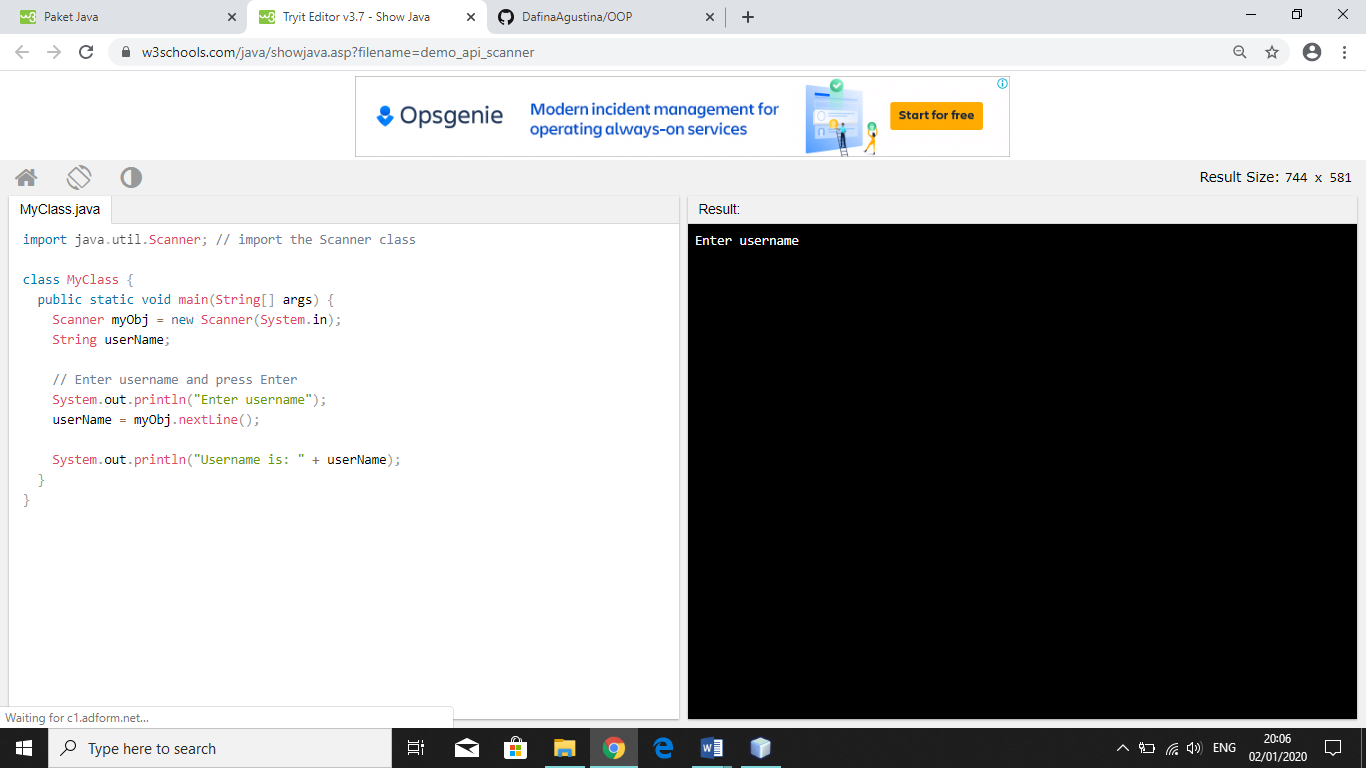
Source code: code program java pada java encapsulation untuk mengambil object dengan menggunakan method Get and Set



1. **Java Package/API**

* **Class Scanner**

Output dari proses running:



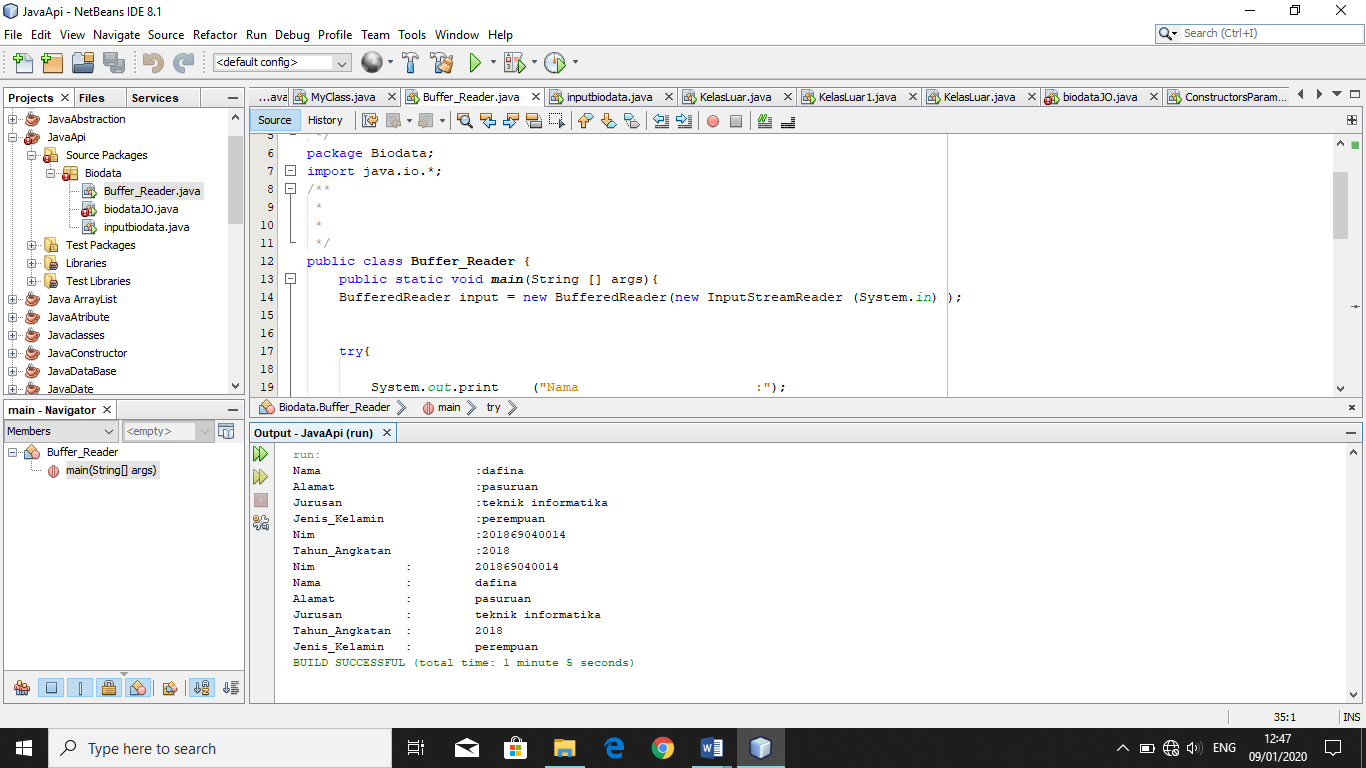
Yaitu masukkan username secara manual , lalu tekan enter maka akan mendapatkan hasilnya dari script di atas.

Source code: code program java pada java Package/API untuk class scanner

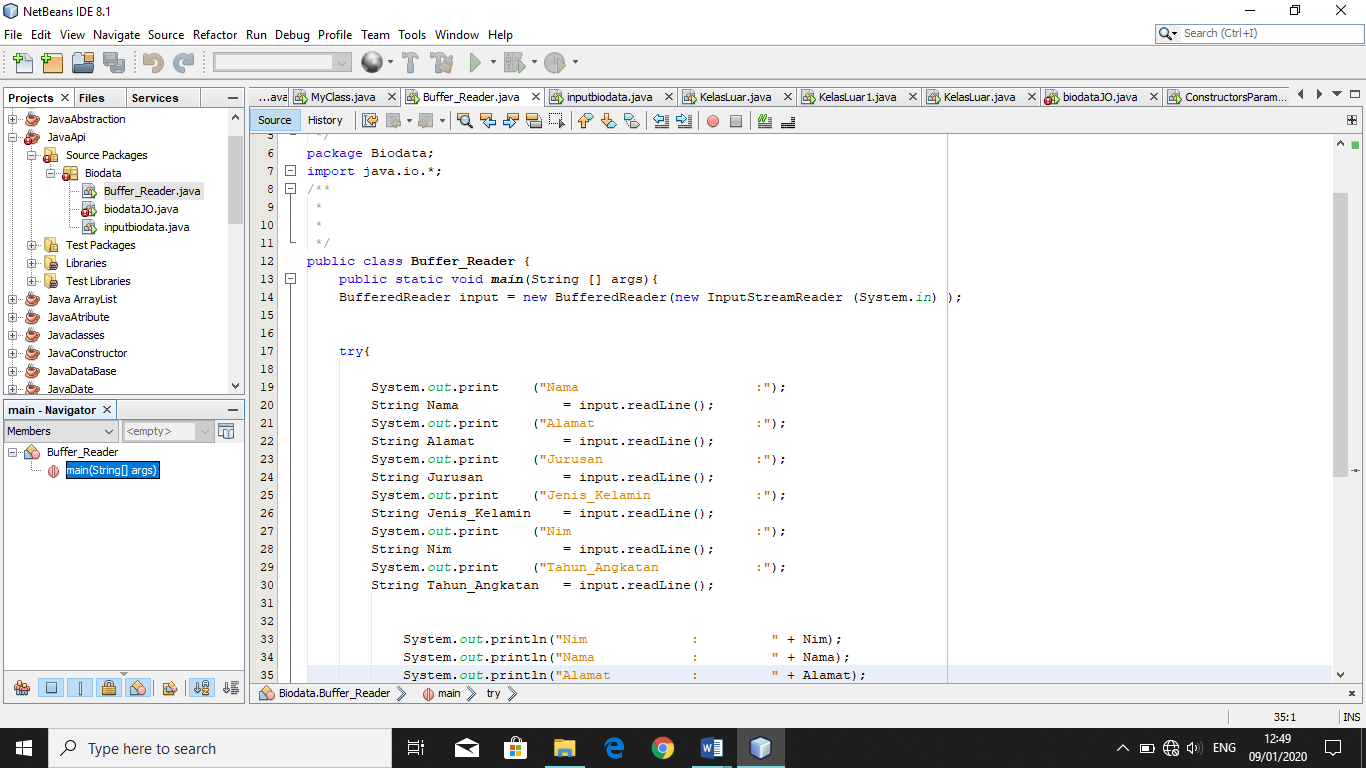
|  |
| --- |
| import java.util.Scanner; // import the Scanner class  class MyClass {  public static void main(String[] args) {  Scanner myObj = new Scanner(System.in);  String userName;    // Enter username and press Enter  System.out.println("Enter username");  userName = myObj.nextLine();    System.out.println("Username is: " + userName);  }  } |

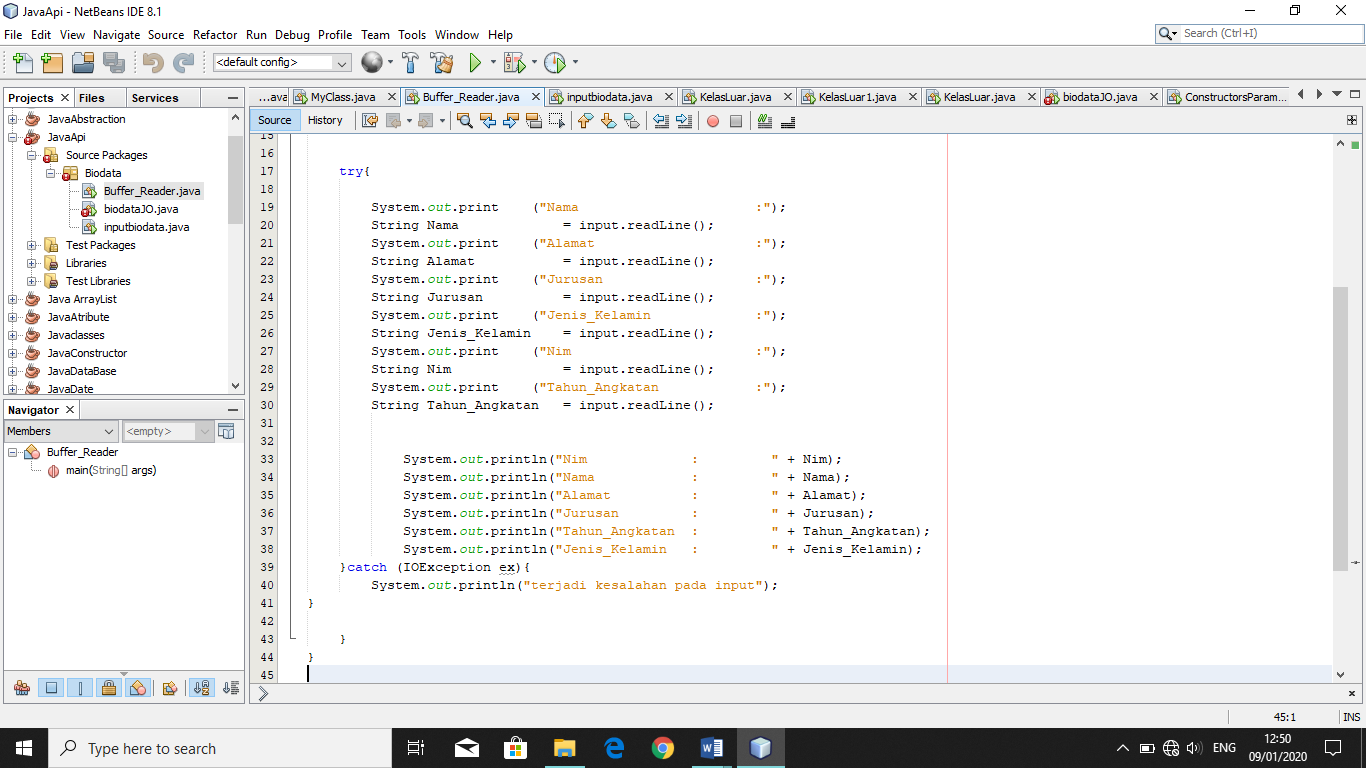
* **Buffer Reader**

Output dari proses running:



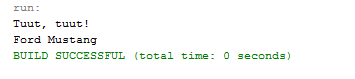
Source code: code program java pada java package/API untuk class buffer reader



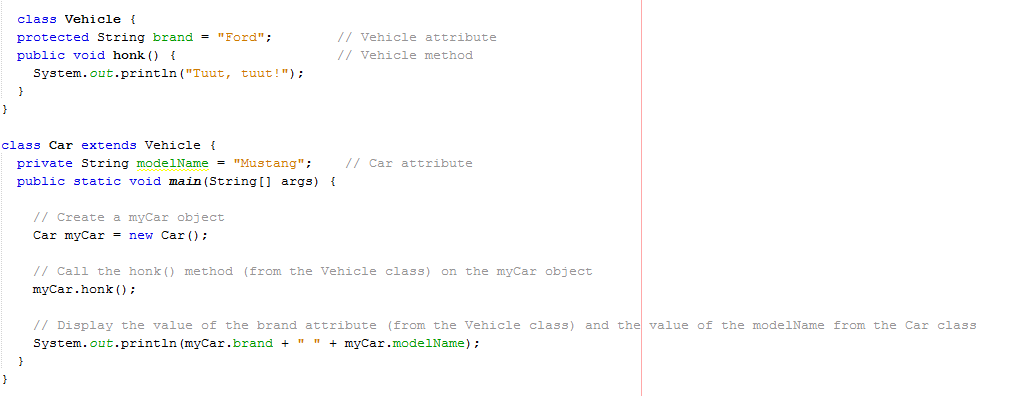


1. **Java Inheritance**

Output dari proses running:

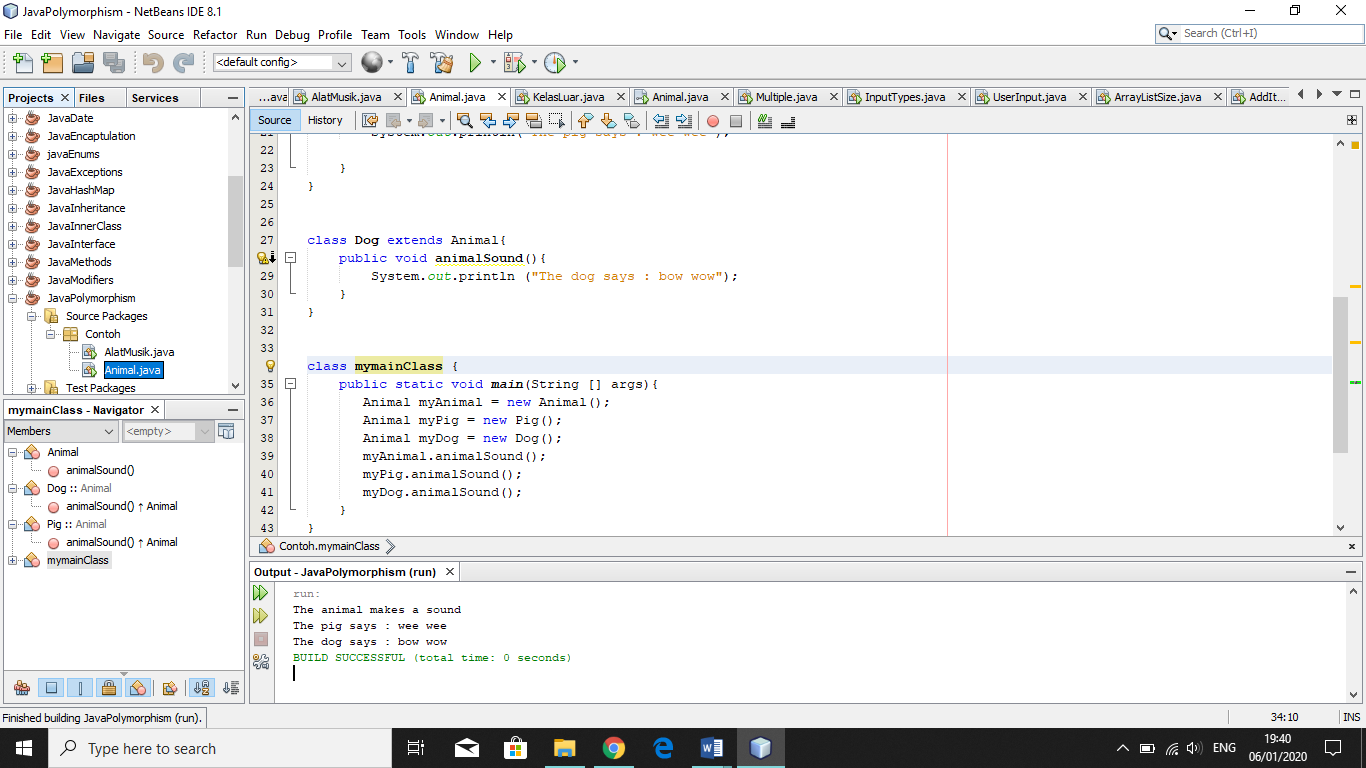


Source code: code program java pada java inheritance untuk kelas car yang mewarisi atribut dan method dari kelas vehicle

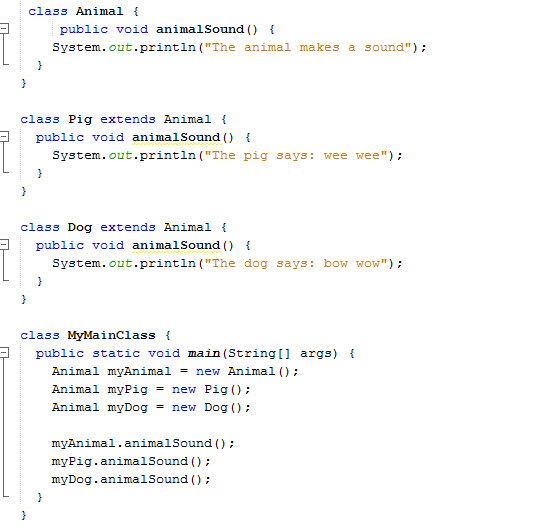


1. **Java Polymorphism**

Output dari proses running:



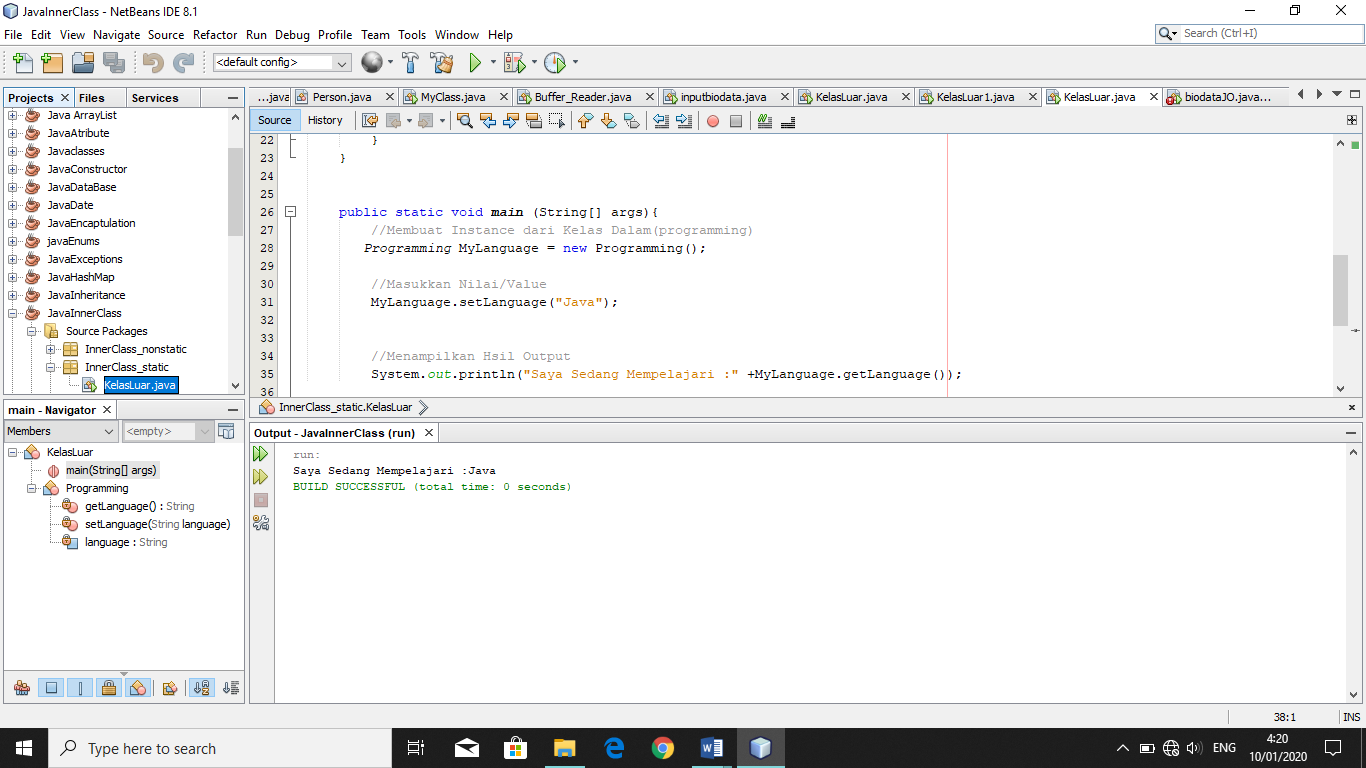
Source code: code progam java pada java polymorphism yang membuat object pig and dog dengan memanggil method animalSound() pada keduanya



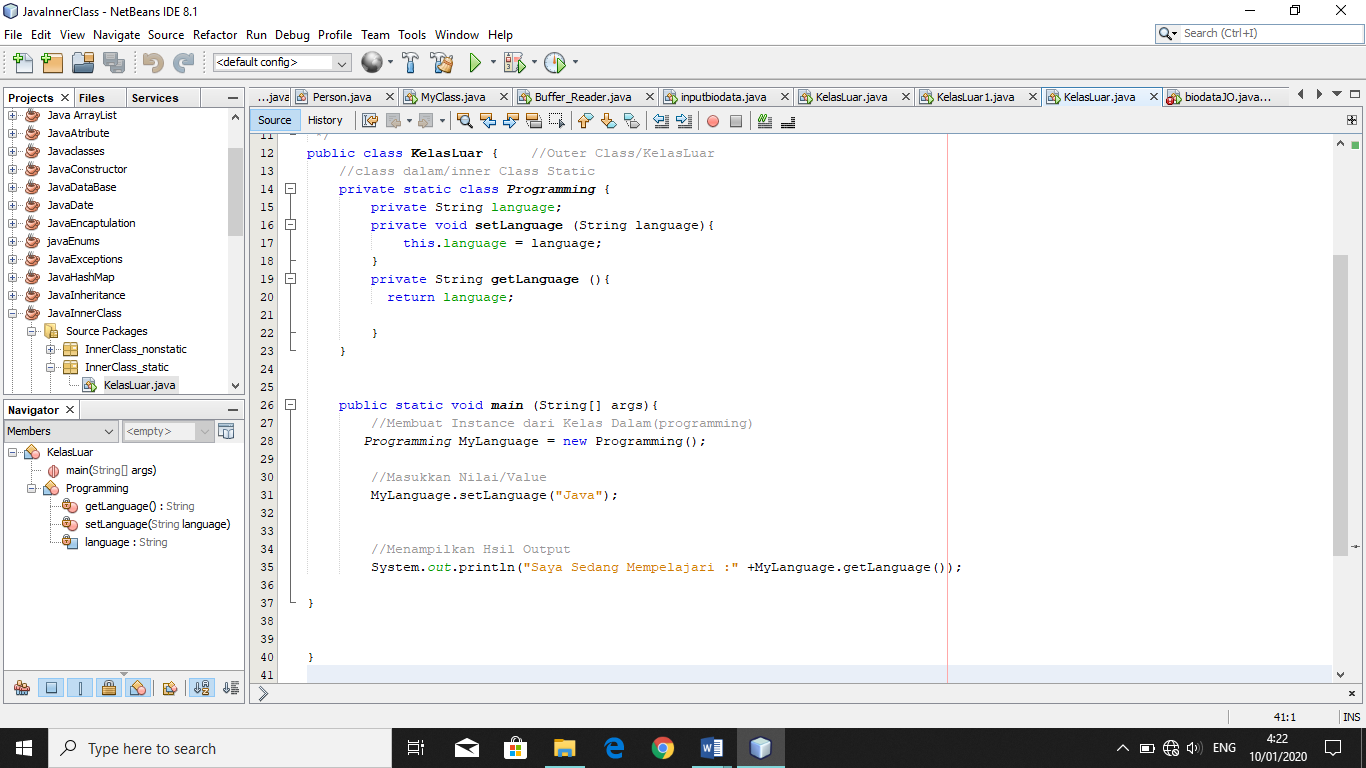
1. **Java Inner Class**

* **Inner Class Static**

Output dari proses running :

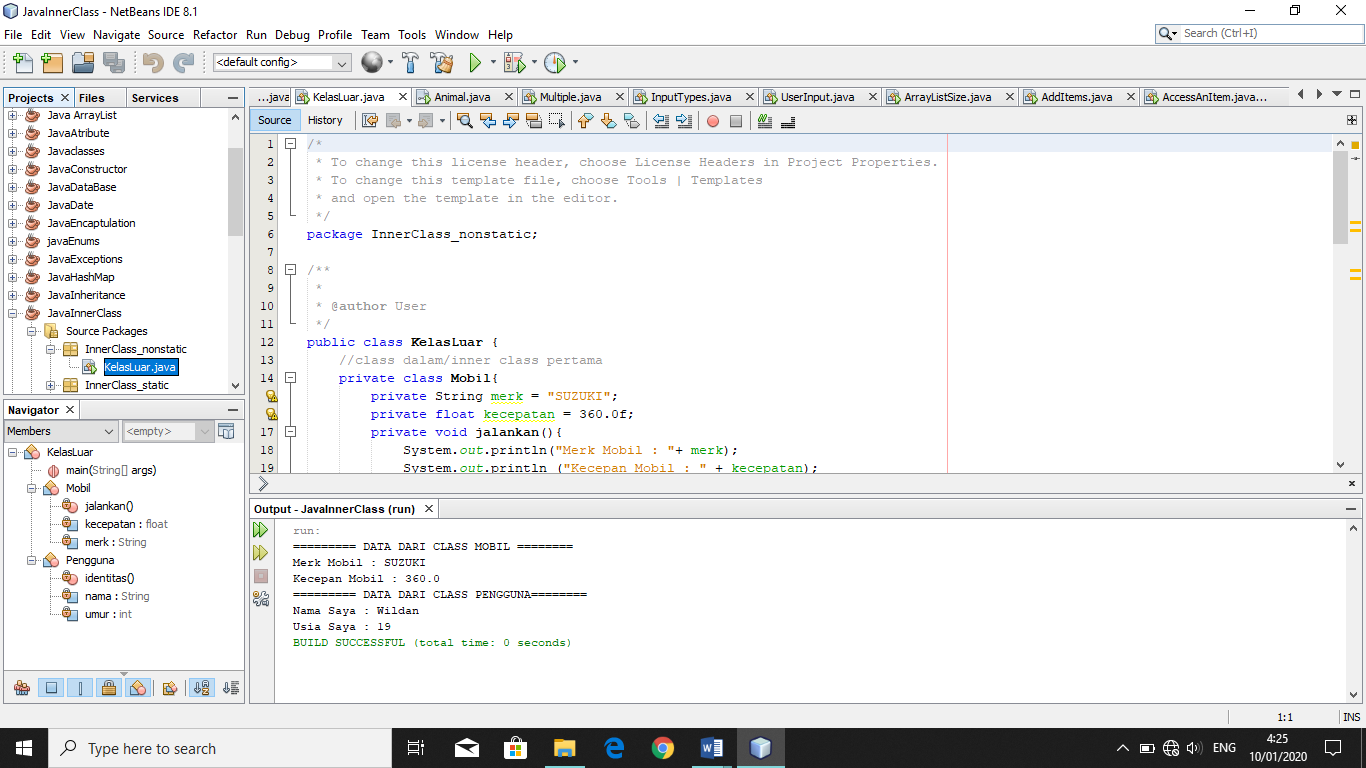


Source code: code program java pada jaa inner class untuk inner class static

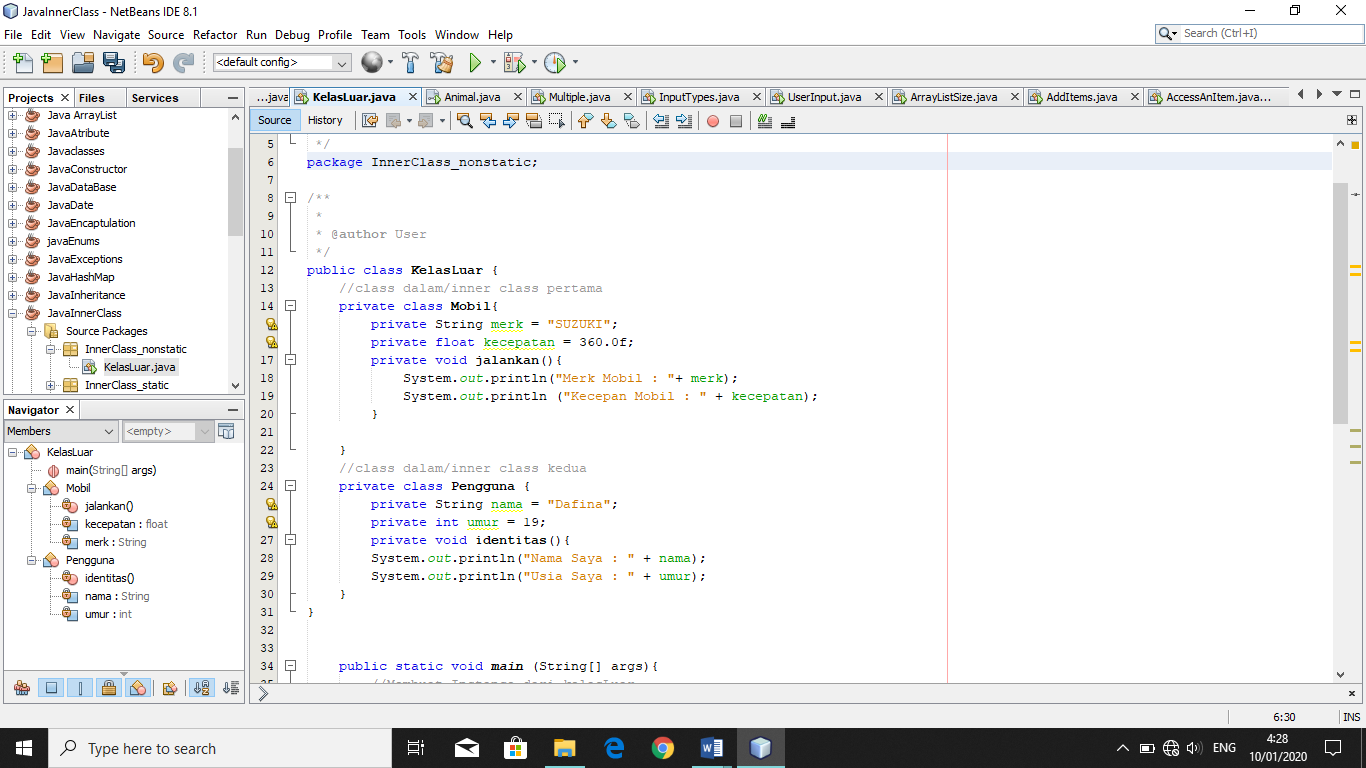


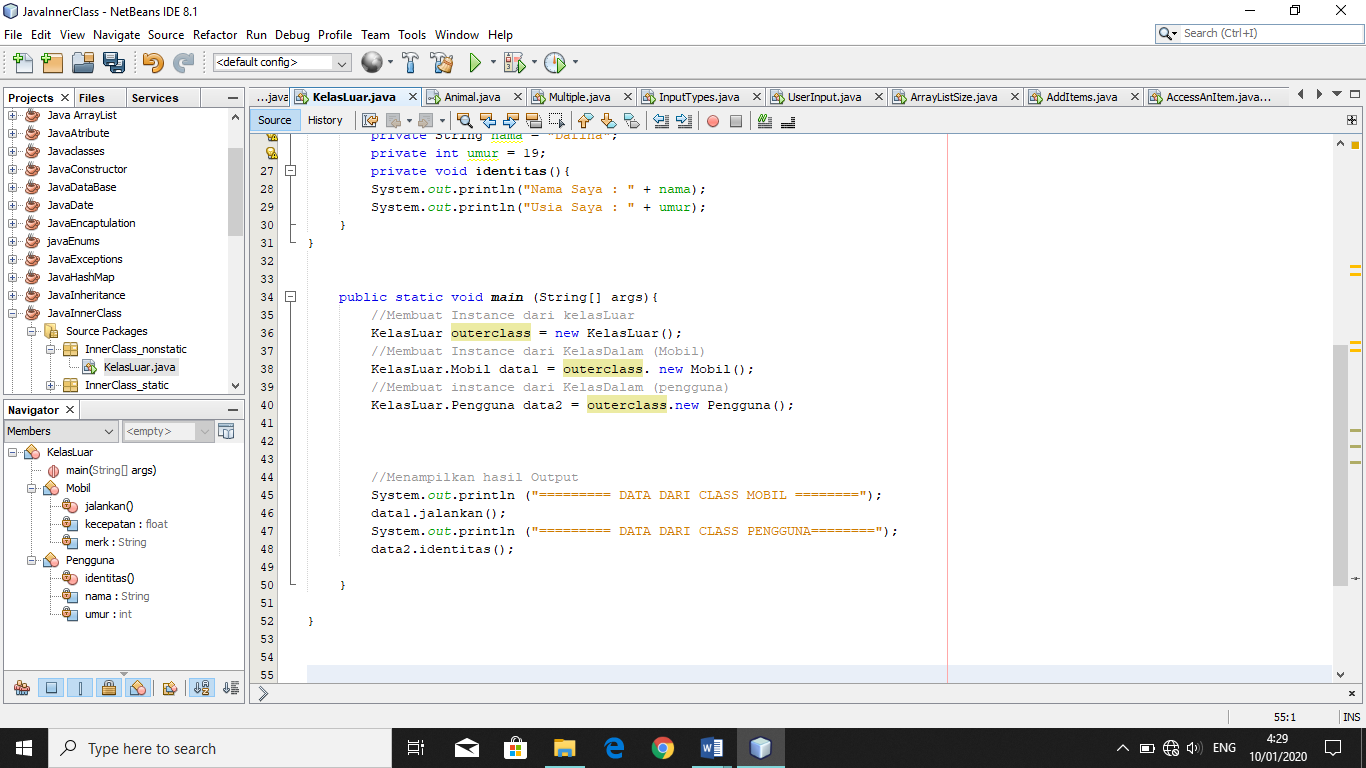
* **Inner Class Non-Static**

Output dari proses running



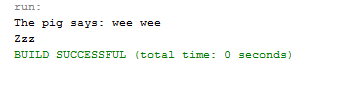
Source code: code program java pada java inner class untuk inner class non-static



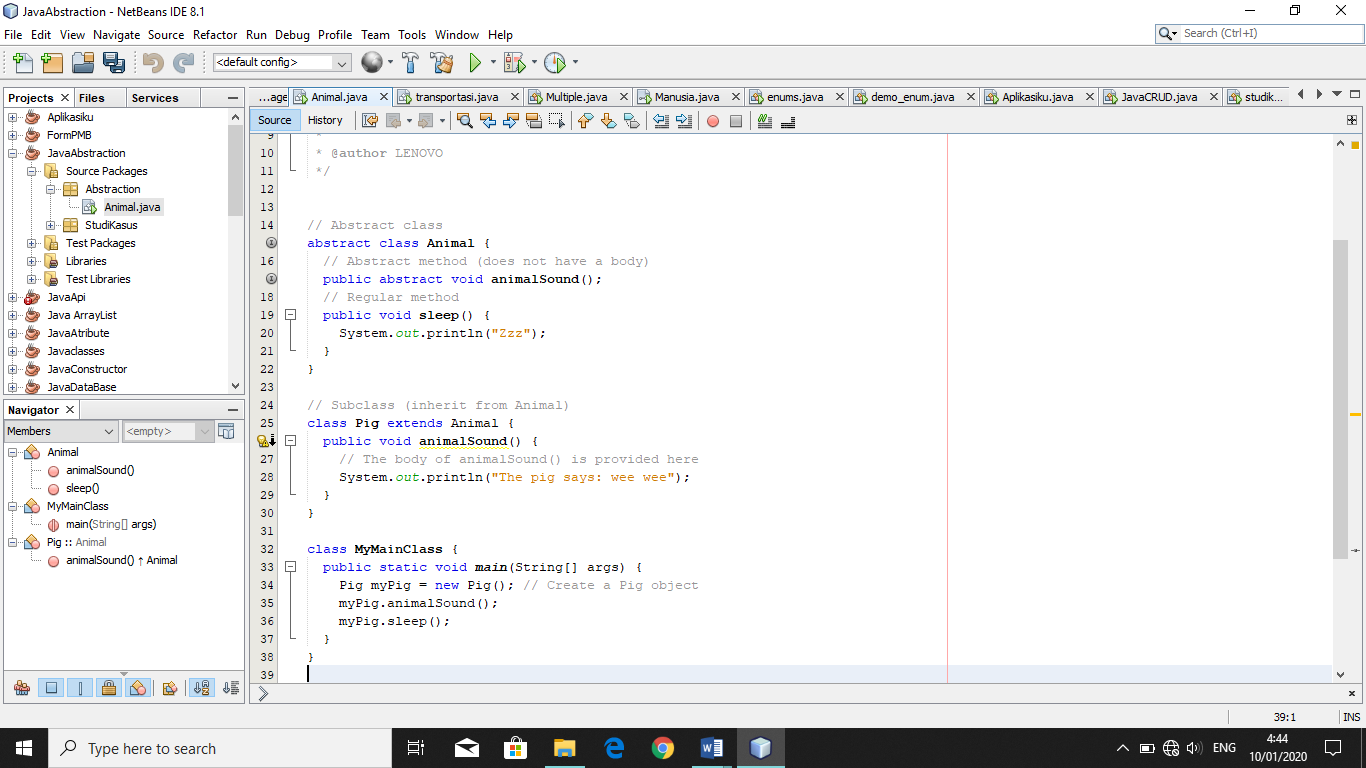


1. **Java Abstraksi**

Output dari proses running:



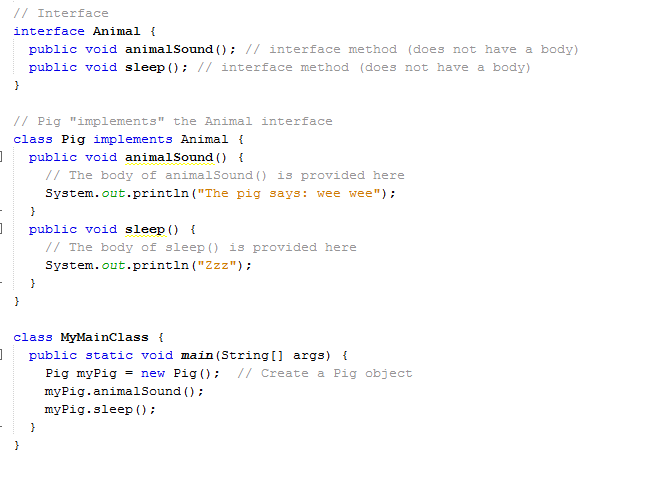
Source code: code program java pada java abstraksi untuk contoh suara binatang



1. **Java Interface**

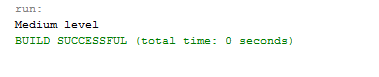
Output dari proses running:

Source code: code program java pada java interface dalam contoh suara binatang

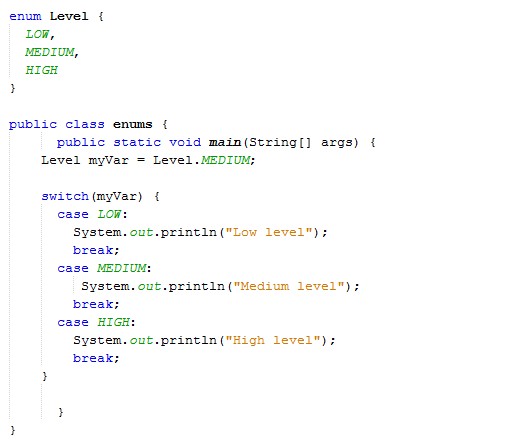


1. **Java Enum**

Output dari proses running:

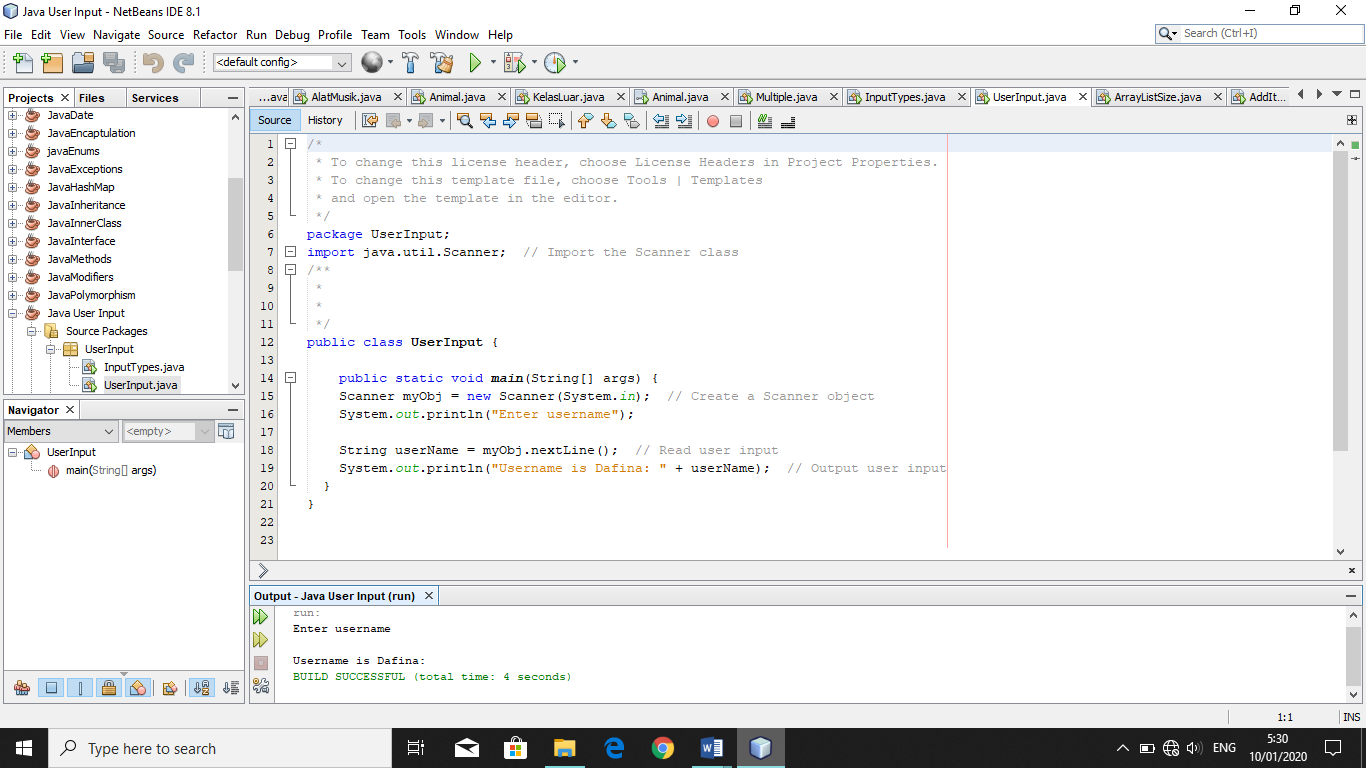


Source code: code program java pada percobaan java enum

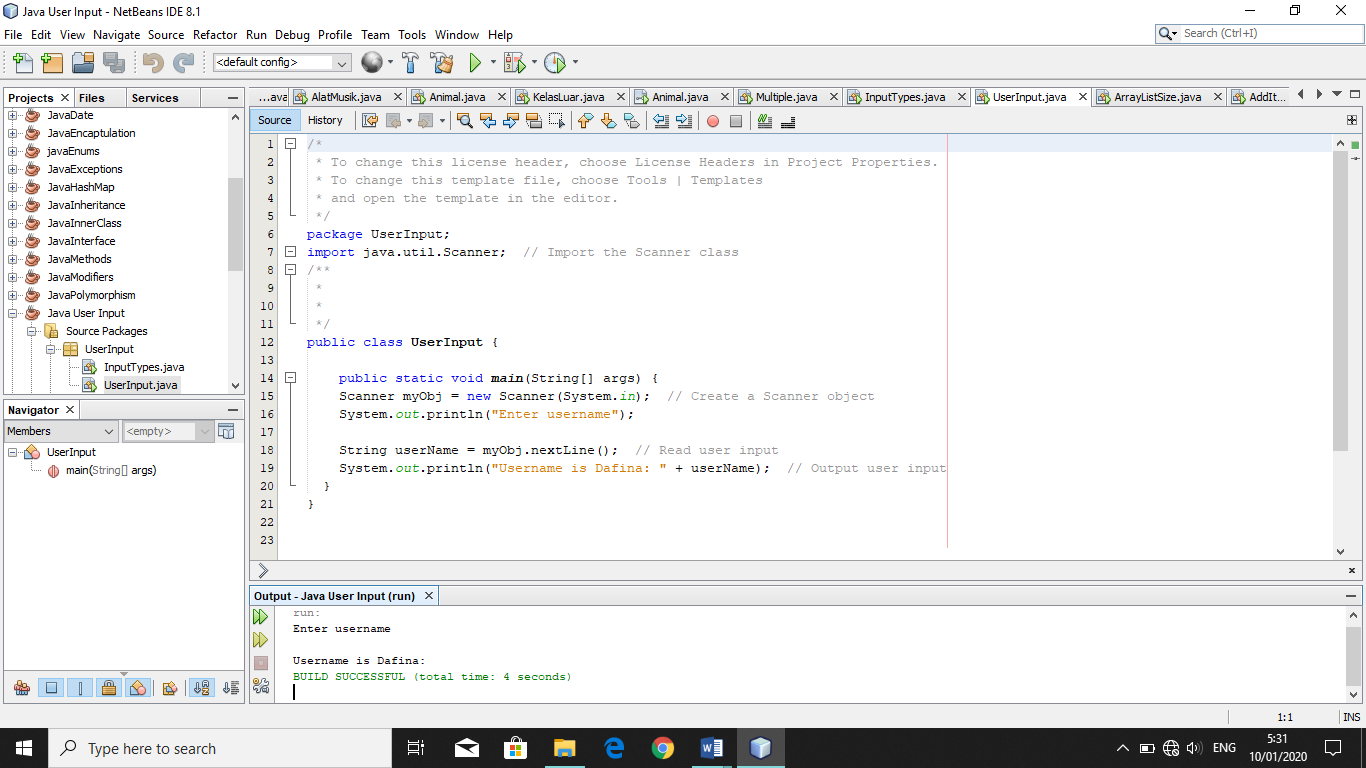


1. **Java User Input**

Output dari proses running:



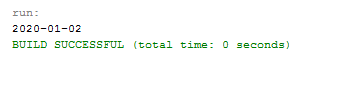
Source code code program java pada java user input dengan menggunakan class scanner



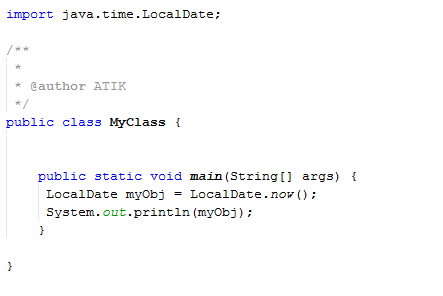
1. **Java Date**

* **Menampilkan Tanggal Saat ini**

Output dari proses running:

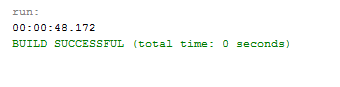


Source code: code program java pada java date untuk menampilkan tanggal saat ini

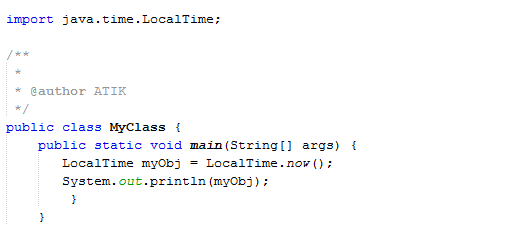


* **Menampilkan Waktu Saat Ini**

Output dari proses running:

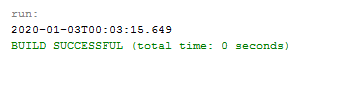


Source code: code program java pada java date untuk menampilkan waktu saat ini

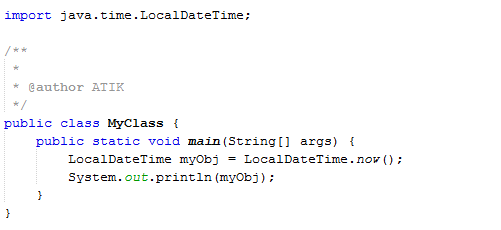


* **Menampilkan Waktu dan Tanggal Saat Ini**

Output dari proses running:



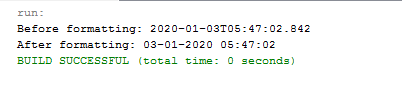
Source code: code program java pada java date untuk menampilkan waktu dan tanggal saat ini



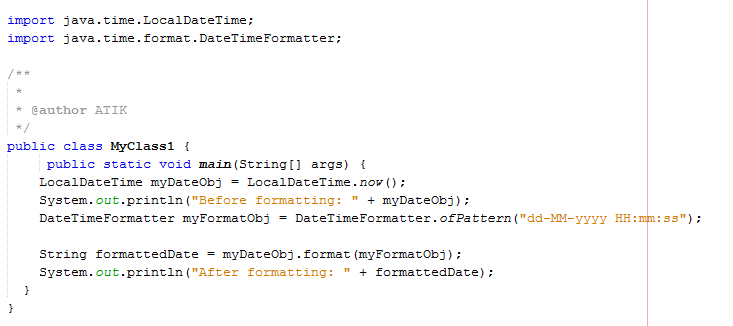
* **Memformat Tanggal dan Waktu**

1. Value: yyy – MM – dd

Output dari proses running:

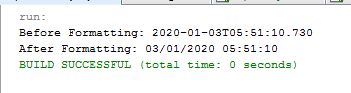


Source code: code program java pada java date untuk memformat tangggal dan waktu dengan format yyyy – MM – dd

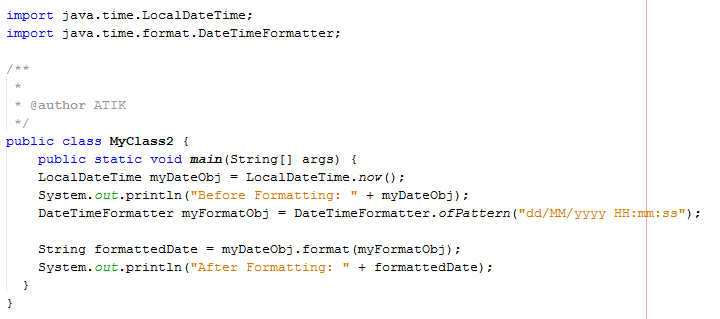


1. Value: dd/MM/yyy

Output dari proses running:

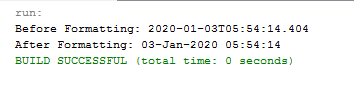


Source code: code program java pada java date untuk memformat tanggal dan waktu dengan format dd/MM/yyyy

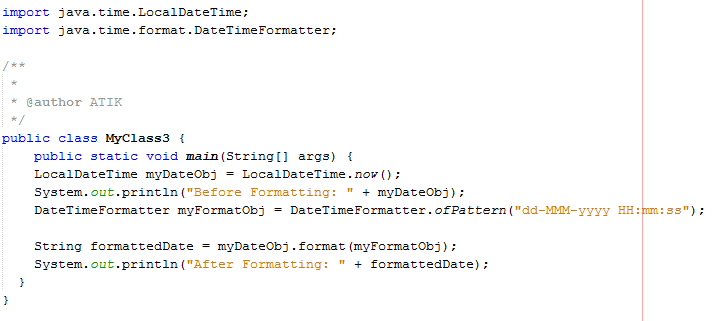


1. Value: dd – MMM – yyyy

Output dari prose running:

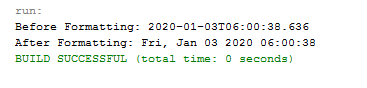


Source code: code program java pada java date untuk memformat tanggal dan waktu dengan format dd – MM – yyyy

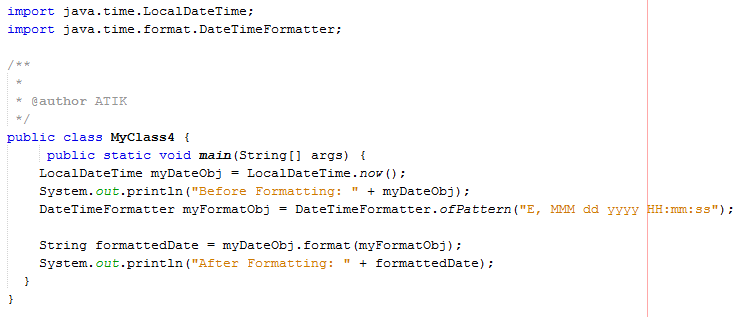


1. Value: E, MMM dd yyyy

Output dari proses running:



Source code : code program java pada java date untuk memformat tanggal dan waktu saat ini dengan format E, MMM dd yyyy.



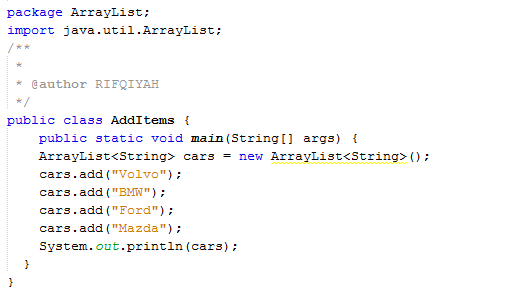
1. **Java Array List**

* **Menambahkan Item**

Output dari proses running:

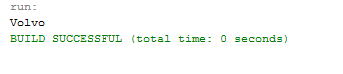


Source code: code program java pada java arraylist untuk menambahkan item



* **Mengakses Item**

Output dari proses running:

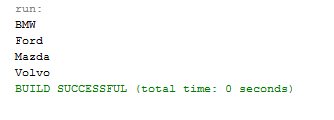


Source code: code program java pada java arraylist untuk mengakses sebuah item

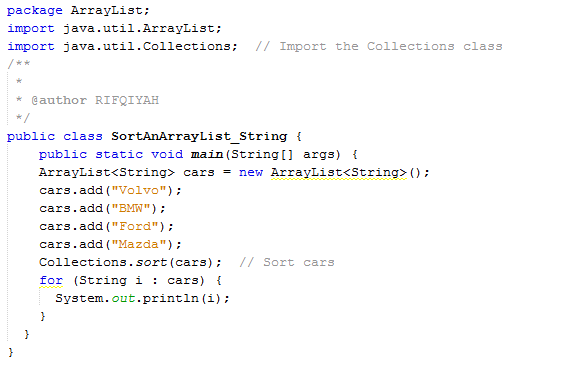


* **Mensortir Daftar Array**

Output dari proses running:



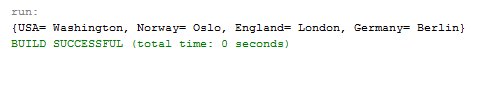
Source code: code program java pada java arraylist untuk mensortir daftar array dengan tipe data string.



1. **Java HashMap**

* **Menambahkan Item**

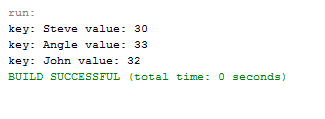
Output dari proses running:



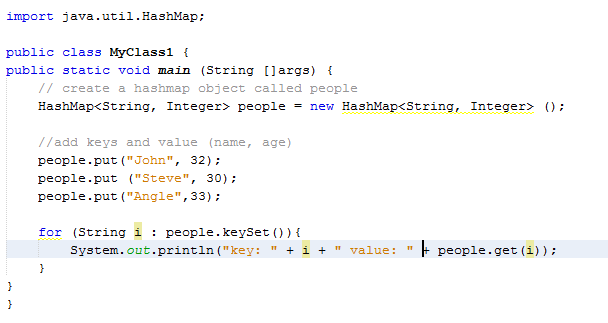
Source code: code program java pada java HashMap untuk menambahkan item dengan kata kunci string dan nilai string



Output dari proses running:



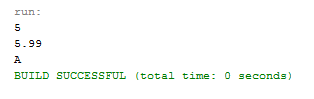
Source code: code program java pada java HashMap untuuk menambahkan item dengan kunci String, dan nilai Integer



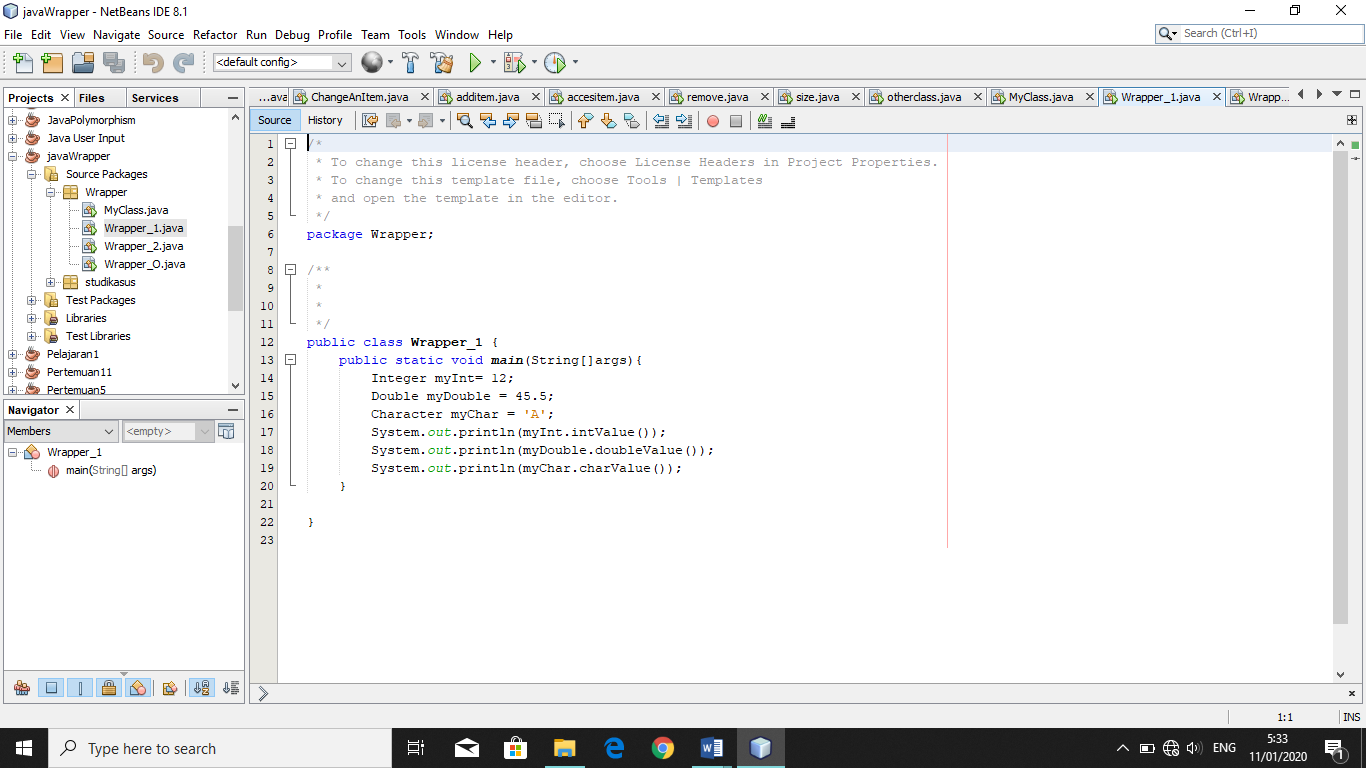
1. **Java Wrapper**

* **Membuat Objek**

Output dari proses running:

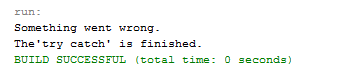


Source code: code program java pada java wrapper class untuk membuat objek

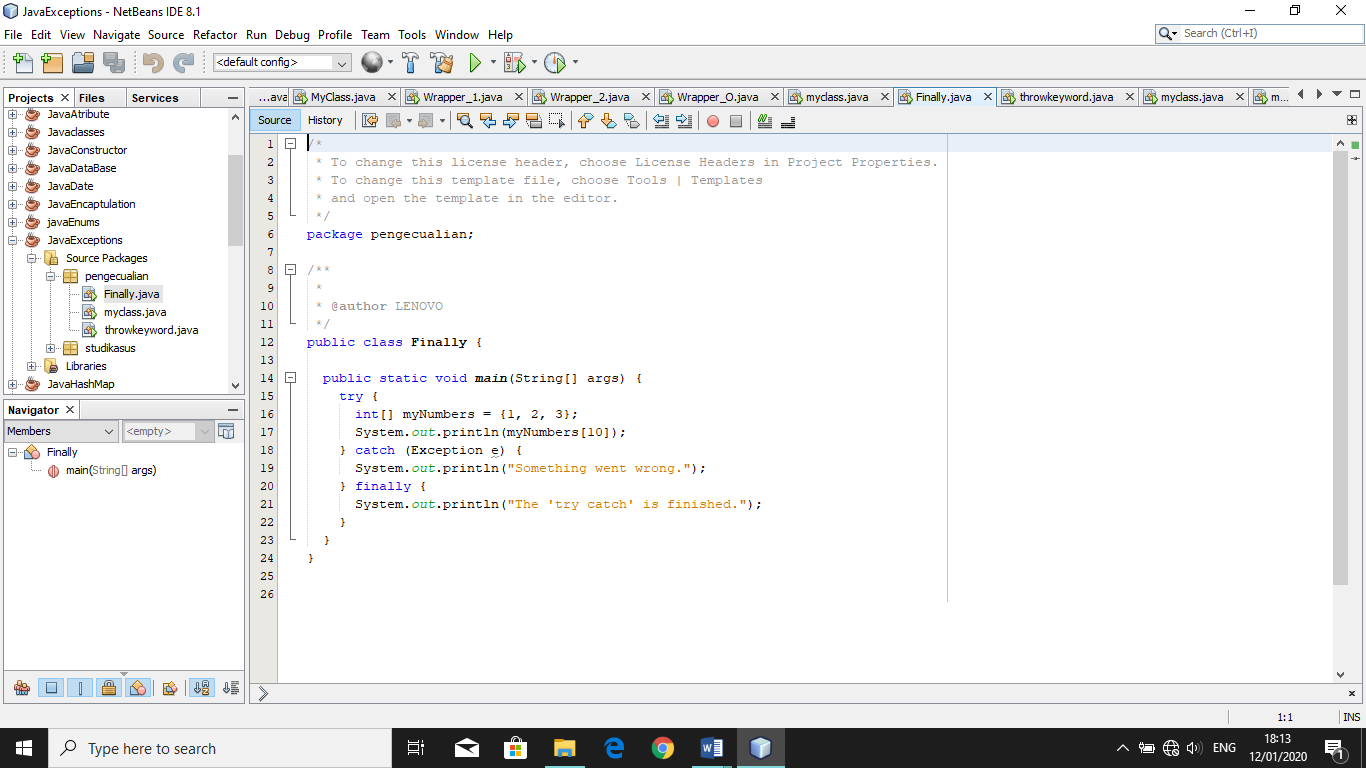


1. **Java Exceptions**

Output dari proses running:



Source code: code program java pada java exceptions dengan menggunaan kata kunci try and catch

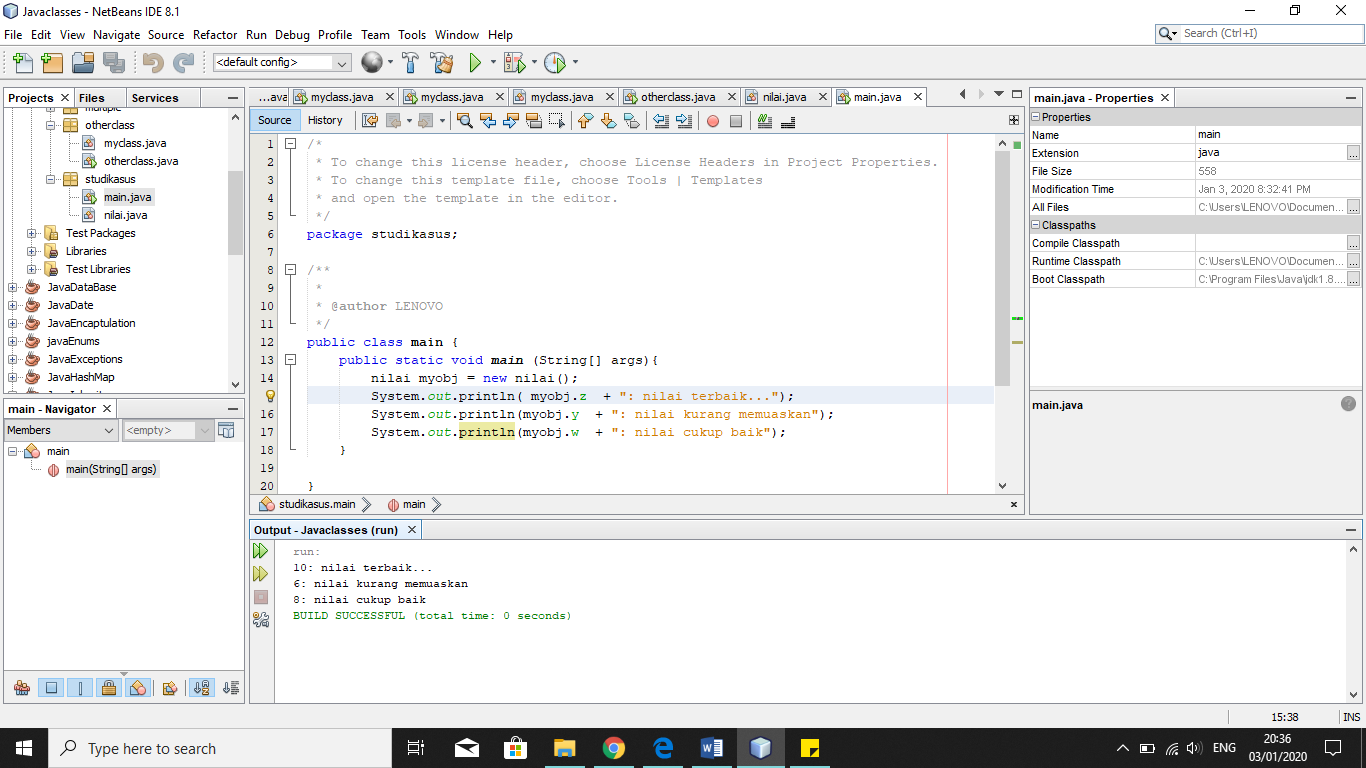


**BAB IV**

**STUDI KASUS**

* + 1. **Java Classes/Object**

Terdapat beberapa nilai pada suatu java class yaitu nilai dari z = 10, nilai dari y = 6,dan nilai dari w = 8.



Yaitu terdapat dua kelas, kelas yang pertama yaitu terdapat isi nilainya saja atau nilai yang akan di jalankan , dan kelas yang kedua yaitu untuk memanggil nilai yang sudah diterapkan.

***Kode inputannya :***

package studikasus;

/\*\*

\*

\*

\*/

public class nilai {

int z = 10;

int y = 6;

int w = 8;

}

int() berfungsi mengkonversi bilangan maupun string angka menjadi bilangan bulat (**integer**).

package studikasus;

/\*\*

\*

\*

\*/

public class main {

public static void main (String[] args){

nilai myobj = new nilai();

System.out.println( myobj.z + ": nilai terbaik...");

System.out.println(myobj.y + ": nilai kurang memuaskan");

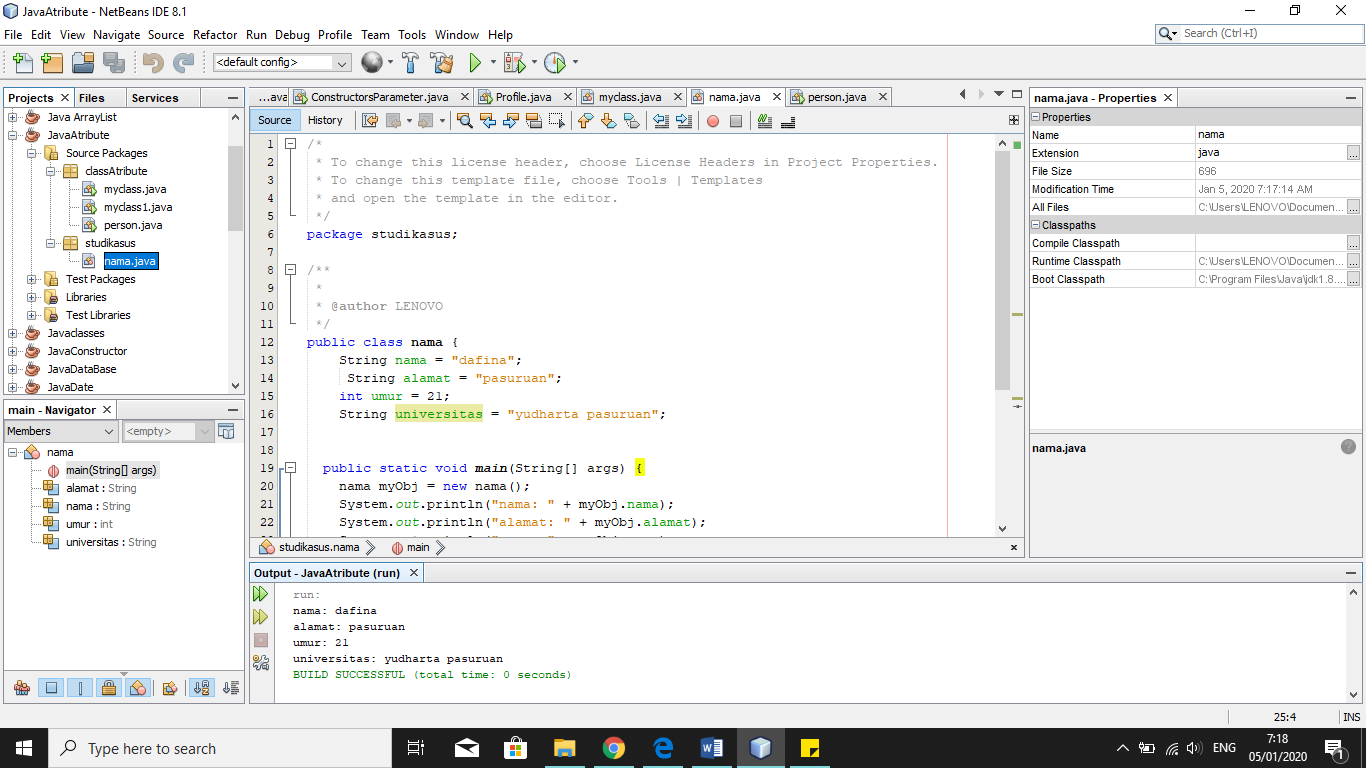
System.out.println(myobj.w + ": nilai cukup baik");

}

}

* + 1. **Java Class Atribute**

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



***Kode inputannya :***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class nama {

String nama = "dafina";

String alamat = "pasuruan";

int umur = 21;

String universitas = "yudharta pasuruan";

public static void main(String[] args) {

nama myObj = new nama();

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

System.out.println("alamat: " + myObj.alamat);

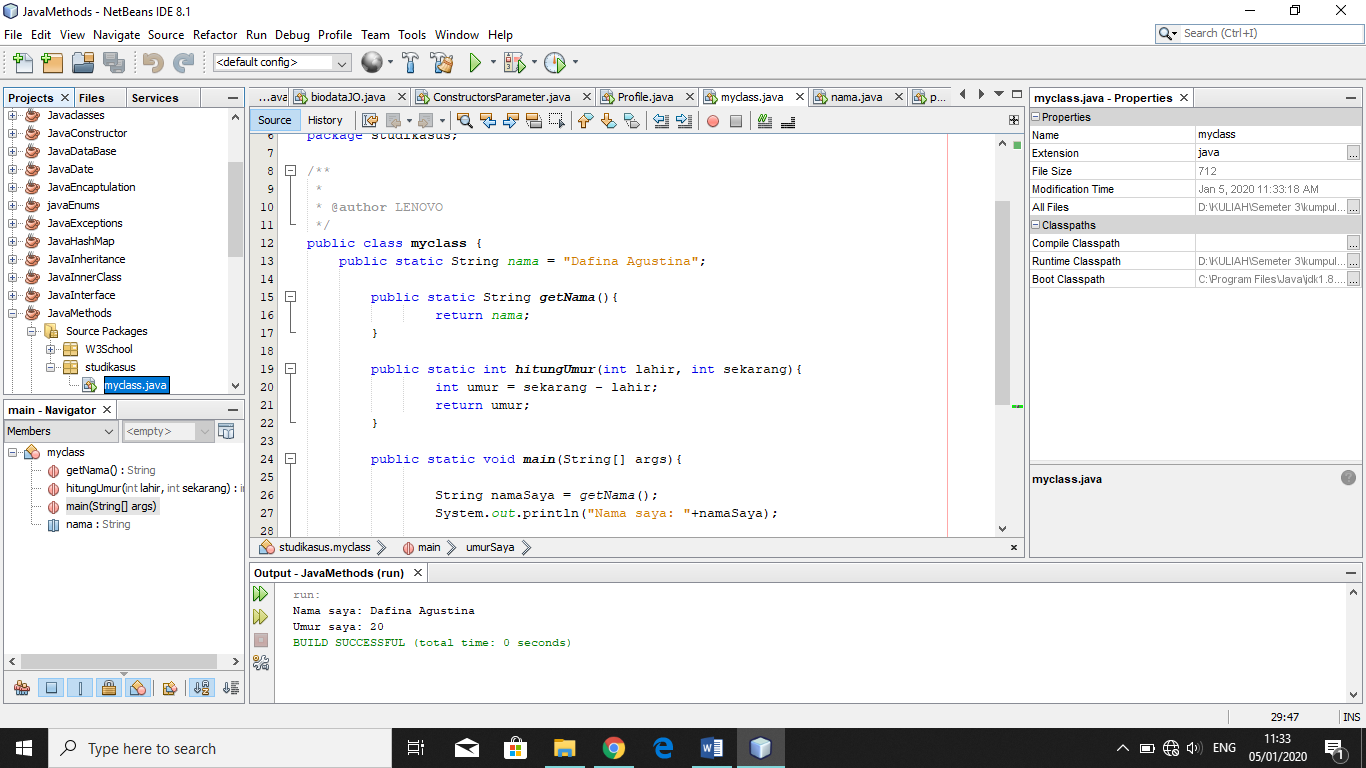
System.out.println("umur: " + myObj.umur);

System.out.println("universitas: " + myObj.universitas);

}

}

* + 1. **Java Class Method**



***Kode inputannya :***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class myclass {

public static String nama = "Dafina Agustina";

public static String getNama(){

return nama;

}

public static int hitungUmur(int lahir, int sekarang){

int umur = sekarang - lahir;

return umur;

}

public static void main(String[] args){

String namaSaya = getNama();

System.out.println("Nama saya: "+namaSaya);

int umurSaya = hitungUmur(1999, 2019);

System.out.println("Umur saya: "+umurSaya);

}

}

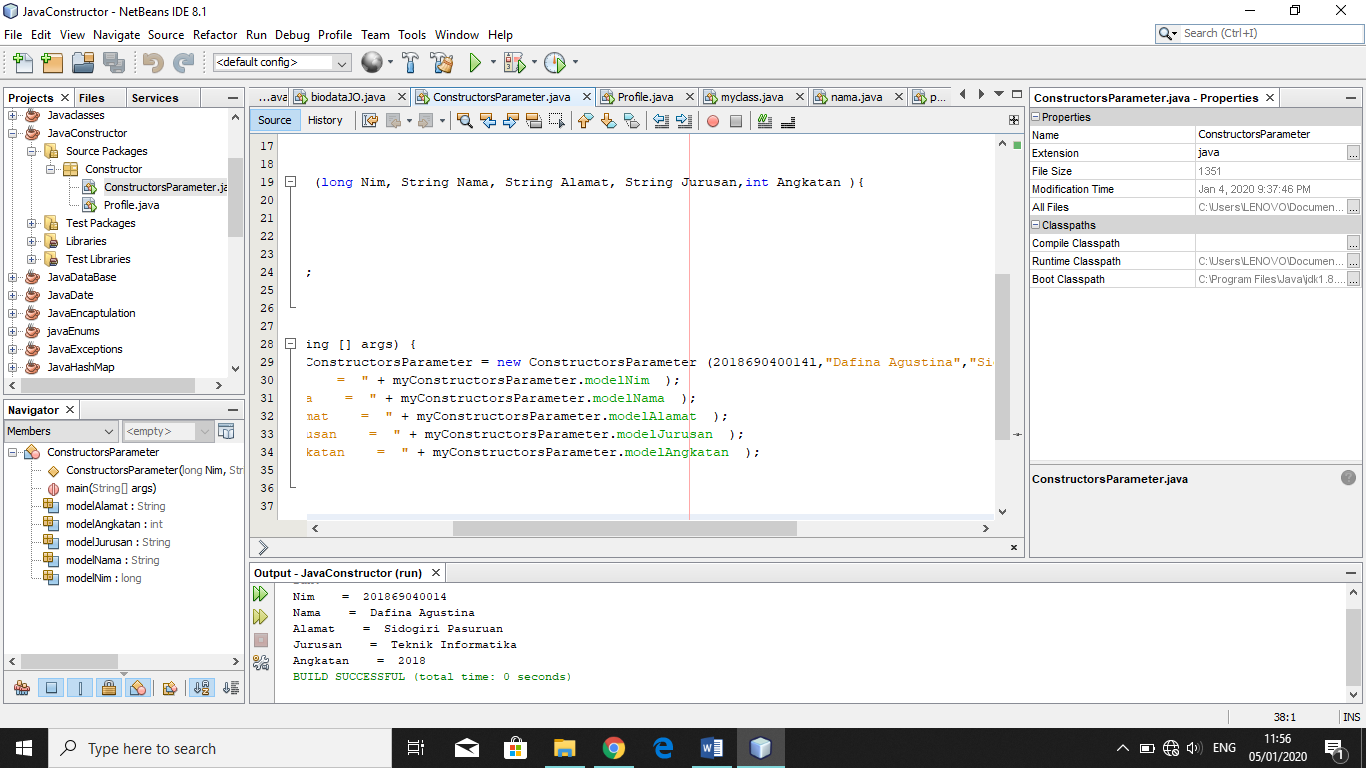
Keterangan:

Jika kita perhatikan dalam program diatas kita menggunakan keyword return untuk tiap method. Namun tipe data untuk tiap method berbeda sesuai kebutuhan kita. fungsi return adalah untuk mengembalikan nilai kedalam fungsi itu sendiri. Intinya nilai yang dikembalikan akan disimpan dalam nama method. Inilah sebabnya kita dapat mencetak atau memindakan nilai dari method return kedalam variabel. Contohnya kita memindahkan nilai getNama() kedalam variabel namaSaya.

Dalam program diatas kita membuat kelas bernama MethodReturn dimana kita telah mendeklarasikan variabel nama dan method getNama() juga method hitungUmur()

* + 1. **Java Constructors**

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



***Kode inputannya :***

package Constructor;

/\*\*

\*

\* @author LENOVO

\*/

public class ConstructorsParameter {

long modelNim ;

String modelNama;

String modelAlamat;

String modelJurusan;

int modelAngkatan;

public ConstructorsParameter (long Nim, String Nama, String Alamat, String Jurusan,int Angkatan ){

modelNim = Nim;

modelNama = Nama;

modelAlamat = Alamat;

modelJurusan = Jurusan;

modelAngkatan = Angkatan;

}

public static void main (String [] args) {

ConstructorsParameter myConstructorsParameter = new ConstructorsParameter (201869040014l,"Dafina Agustina","Sidogiri Pasuruan","Teknik Informatika",2018);

System.out.println ("Nim = " + myConstructorsParameter.modelNim );

System.out.println ("Nama = " + myConstructorsParameter.modelNama );

System.out.println ("Alamat = " + myConstructorsParameter.modelAlamat );

System.out.println ("Jurusan = " + myConstructorsParameter.modelJurusan );

System.out.println ("Angkatan = " + myConstructorsParameter.modelAngkatan );

}

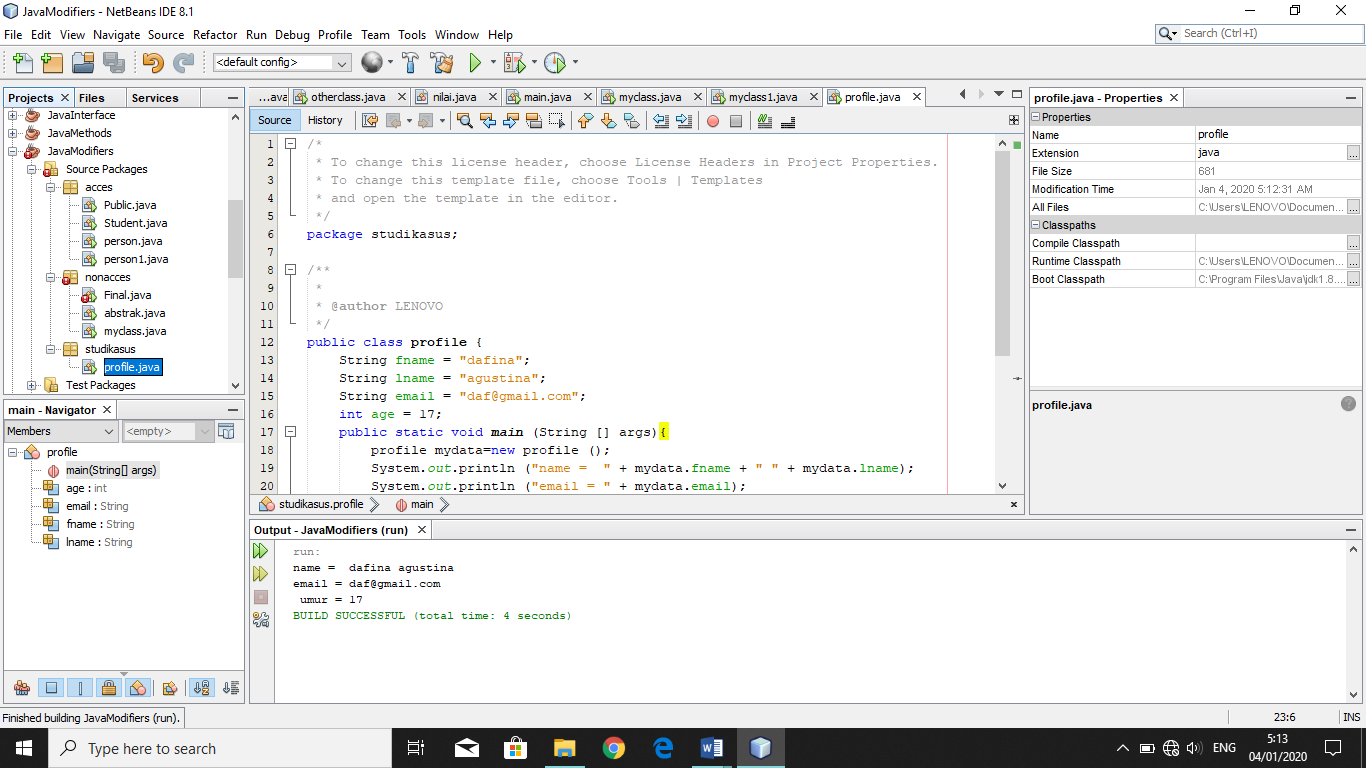
}

* + 1. **Java Modifier**

Di java ini terdapat dua bagian yaitu acces dan non acces, contoh dari

* modifier acces yaitu

saya mebuat studi kasus tetntang identitas yaitu terdapat nama, umur, dan email.yang terdapat 2 variabel yaitu String dan integer



***Kode Inputannya* :**

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class profile {

String fname = "dafina";

String lname = "agustina";

String email = "daf@gmail.com";

int age = 17;

public static void main (String [] args){

profile mydata=new profile ();

System.out.println ("name = " + mydata.fname + " " + mydata.lname);

System.out.println ("email = " + mydata.email);

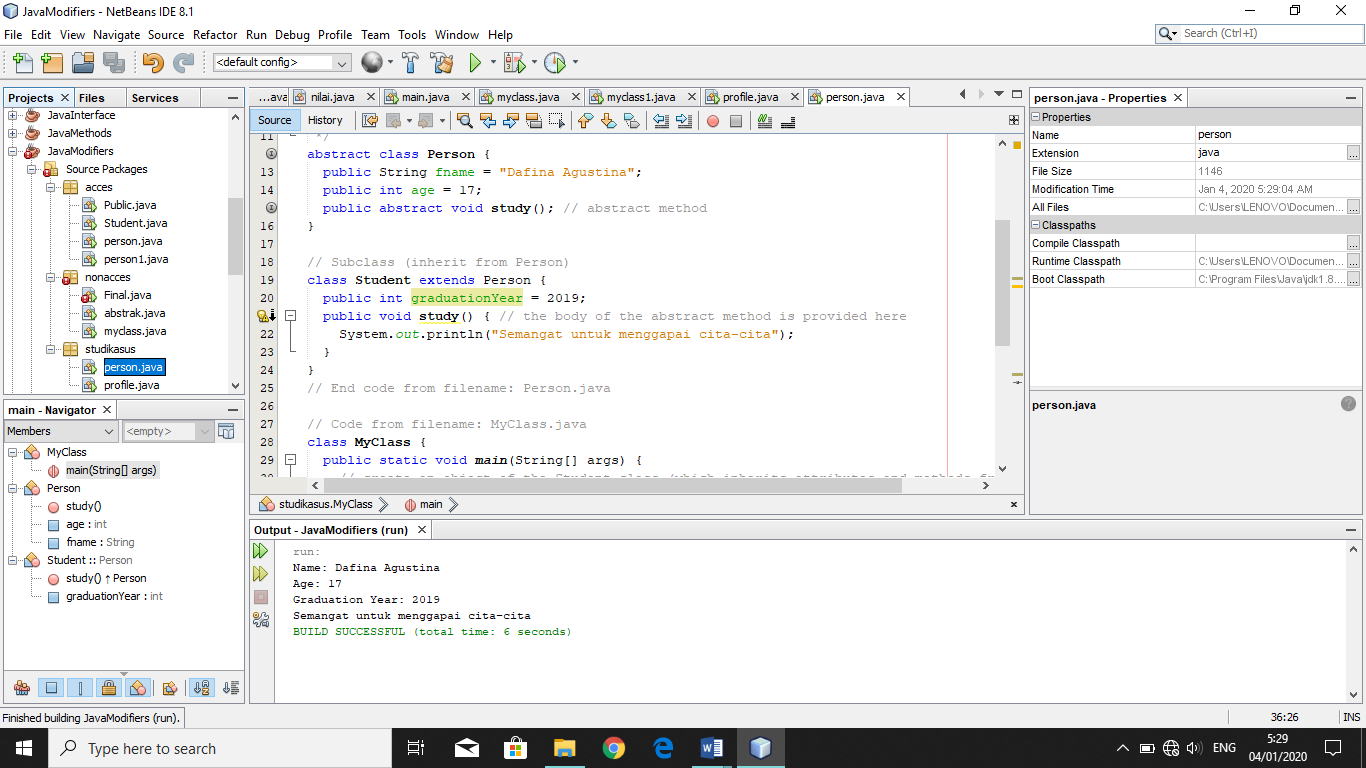
System.out.println (" umur = " + mydata.age);

}

}

Untuk objek nama saya memakai atribut String,dan untuk objek angka saya memkai atribut interger(int). void main fungsinya yaitu untuk memanggil suatu objek untuk dijalankan dan kode untuk dijalankan adalah profile mydata=new profile ();

* Modifier Non-access



***Kode Inputannya :***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

abstract class Person {

public String fname = "Dafina Agustina";

public int age = 17;

public abstract void study(); // abstract method

}

// Subclass (inherit from Person)

class Student extends Person {

public int graduationYear = 2019;

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

System.out.println("Semangat untuk menggapai cita-cita");

}

}

// End code from filename: Person.java

// Code from filename: MyClass.java

class MyClass {

public static void main(String[] args) {

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

Student myObj = new Student();

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

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

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

myObj.study(); // call abstract method

}

}

* + 1. Java Encapsulation

membuat kelas yang berbeda agar dapat menjalankan suatu programnya, di java encapsulation kita menggunakan set dan get

kelas 1 (buat kelas dengan nama profile)

***kode inputanya:***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class profile {

String nama;

int id;

public String getNama() {

return nama;

}

public void setNama(String nama) {

this.nama = nama;

}

public int getId() {

return id;

}

public void setId(int id) {

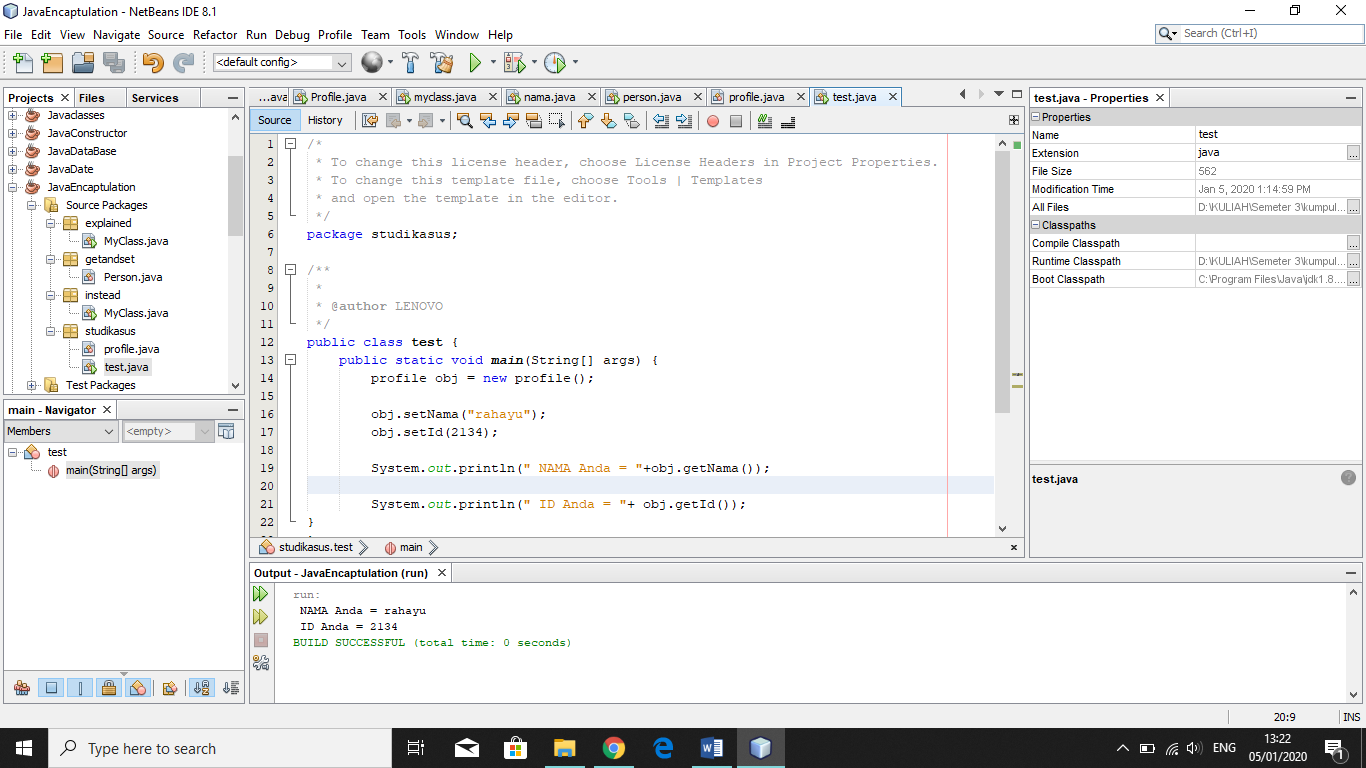
this.id = id;

}

}

Disini tidak dapat dijalankan karna tidak ada kata main() hanya untuk inputan yang akan dijalankan atau seperti rumusnya,

Kelas ke 2 (untuk menjalankan programnya)



***Kode inputanya :***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class test {

public static void main(String[] args) {

profile obj = new profile();

obj.setNama("rahayu");

obj.setId(2134);

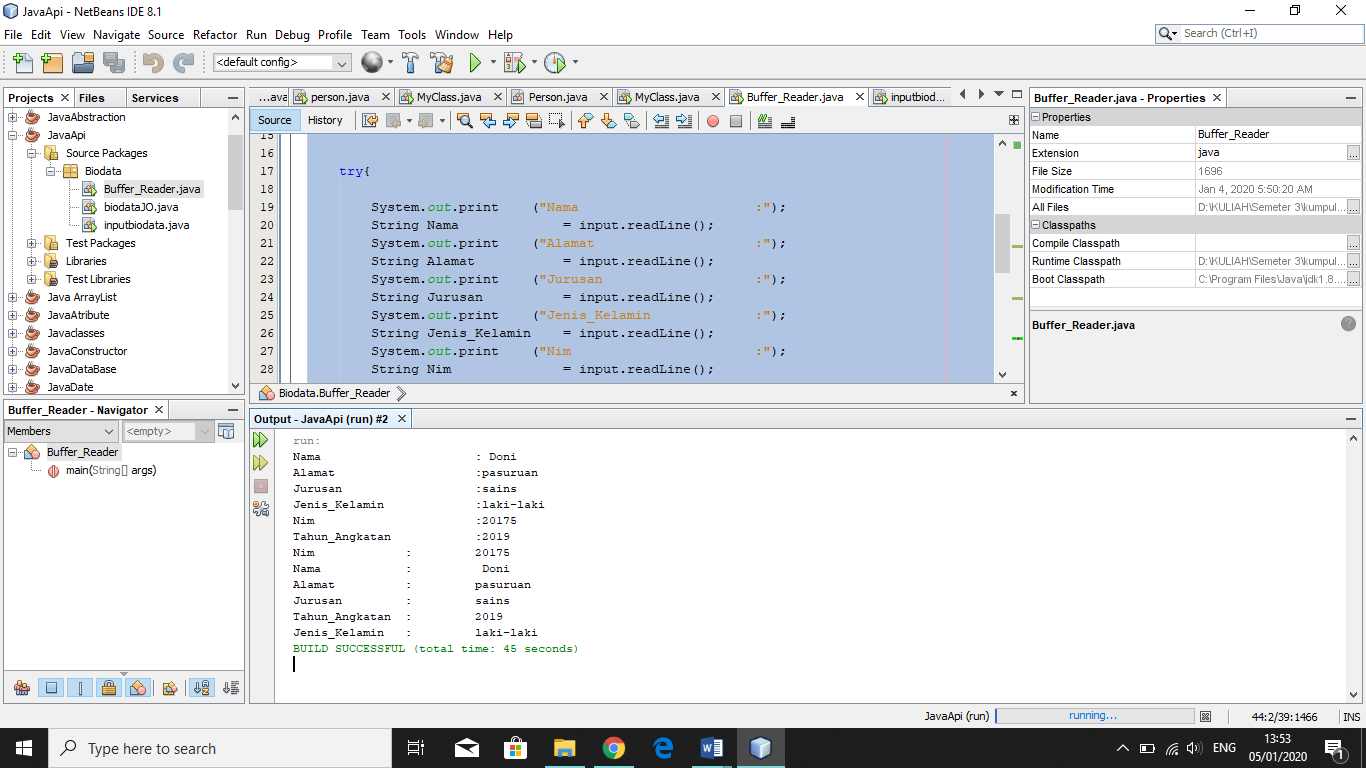
System.out.println(" NAMA Anda = "+obj.getNama());

System.out.println(" ID Anda = "+ obj.getId());

}

}

* + 1. **Java Packages/API**



Disini saya akan membuat dua kelas yaitu tetntang scanner dan bufferrider

* **Kelas ke-1 kelas Scanner**

**Keterangan :** 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



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();**

Berikut ini adalah source code lengkapnya:

package Biodata;

import java.util.Scanner;

/\*\*

\*

\*

\*/

public class inputbiodata {

public static void main(String [] args){

System.out.println("Sistem Input inputbiodata");

Scanner input = new Scanner (System.in);

System.out.print("Nama :");

String Nama = input.nextLine();

System.out.print("Alamat :");

String Alamat = input.nextLine();

System.out.print("Jurusan :");

String Jurusan = input.nextLine();

System.out.print("Jenis\_Kelamain :");

String Jenis\_Kelamin = input.nextLine();

System.out.print("Tahun\_Angkatan :");

int Tahun\_Angkatan = input.nextInt();

System.out.print("NIM :");

Long Nim = input.nextLong();

System.out.println("Nim : " + Nim);

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

System.out.println("Alamat : " + Alamat);

System.out.println("Jurusan : " + Jurusan);

System.out.println("Tahun\_Angkatan : " + Tahun\_Angkatan);

System.out.println("Jenis\_Kelamin : " + Jenis\_Kelamin);

}

}

* **Kelas yang ke-2 kelas BufferReader**

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

**import java.io.BufferedReader;**

**import java.io.IOException;**

**import java.io.inputStreamReader;**

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



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



**Berikut source code nya:**

package Biodata;

import java.io.\*;

/\*\*

\*

\*

\*/

public class Buffer\_Reader {

public static void main(String [] args){

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

try{

System.out.print ("Nama :");

String Nama = input.readLine();

System.out.print ("Alamat :");

String Alamat = input.readLine();

System.out.print ("Jurusan :");

String Jurusan = input.readLine();

System.out.print ("Jenis\_Kelamin :");

String Jenis\_Kelamin = input.readLine();

System.out.print ("Nim :");

String Nim = input.readLine();

System.out.print ("Tahun\_Angkatan :");

String Tahun\_Angkatan = input.readLine();

System.out.println("Nim : " + Nim);

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

System.out.println("Alamat : " + Alamat);

System.out.println("Jurusan : " + Jurusan);

System.out.println("Tahun\_Angkatan : " + Tahun\_Angkatan);

System.out.println("Jenis\_Kelamin : " + Jenis\_Kelamin);

}catch (IOException ex){

System.out.println("terjadi kesalahan pada input");

}

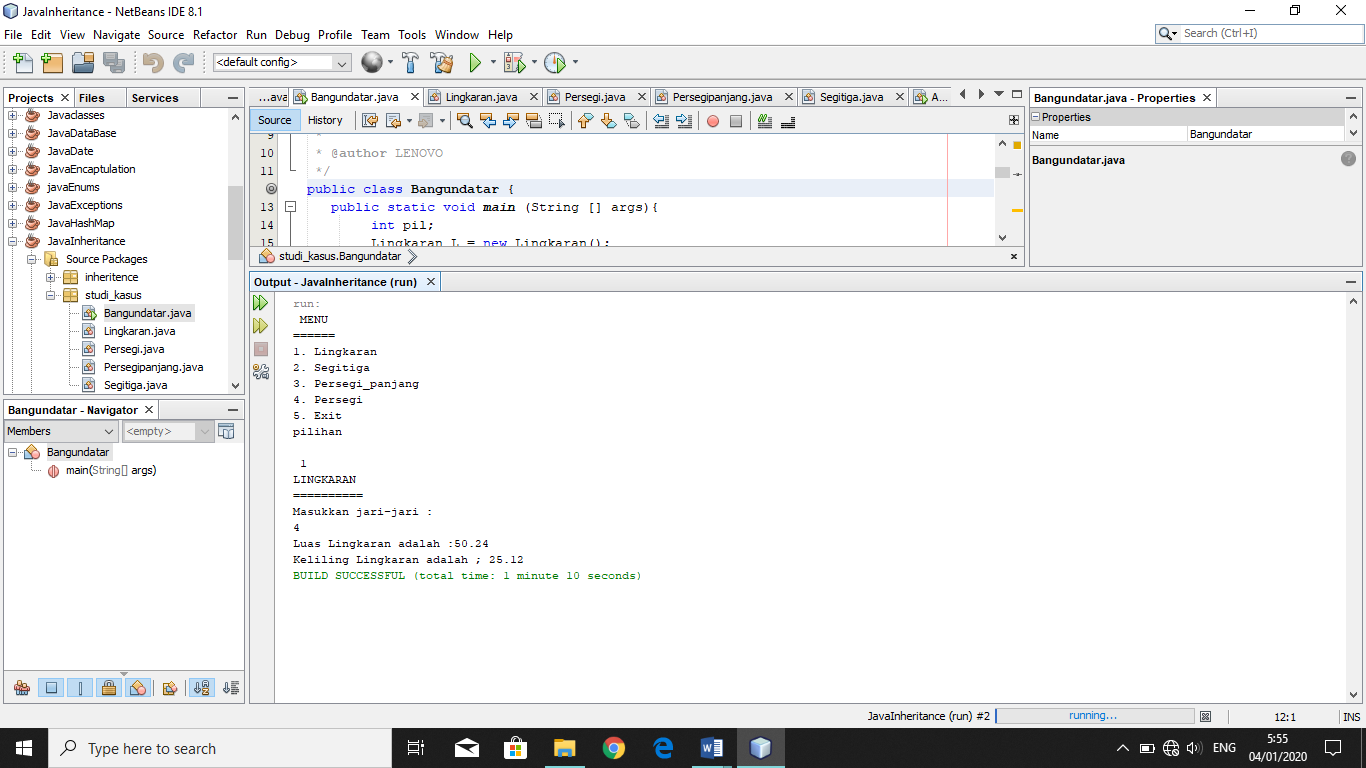
}

}

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().

* + 1. **Java Inheritance**

Saya membuat studi kasus dari bangun datar yang di jadikan menjadi beberapa kelas tapi untuk menjalankannya hanya ada pada kelas bangun datar. Dan akan menghasilkan output seperti di bawah ini:



***Kode inputannya:***

package studi\_kasus;

import java.util.\*;

/\*\*

\*

\* @author LENOVO

\*/

public class Bangundatar {

public static void main (String [] args){

int pil;

Lingkaran L = new Lingkaran();

Segitiga S = new Segitiga();

Persegipanjang PP = new Persegipanjang();

Persegi P = new Persegi();

Scanner inp = new Scanner (System.in);

System.out.println(" MENU");

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

System.out.println("1. Lingkaran");

System.out.println("2. Segitiga");

System.out.println("3. Persegi\_panjang");

System.out.println("4. Persegi");

System.out.println("5. Exit");

System.out.println("pilihan");

pil = inp.nextInt();

switch(pil){

case 1 : L.Lingkaran();break;

case 2 : S.Segitiga();break;

case 3 : PP.Persegipanjang();break;

case 4 : P.Persegi();break;

case 5 : System.exit(0);

}

}

}

Di bagian kelas bangun datar ini adalah untuk menjalankan suatu project nya, di java ini kita memakai perulangan (switch-case). Di bagian outputnya kita bisa memilih salah satu dari pilihannya dan memasukkan isinya secara manual.

Berikut inputan dari bebrapa kelas bangundatar.

* *Bangun datar lingkaran*

package studi\_kasus;

import java.util.Scanner;

/\*\*

\*

\* @author LENOVO

\*/

public class Lingkaran extends Bangundatar {

public void Lingkaran(){

double ll,rl,kl,phi = 3.14;

Scanner b = new Scanner (System.in);

System.out.println("LINGKARAN");

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

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

rl = b.nextDouble();

ll = phi\*rl\*rl;

kl = 2\*phi\*rl;

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

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

}

}

* *Bangun datar persegi*

package studi\_kasus;

import java.util.Scanner;

/\*\*

\*

\* @author LENOVO

\*/

public class Persegi extends Bangundatar {

public void Persegi(){

double sp,lp,kp;

Scanner c = new Scanner (System.in);

System.out.println("PERSEGI");

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

System.out.println("Masukkan sisi");

sp = c.nextDouble();

lp = sp\*sp;

kp = 4\*sp;

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

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

}

}

* *Bangun datar persegi panjang*

package studi\_kasus;

import java.util.Scanner;

/\*\*

\*

\* @author LENOVO

\*/

public class Persegipanjang {

public static void Persegipanjang(){

double pp,lep,lp,kp;

Scanner d = new Scanner(System.in);

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

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

System.out.println("Masukkan Panjang");

pp = d.nextDouble();

System.out.println("Masukkan Lebar");

lep = d.nextDouble();

lp = pp\*lep;

kp = (2\*pp)+(2\*lep);

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

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

}

}

* *Bangun datar segitiga*

package studi\_kasus;

import java.util.\*;

/\*\*

\*

\* @author LENOVO

\*/

public class Segitiga extends Bangundatar {

public void Segitiga(){

double ls,as,ts,ss,ks;

Scanner a = new Scanner (System.in);

System.out.println(System.in);

System.out.println("SEGITIGA");

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

System.out.println("Masukkan alas");

as = a.nextDouble();

System.out.println("Masukkan tinggi");

ts = a.nextDouble();

System.out.println("Masukkan sisi");

ss = a.nextDouble();

ls = (as\*ts)/2;

ks = (ss\*2)+as;

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

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

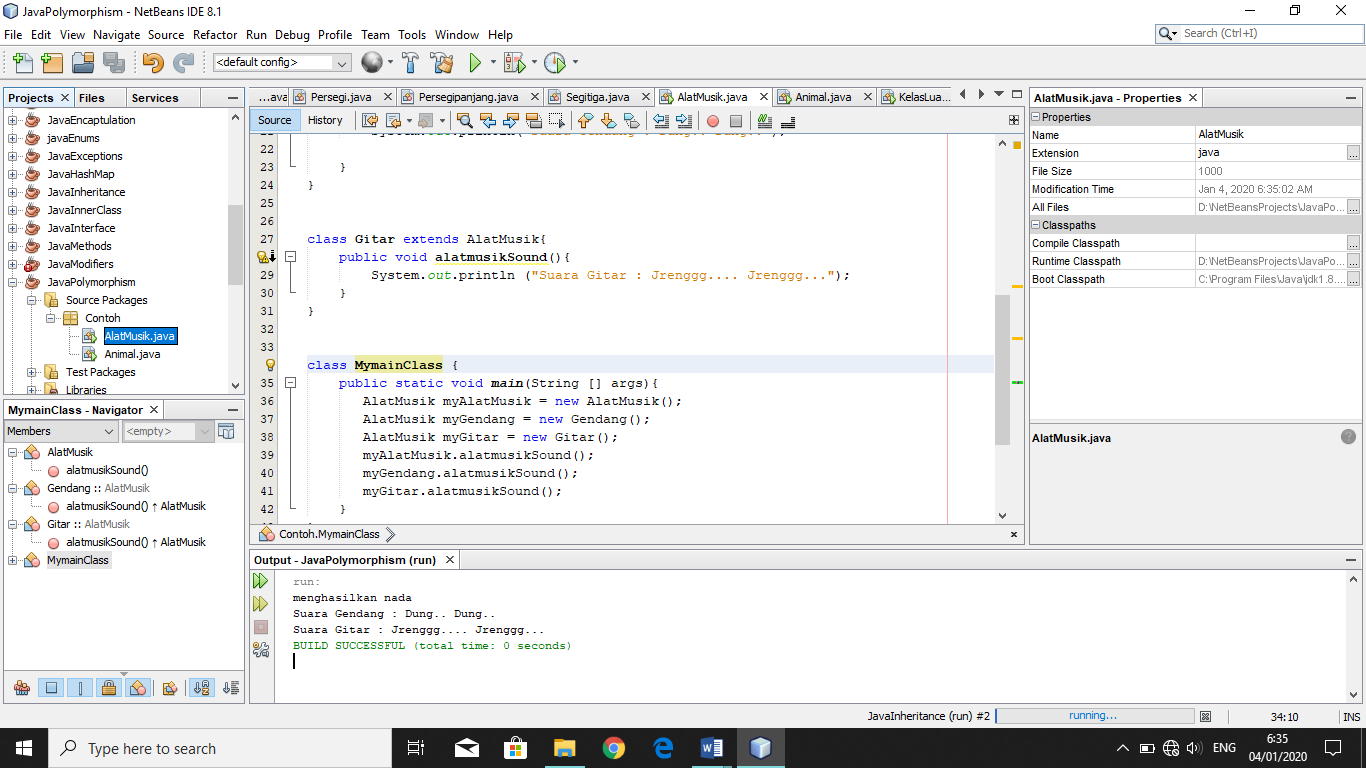
}

}

Berikut adalah inputan dari bangun datar yang terdiri beberapa kelas.

Program ini menentukan suatu bangun datar yang ada pada matematika yang di dalamnya terdapat rumus dari bangun datar tersebut, lalu pada outputnya kita harus mengisi angka untuk mengisi rumus dari bangun datar tersebut.

* + 1. **Java Polymorphism**

Polymorphism artinya memiliki banyak bentuk.

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

***Kode inputannya :***

package Contoh;

/\*\*

\*

\* @author User

\*/

public class AlatMusik {

public void alatmusikSound(){

System.out.println ("menghasilkan nada");

}

}

class Gendang extends AlatMusik{

public void alatmusikSound(){

System.out.println("Suara Gendang : Dung.. Dung..");

}

}

class Gitar extends AlatMusik{

public void alatmusikSound(){

System.out.println ("Suara Gitar : Jrenggg.... Jrenggg...");

}

}

class MyMainClass {

public static void main(String [] args){

AlatMusik myAlatMusik = new AlatMusik();

AlatMusik myGendang = new Gendang();

AlatMusik myGitar = new Gitar();

myAlatMusik.alatmusikSound();

myGendang.alatmusikSound();

myGitar.alatmusikSound();

}

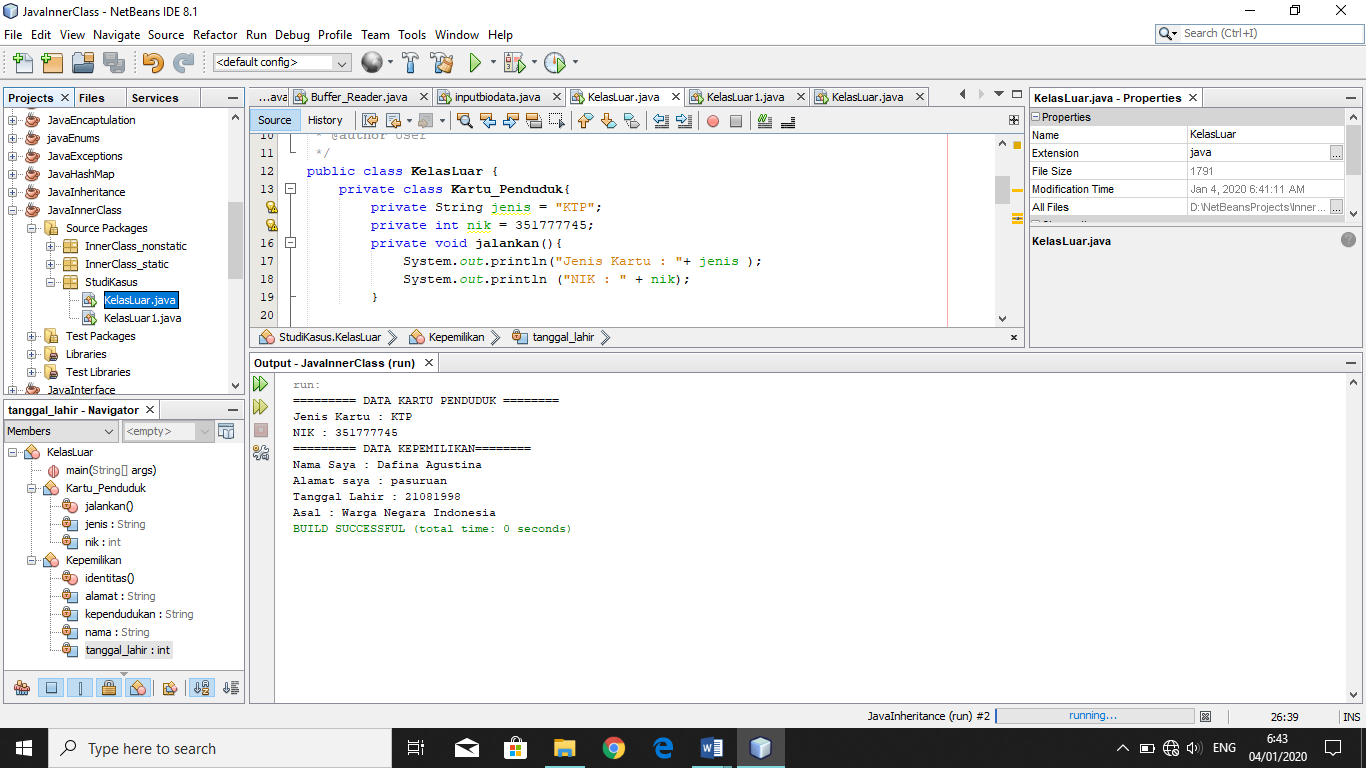
}

1. **Java Inner Class**

Di java ini terdapat dua kelas yaitu kelas non-static dan kelas static

* + Kelas non-static

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



***Kode inputannya :***

package StudiKasus;

/\*\*

\*

\* @author User

\*/

public class KelasLuar {

private class Kartu\_Penduduk{

private String jenis = "KTP";

private int nik = 351777745;

private void jalankan(){

System.out.println("Jenis Kartu : "+ jenis );

System.out.println ("NIK : " + nik);

}

}

//class dalam/inner class kedua

private class Kepemilikan {

private String nama = "Dafina Agustina";

private String alamat = "pasuruan";

private int tanggal\_lahir = 21\_08\_1998;

private String kependudukan = "Warga Negara Indonesia";

private void identitas(){

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

System.out.println("Alamat saya : " + alamat);

System.out.println("Tanggal Lahir : " + tanggal\_lahir);

System.out.println("Asal : " +kependudukan);

}

}

public static void main (String[] args){

//Membuat Instance dari kelasLuar

KelasLuar outerclass = new KelasLuar();

//Membuat Instance dari KelasDalam (Mobil)

KelasLuar.Kartu\_Penduduk data1 = outerclass. new Kartu\_Penduduk();

//Membuat instance dari KelasDalam (pengguna)

KelasLuar.Kepemilikan data2 = outerclass.new Kepemilikan();

//Menampilkan hasil Output

System.out.println ("========= DATA KARTU PENDUDUK ========");

data1.jalankan();

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

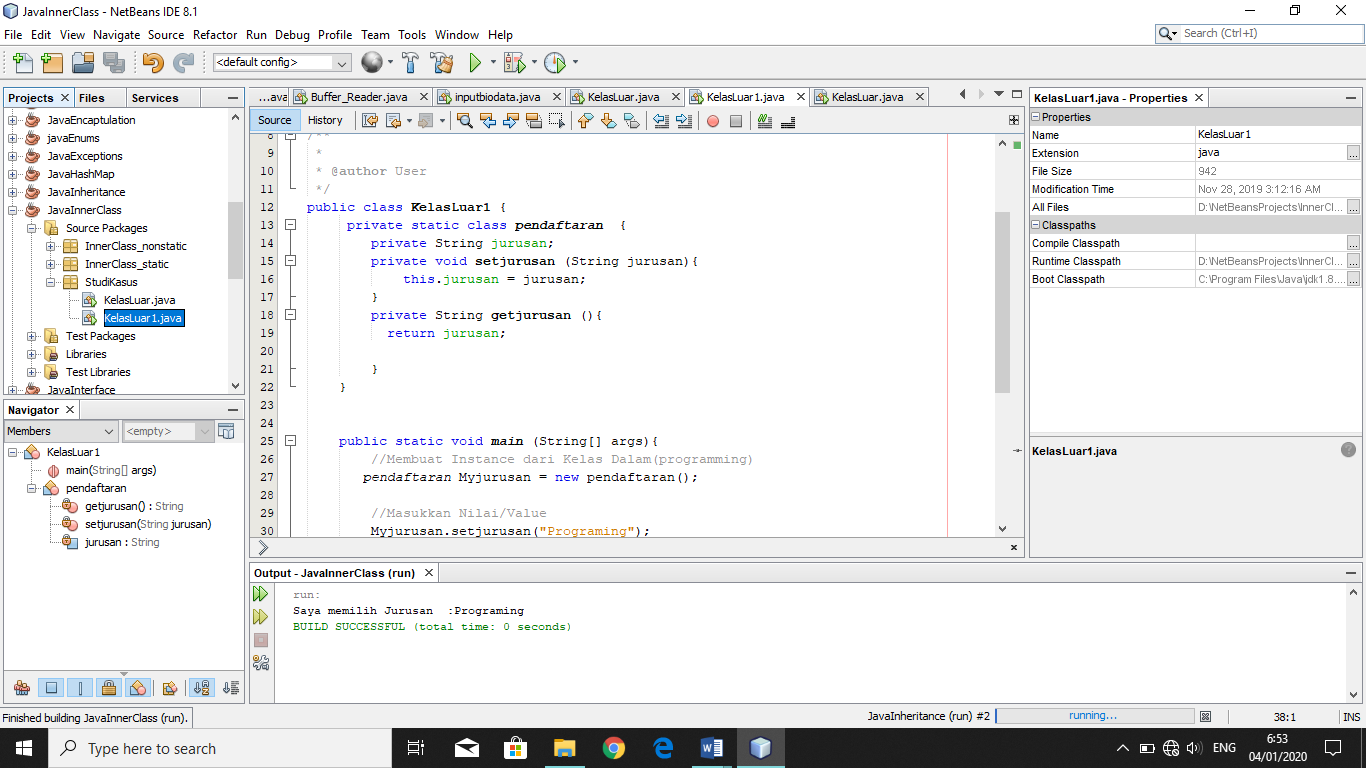
data2.identitas();

}

}

* + Untuk kelas static

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



***Kode inputannya :***

package StudiKasus;

/\*\*

\*

\* @author User

\*/

public class KelasLuar1 {

private static class pendaftaran {

private String jurusan;

private void setjurusan (String jurusan){

this.jurusan = jurusan;

}

private String getjurusan (){

return jurusan;

}

}

public static void main (String[] args){

//Membuat Instance dari Kelas Dalam(programming)

pendaftaran Myjurusan = new pendaftaran();

//Masukkan Nilai/Value

Myjurusan.setjurusan("Programing");

//Menampilkan Hsil Output

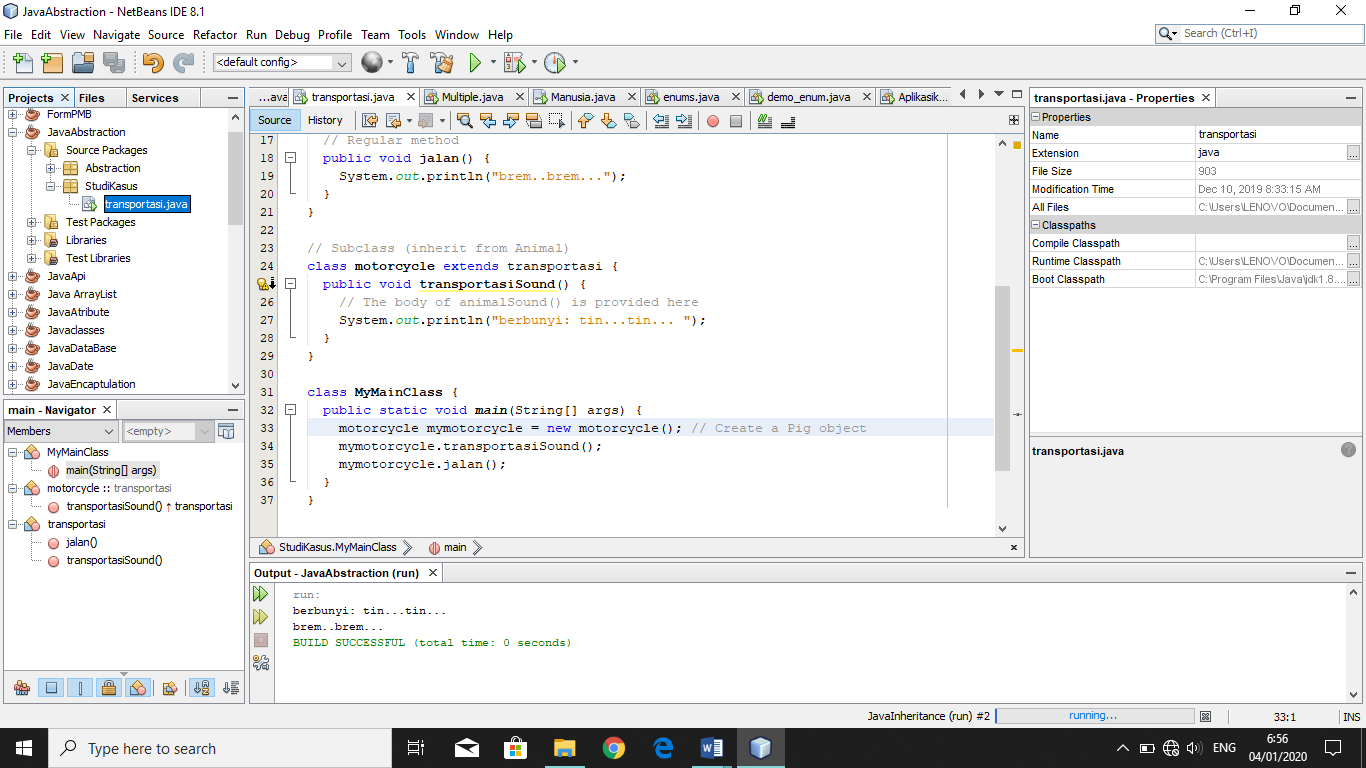
System.out.println("Saya memilih Jurusan :" +Myjurusan.getjurusan());

}

}

1. **Java Abstraction**

Saya membuat studi kasus tentang transportasi dalam satu kelas.



***Kode inputannya :***

package StudiKasus;

/\*\*

\*

\* @author LENOVO

\*/

abstract class transportasi{

// Abstract method (does not have a body)

public abstract void transportasiSound();

// Regular method

public void jalan() {

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

}

}

// Subclass (inherit from Animal)

class motorcycle extends transportasi {

public void transportasiSound() {

// The body of animalSound() is provided here

System.out.println("berbunyi: tin...tin... ");

}

}

class MyMainClass {

public static void main(String[] args) {

motorcycle mymotorcycle = new motorcycle(); // Create a Pig object

mymotorcycle.transportasiSound();

mymotorcycle.jalan();

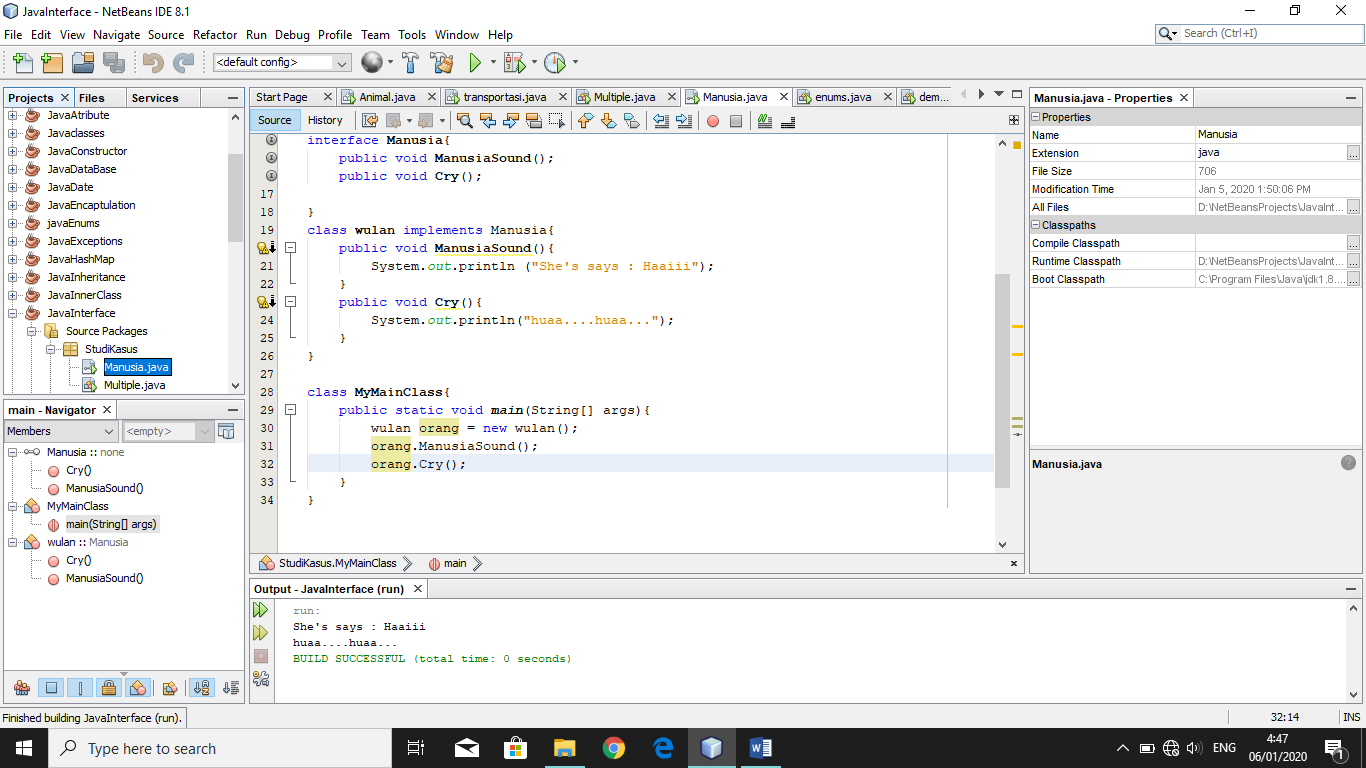
}

}

1. **Java Interface**

Disini terdapat dua kelas yaitu

* + Kelas pertama yaitu saya beri nama kelas manusia



***Kode inputannya :***

package StudiKasus;

/\*\*

\*

\* @author User

\*/

interface Manusia{

public void ManusiaSound();

public void Cry();

}

class wulan implements Manusia{

public void ManusiaSound(){

System.out.println ("She's says : Haaiii");

}

public void Cry(){

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

}

}

class MyMainClass{

public static void main(String[] args){

wulan orang = new wulan();

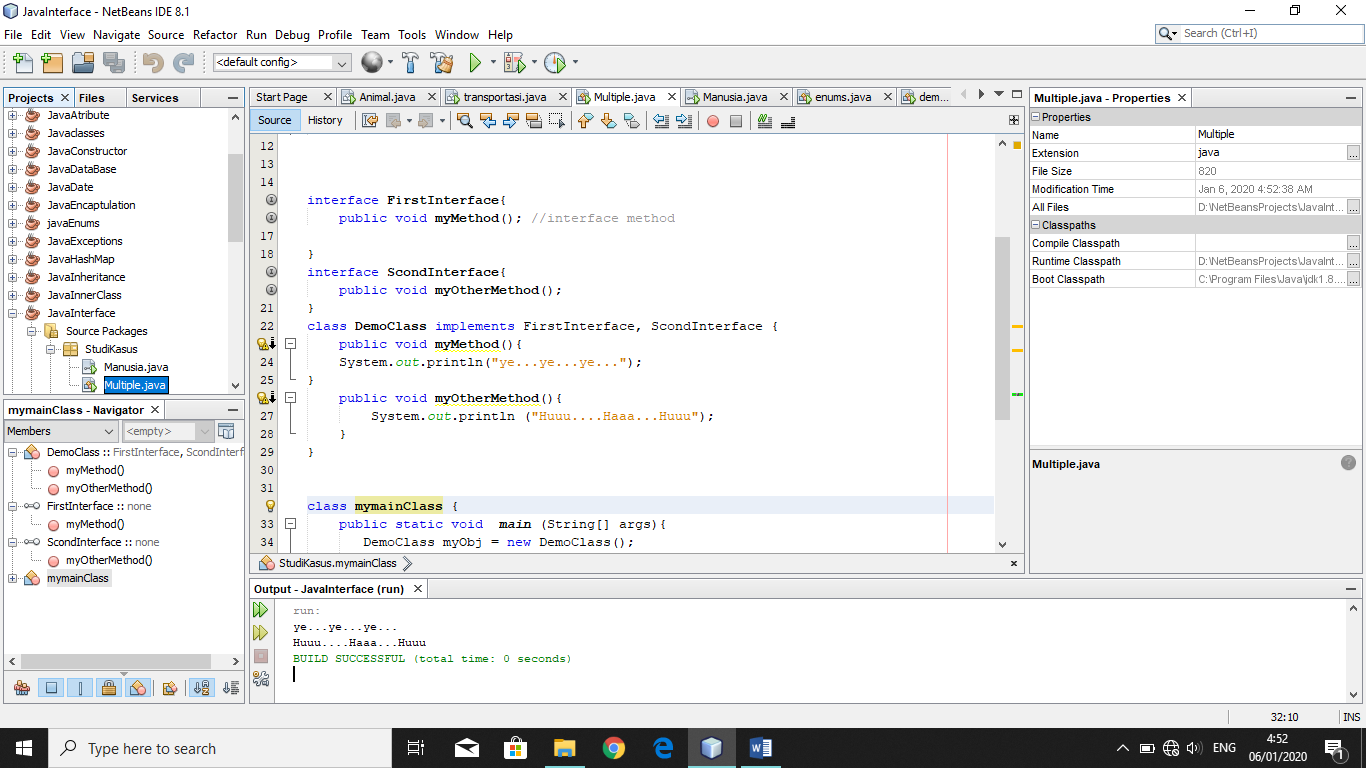
orang.ManusiaSound();

orang.Cry();

}

}

* + Kelas ke-2 yaitu saya beri kelas multiple



***Kode inputannya :***

package StudiKasus;

/\*\*

\*

\* @author User

\*/

interface FirstInterface{

public void myMethod(); //interface method

}

interface ScondInterface{

public void myOtherMethod();

}

class DemoClass implements FirstInterface, ScondInterface {

public void myMethod(){

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

}

public void myOtherMethod(){

System.out.println ("Huuu....Haaa...Huuu");

}

}

class mymainClass {

public static void main (String[] args){

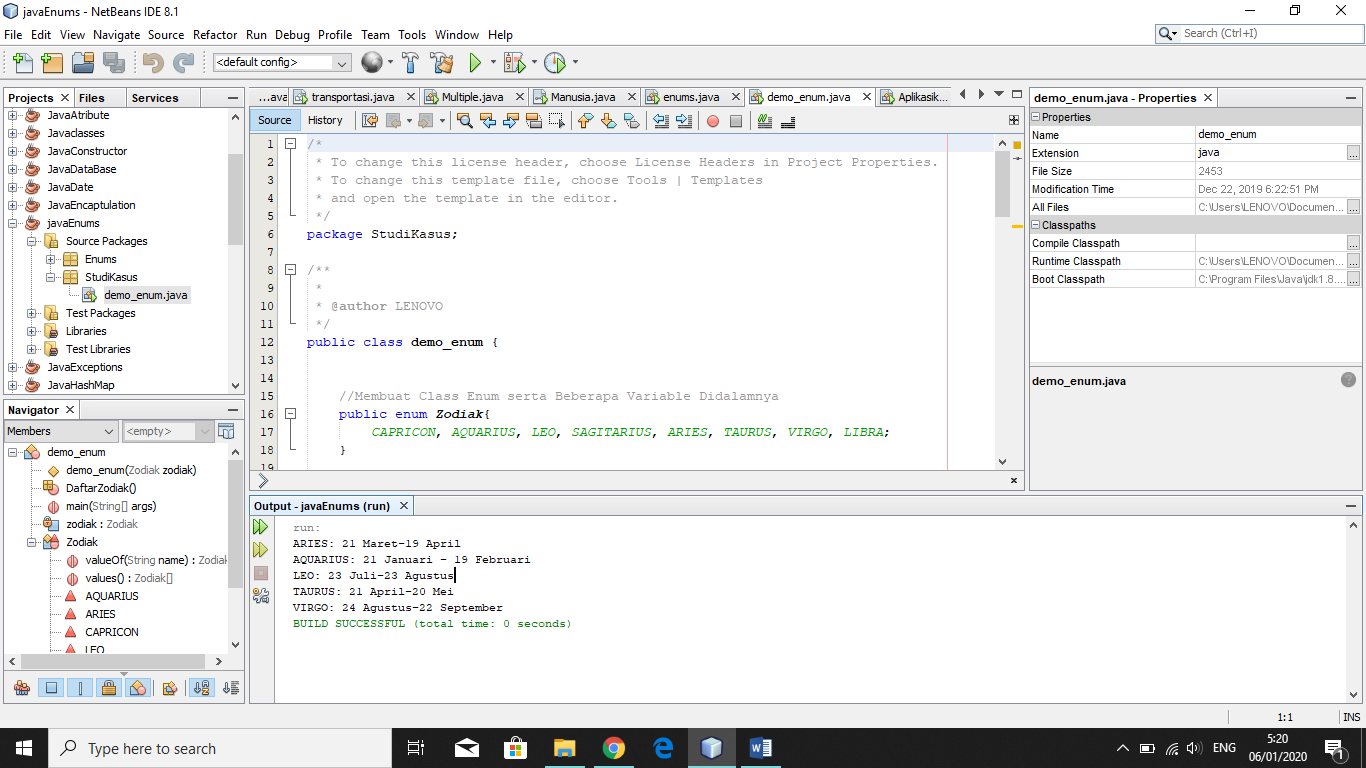
DemoClass myObj = new DemoClass();

myObj.myMethod();

myObj.myOtherMethod();

}

1. **Java Enums**



***Kode inputannya :***

package StudiKasus;

/\*\*

\*

\* @author LENOVO

\*/

public class demo\_enum {

//Membuat Class Enum serta Beberapa Variable Didalamnya

public enum Zodiak{

CAPRICON, AQUARIUS, LEO, SAGITARIUS, ARIES, TAURUS, VIRGO, LIBRA;

}

//Membuat Objek dari Enum

private Zodiak zodiak;

//Sebuah Konstruktor dengan Parameter Enum

public demo\_enum(Zodiak zodiak){

this.zodiak = zodiak;

}

//Mendefinisikan Nilai pada Masing-Masing Variable Enum

void DaftarZodiak(){

//Method Switch Digunakan Untuk Menentukan Opsi Mana Yang Akan Dipilih

switch(zodiak){

case AQUARIUS:

System.out.println("AQUARIUS: 21 Januari - 19 Februari");

break;

case ARIES:

System.out.println("ARIES: 21 Maret-19 April");

break;

case CAPRICON:

System.out.println("CAPRICON: 22 Desember - 20 Januari");

break;

case LIBRA:

System.out.println("LIBRA: 23 September-23 Oktober");

break;

case LEO:

System.out.println("LEO: 23 Juli-23 Agustus");

break;

case SAGITARIUS:

System.out.println("SAGITARIUS: 23 November-21 Desember");

break;

case TAURUS:

System.out.println("TAURUS: 21 April-20 Mei");

break;

case VIRGO:

System.out.println("VIRGO: 24 Agustus-22 September");

break;

}

}

public static void main(String[] args){

//Membuat Instance/Objek Serta Menentukan Variable/Opsi Mana Yang Akan Dipilih

demo\_enum data1 = new demo\_enum(Zodiak.ARIES);

data1.DaftarZodiak();

demo\_enum data2 = new demo\_enum(Zodiak.AQUARIUS);

data2.DaftarZodiak();

demo\_enum data3 = new demo\_enum(Zodiak.LEO);

data3.DaftarZodiak();

demo\_enum data4 = new demo\_enum(Zodiak.TAURUS);

data4.DaftarZodiak();

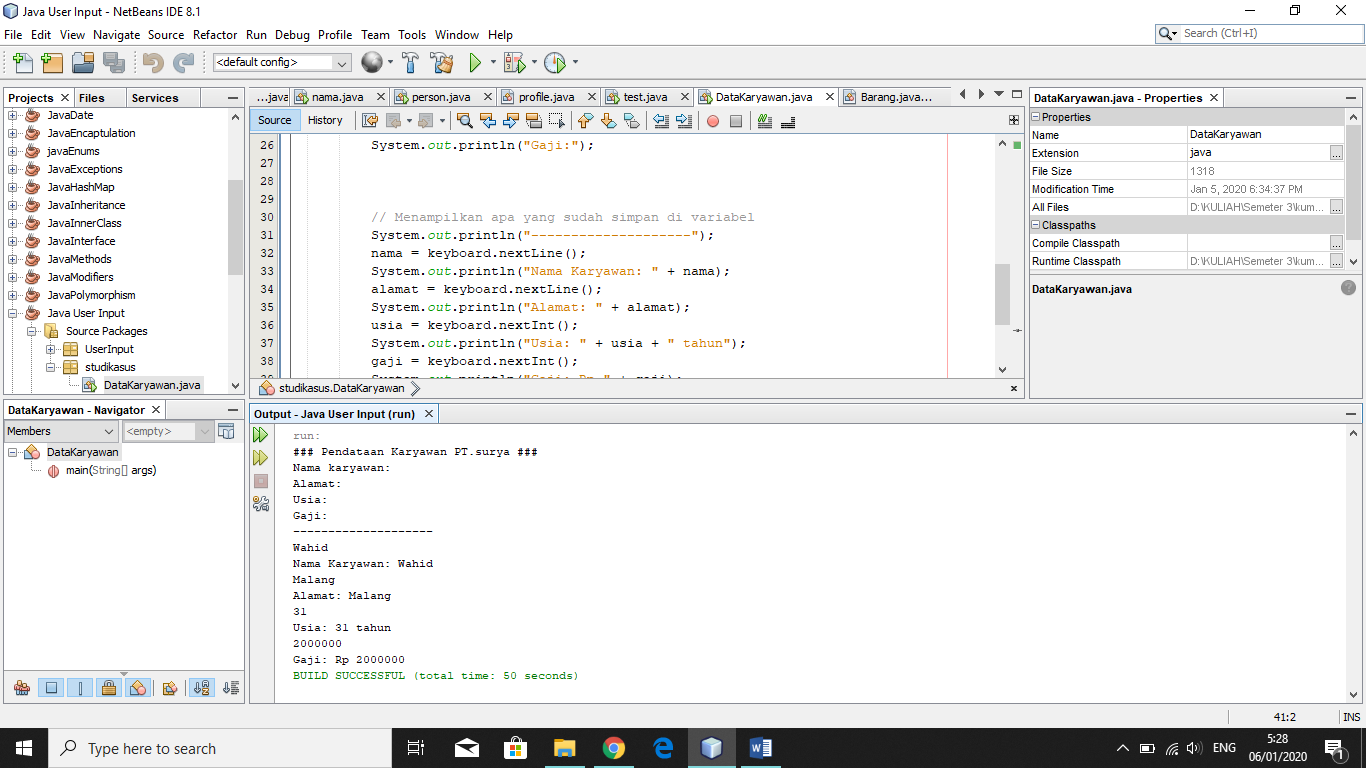
demo\_enum data5 = new demo\_enum(Zodiak.VIRGO);

data5.DaftarZodiak();

}

}

1. **Java User Input**



***Kode inputannya :***

package studikasus;

import java.util.Scanner;

/\*\*

\*

\* @author LENOVO

\*/

public class DataKaryawan {

public static void main(String[] args) {

// deklarasi variabel

String nama, alamat;

int usia, gaji;

// membuat scanner baru

Scanner keyboard = new Scanner(System.in);

// Tampilkan output ke user

System.out.println("### Pendataan Karyawan PT.surya ###");

System.out.println("Nama karyawan: ");

System.out.println("Alamat: ");

System.out.println("Usia: ");

System.out.println("Gaji:");

// Menampilkan apa yang sudah simpan di variabel

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

nama = keyboard.nextLine();

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

alamat = keyboard.nextLine();

System.out.println("Alamat: " + alamat);

usia = keyboard.nextInt();

System.out.println("Usia: " + usia + " tahun");

gaji = keyboard.nextInt();

System.out.println("Gaji: Rp " + gaji);

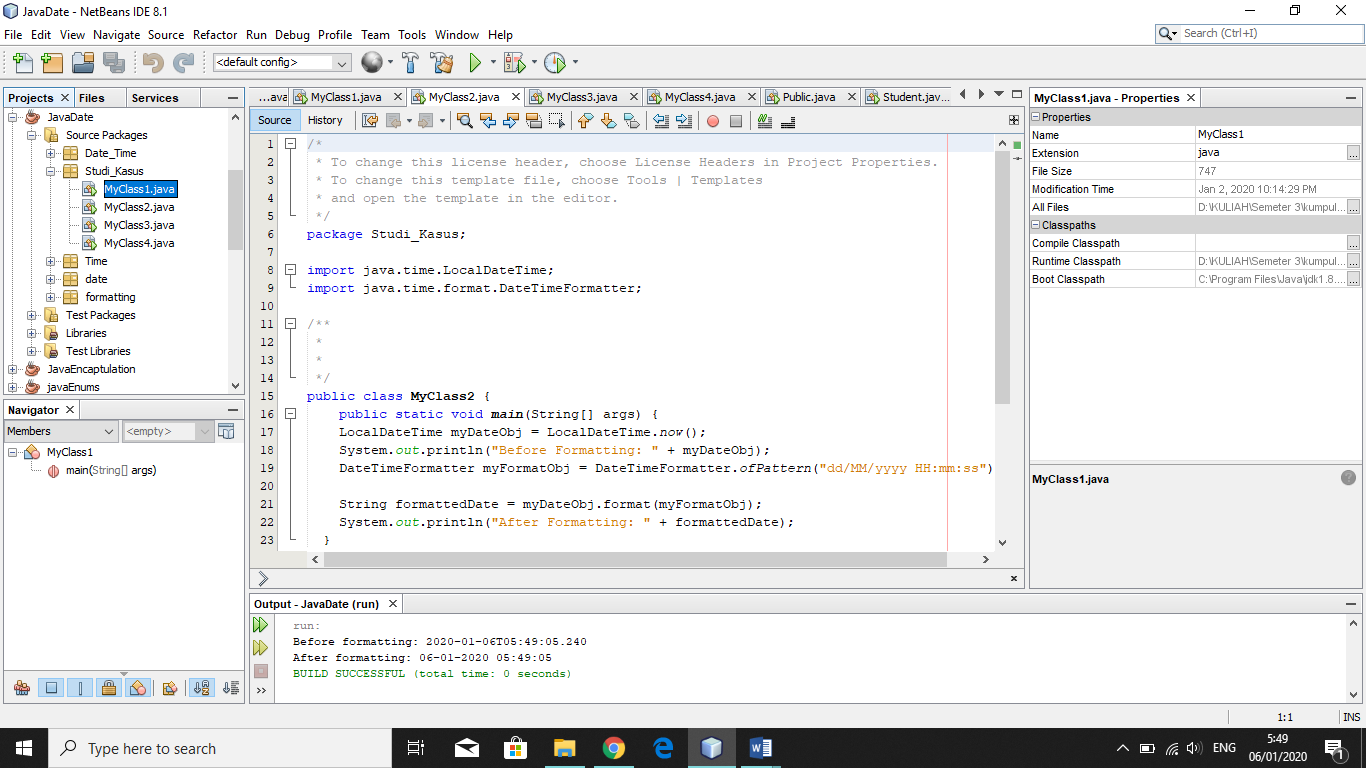
}

}

1. **Java Date**

Di java date terdapat empat kelas yaitu

* + dd-MM-yyyy



***Kode inputannya :***

package Studi\_Kasus;

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");

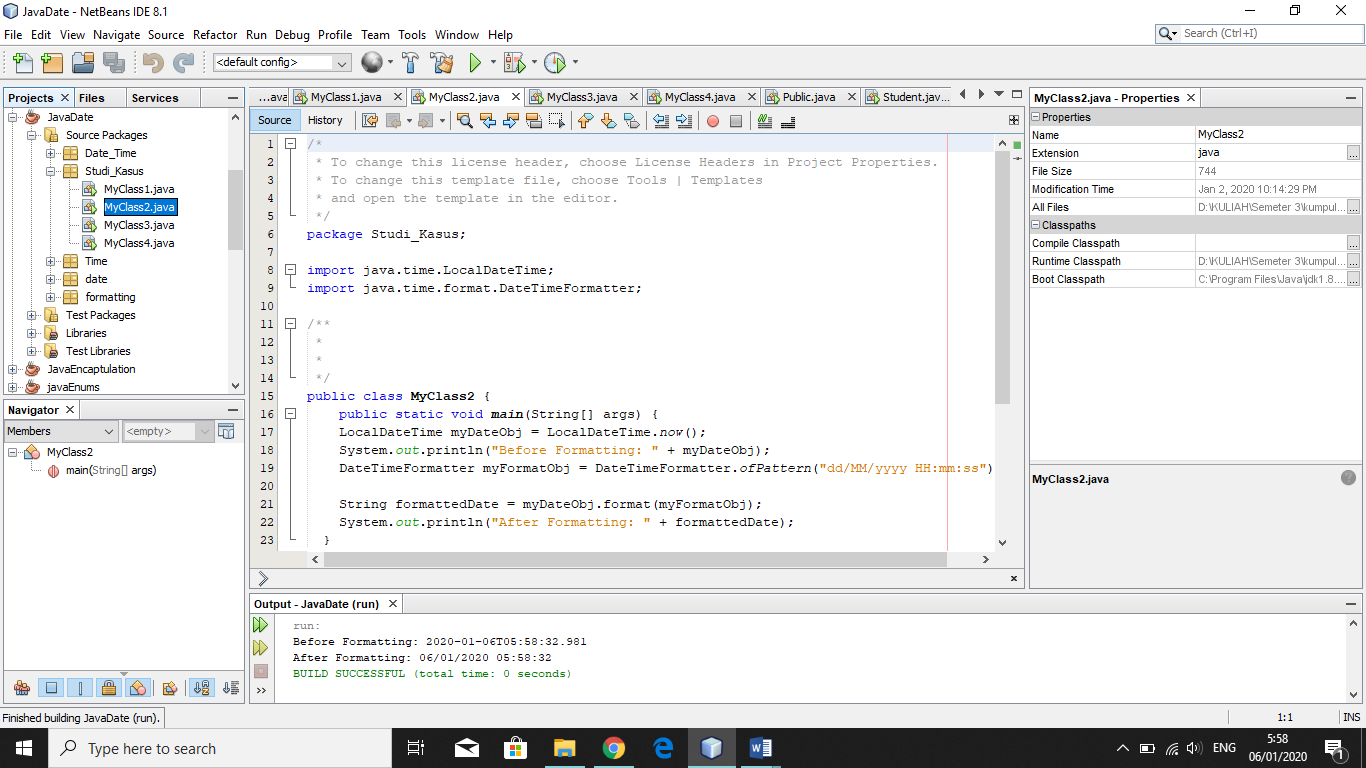
String formattedDate = myDateObj.format(myFormatObj);

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

}

}

* + dd/MM/yyyy



***Kode inputannya:***

package Studi\_Kasus;

import java.time.LocalDateTime;

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");

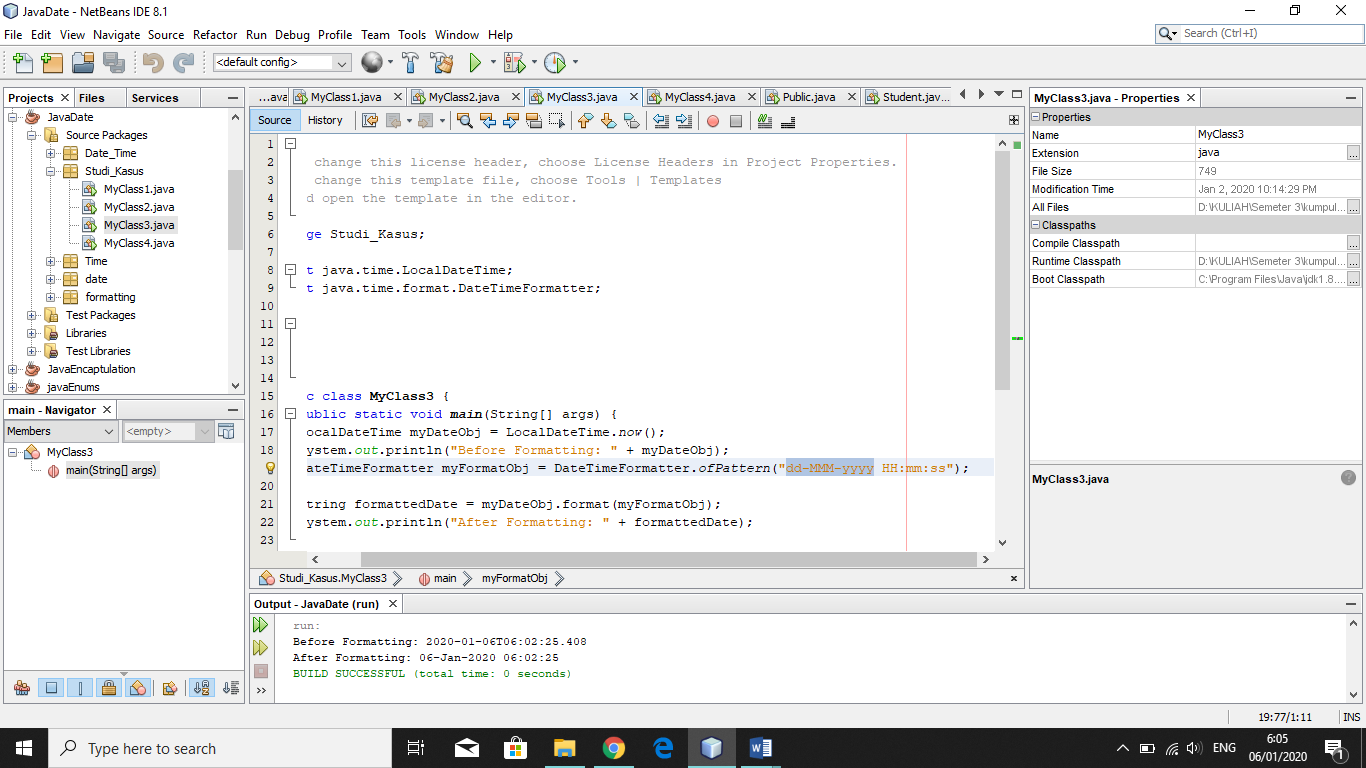
String formattedDate = myDateObj.format(myFormatObj);

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

}

}

* + dd-MMM-yyyy



***Kode inputannya :***

package Studi\_Kasus;

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");

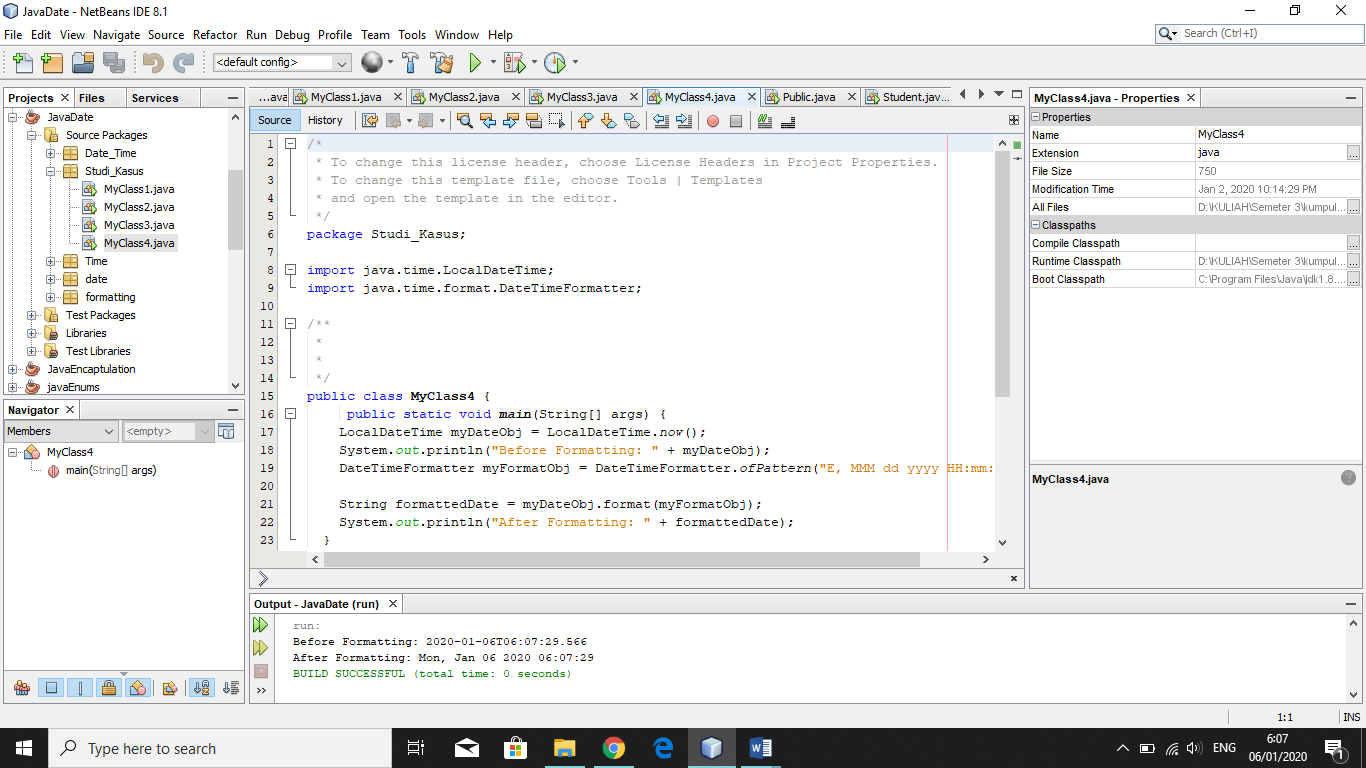
String formattedDate = myDateObj.format(myFormatObj);

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

}

}

* + E, MMM dd yyyy



***Kode inputannya:***

package Studi\_Kasus;

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);

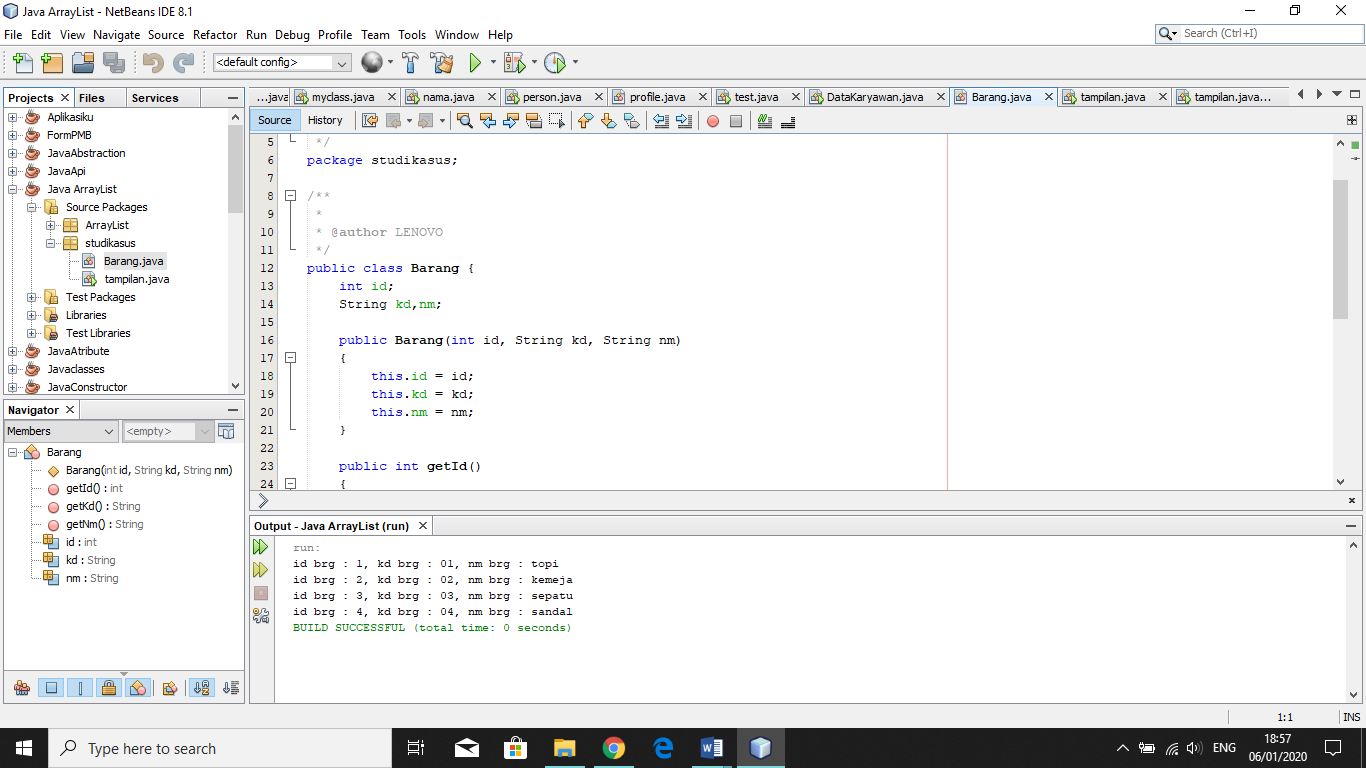
System.out.println("After Formatting: " + formattedDate);

}

}

1. **Java ArrayList**

Di java ini terdapat dua kelas yaitu kelas pertama untuk nama barangnya saja maksudnya hanya isi inputannya saja tidak dapat di jalankan, jika kelas yang ke dua yaitu untuk menjalan suatu programnya pada kelas pertama dan akan mendapatkan hasil outputnya.



***Kode inputannya :***

Kelas pertama

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class Barang {

int id;

String kd,nm;

public Barang(int id, String kd, String nm)

{

this.id = id;

this.kd = kd;

this.nm = nm;

}

public int getId()

{

return id;

}

public String getKd()

{

return kd;

}

public String getNm()

{

return nm;

}

}

Kelas tampilannya :

package studikasus;

import java.util.ArrayList;

/\*\*

\*

\* @author LENOVO

\*/

public class tampilan {

ArrayList<Barang> tampung;

public tampilan()

{

//instansiasi

tampung = new ArrayList<Barang>();

}

public void isiData(int id,String kd,String nm)

{

tampung.add(new Barang(id,kd,nm));

}

public void showData()

{

for(Barang barang:tampung)

{

System.out.println("id brg : "+barang.getId()+""+", kd brg : "+barang.getKd()+""+", nm brg : "+barang.getNm());

}

}

public static void main(String args[])

{

tampilan td = new tampilan();

td.isiData(1,"01","topi");

td.isiData(2,"02","kemeja");

td.isiData(3,"03","sepatu");

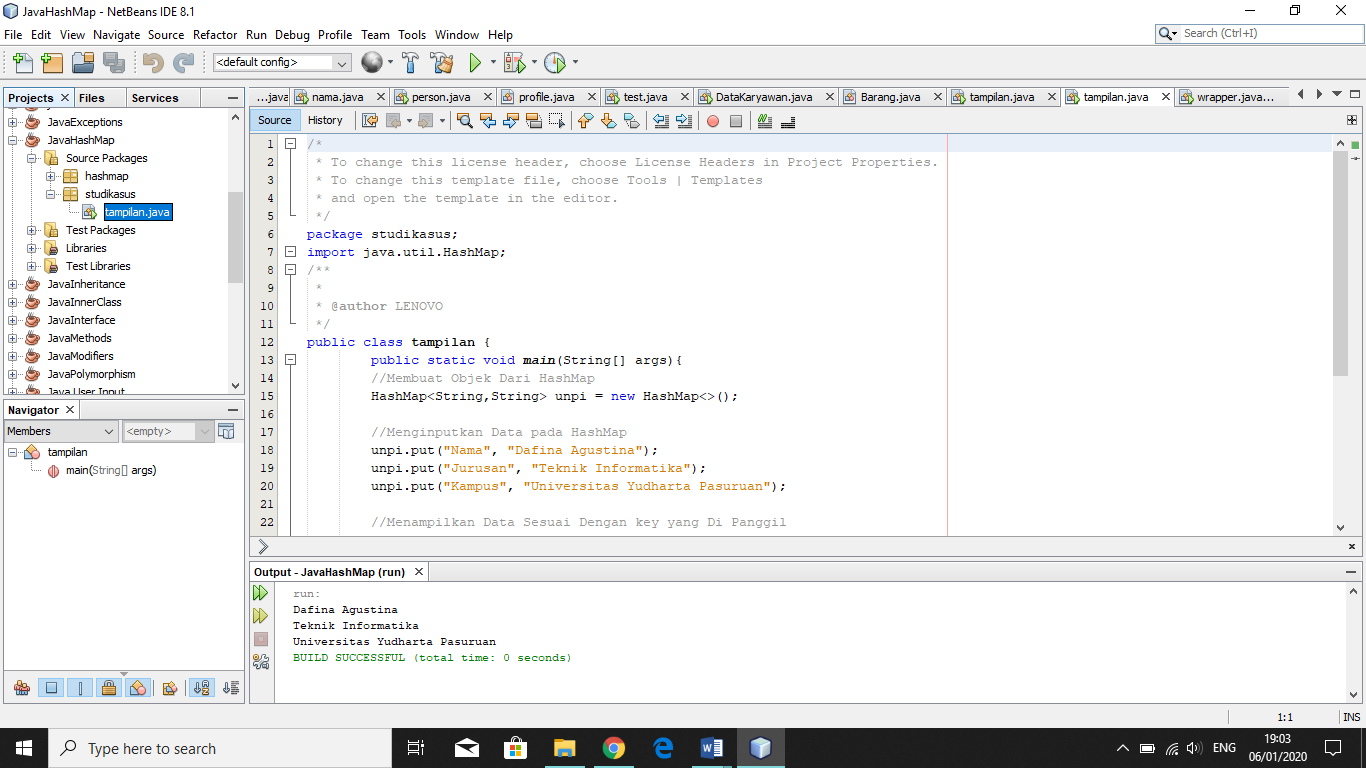
td.isiData(4,"04","sandal");

td.showData();

}

}

1. **Java HashMap**



***Kode inputannya :***

package studikasus;

import java.util.HashMap;

/\*\*

\*

\* @author LENOVO

\*/

public class tampilan {

public static void main(String[] args){

//Membuat Objek Dari HashMap

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

//Menginputkan Data pada HashMap

unpi.put("Nama", "Dafina Agustina");

unpi.put("Jurusan", "Teknik Informatika");

unpi.put("Kampus", "Universitas Yudharta Pasuruan");

//Menampilkan Data Sesuai Dengan key yang Di Panggil

System.out.println(unpi.get("Nama"));

System.out.println(unpi.get("Jurusan"));

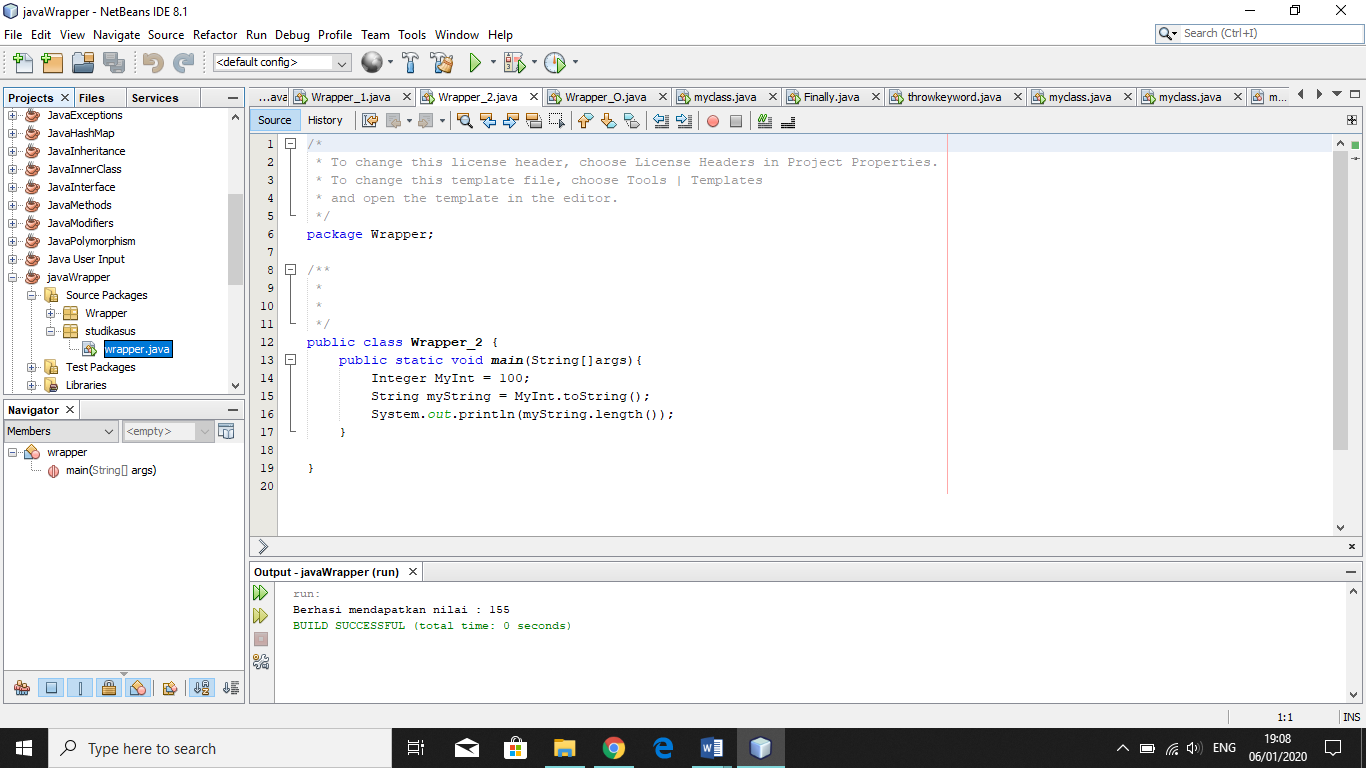
System.out.println(unpi.get("Kampus"));

}

}

Untuk membuat mendeklarasikan objek HashMap, syntax awalan seperti berikut ini:  
**HashMap<Key,Value>objek=newHashMap<>();**  
**Key:** Digunakan sebagai Kunci Untuk Memanggil Value.  
**Value:** Merupakan Isi Nilai Yang Akan Dipanggil Oleh Key.  
Pada program tersebut, kita membuat objek awal seperti ini **HashMap<String, String>** yang artinya, pada Key dan Value mempunyai tipe data String, lalu untuk menginputkan data, kalian dapat menggunakan fungsi put() diisi dengan Key dan Valuenya, lalu untuk mengakses valuenya, kalian dapat menggunakan fungsi get() didalamnya masukan Key untuk mengakses value yang diinginkan.  
Tipe data pada Key dan Value dapat kalian bedakan, misalnya **<Integer, String>, <String, Integer>, <Integer, Integer>, <Boolean, Double>, dsb**.

1. **Java Wrapper Classes**



***Kode Inputannya :***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class wrapper {

public static void main (String [] args){

Integer a = new Integer (155);

int i = a.intValue();

System.out.println("Berhasi mendapatkan nilai : " + i);

}

}

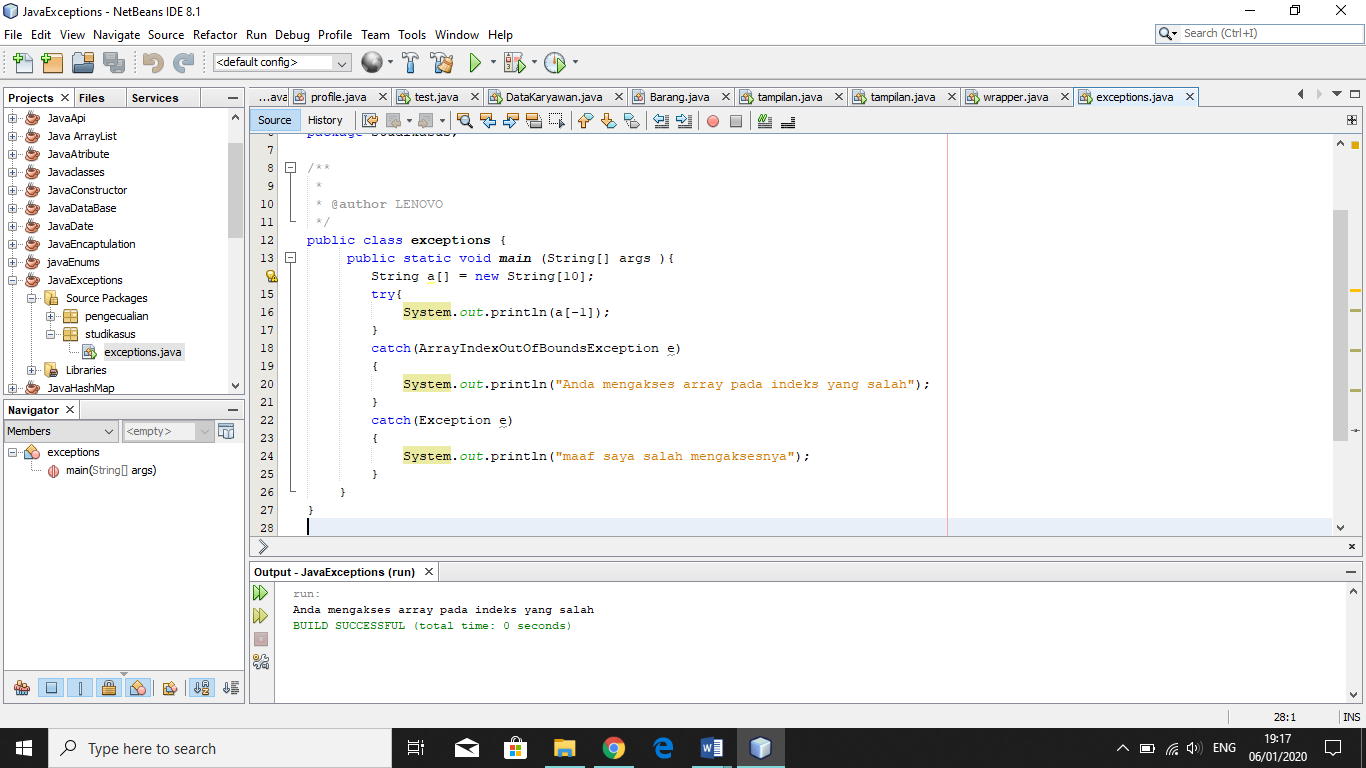
**Penjelasan** : Pada contoh diatas kita melihat bahwa terdapat dua buah tipe data pada input, yaitu Integer dan int.

* Integer merupakan wrapper di dalam java yaitu kata primitif dari tipe data int. Dan variabel a diatas merupakan sebuah objek baru dari tipe data integer dan yang langsung diberi nilai 155.
* int merupakan bilangan bulat yang sering dipakai dalam pemrograman. Pada program diatas tipe data int juga membuat sebuah objek baru dengan nama i yang kemudian memanggil tipe data integer dengan objeknya yaitu a dan kemudian memanggil nilai nya.
* Selanjutnya System.out.println (SOP) mencetak nilai dari tipe data diatas dengan memanggil objek i.
* Dan pada output kita dapat melihat bahwa nilai yang dicetak adalah 155 yaitu nilai dari primitif diatas yaitu integer.

1. **Java Exceptions**

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

Untuk java exception, dibuat sebuah studi kasus yang mengandung error ketika



***Kode Inputannya :***

package studikasus;

/\*\*

\*

\* @author LENOVO

\*/

public class exceptions {

public static void main (String[] args ){

String a[] = new String[10];

try{

System.out.println(a[-1]);

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Anda mengakses array pada indeks yang salah");

}

catch(Exception e)

{

System.out.println("maaf saya salah mengaksesnya");

}

}

}

**BAB V**

**PENUTUP**

1. **KESIMPULAN**

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

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

**DAFTAR PUSTAKA**

W3Schools. 2019. *Java*.

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

(diakses tanggal 24 Desember 2019 pukul 10.09 WIB)