

LAPORAN PRAKTIKUM

BAHASA PEMROGRAMAN JAVA



NIM : 201869040017
NAMA : AVANUL ABDUL FATAH
PRODI : Teknik Informatika 3 A
Dosen : M. Imron Rosyadi, S.Kom., M.Kom.

\

UNIVERSITAS YUDHARTA PASURUAN

Jl. Yudharta No.07 (Pesantren Ngalah) Sengonagung Purwosari

Pasuruan Jawa Timur 67162

Telp/fax : (0343) 611186

Site : <http://www.yudharta.ac.id> | Email : informasi@yudharta.ac.id

KATA PENGANTAR

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

1. Kepada Tuhan Yang Maha Esa yaitu Allah SWT, karena dengan anugerahnya penulis dapat menyelesaikan laporan praktikum ini dengan baik.
2. Orangtua saya yang telah memberikan semangat dan dukungan sehingga laporan praktikum ini dapat terselesaikan dengan baik serta mendapatkan nilai yang diinginkan.
3. M. Imron Rosyadi, S.Kom., M.Kom., selaku dosen Bahasa Pemrograman Java (OOP).
4. Semua pihak yang telah membantu dalam penyusunan laporan ini. Akhirnya penulis mohon maaf apabila ada kekurangan atau kesalahan dalam penyusunan laporan praktikum ini. Semoga laporan praktikum ini dapat bermanfaat bagi penulis, maupun pembacanya.

Purwosari, 12 Januari 2020

Penulis

Avanul Abdul Fatah

DAFTAR ISI

KATA PENGANTAR	1
-----------------------------	----------

DAFTAR ISI.....	2
------------------------	----------

BAB I PENDAHULUAN

1.1 Latar belakang	7
1.2 Tujuan	7
1.3 Manfaat	8

BAB II DASAR TEORI

2.1 Java Methods	9
2.2 Java Methods Parameter	9
2.3 Java Methods Overloading	10
2.4 Java OOP.....	10
2.5 Java Classes / Objects	10

2.6	Java Classes Attributes.....	11
2.7	Java Classes Methods.....	11
2.8	Java Constructors.....	11
2.9	Java Modifiers	12
2.10	Java Encapsulation.....	13
2.11	Java Packages / API.....	14
2.12	Java Inheritance	16
2.13	Java Polymorphism.....	16
2.14	Java Inner Classes.....	16
2.15	Java Abstraction	17
2.16	Java Interface.....	17
2.17	Java Enum	18
2.18	Java User Input	19
2.19	Java Date.....	19
2.20	Java ArrayList	19
2.21	Java HashMap	20

2.22	Java WrapperClasses	20
2.23	Java Exceptions	21

BAB III HASIL PERCOBAAN

3.1	Object.....	22
3.2	Attribute	24
3.3	Method	25
3.4	Constructor.....	27
3.5	Modifier.....	29
3.6	Encapsulation	30
3.7	Package/API.....	31
3.8	Inheritance	33
3.9	Polymorphism.....	36
3.10	InnerClass	37
3.11	Abstraction.....	40
3.12	Interface.....	41
3.13	Enum.....	43

3.14	User Input	45
3.15	Date	47
3.16	ArrayList.....	49
3.17	HashMap	54
3.18	Wrapper Class	58
3.19	Exception.....	60

BAB IV ANALISA HASIL PERCOBAAN

4.1	Analisa Hasil Percobaan Object.....	62
4.2	Analisa Hasil Percobaan Attribute	62
4.3	Analisa Hasil Percobaan Method	62
4.4	Analisa Hasil Percobaan Constructor.....	63
4.5	Analisa Hasil Percobaan Modifier.....	63
4.6	Analisa Hasil Percobaan Encapsulation	64
4.7	Analisa Hasil Percobaan Package/API.....	64
4.8	Analisa Hasil Percobaan Inheritance	65
4.9	Analisa Hasil Percobaan Polymorphism.....	65

4.10	Analisa Hasil Percobaan InnerClass	65
4.11	Analisa Hasil Percobaan Abstraction.....	66
4.12	Analisa Hasil Percobaan Interface.....	66
4.13	Analisa Hasil Percobaan Enum.....	66
4.14	Analisa Hasil Percobaan User Input	66
4.15	Analisa Hasil Percobaan Date	67
4.16	Analisa Hasil Percobaan ArrayList.....	67
4.17	Analisa Hasil Percobaan HashMap	67
4.18	Analisa Hasil Percobaan Wrapper Class	68
4.19	Analisa Hasil Percobaan Exception.....	68

BAB V PENUTUP

Kesimpulan	69
Saran	69

Daftar pustaka

BAB I

PENDAHULUAN

Latar Belakang

Proses penciptaan Java dimulai pada tahun 1991. Java dibuat oleh James Gosling, Mike Sheridan, dan Patrick Naughton. Hingga hari ini, Java dikenal dengan slogan **WORA** (*write once, run anywhere*). Slogan ini digunakan untuk menggambarkan sifat universalitas Java. Coding yang ditulis dengan menggunakan Java dapat digunakan dalam berbagai platform dan situasi.

Konsep dibalik pembuatan Java adalah menghadirkan sebuah bahasa pemrograman yang terorientasi obyek, sederhana, mudah dibaca dan aman. Empat prinsip ini adalah batu penjurul yang menjadi landasan dari pembuatan bahasa Java. Aspek kemudahan untuk dibaca dihadirkan dengan cara membuat sintaks Java mirip dengan sintaks bahasa pemrograman C dan C++. Kemiripan ini dibuat karena bahasa C dan C++ adalah bahasa pemrograman yang paling populer di awal tahun 90an.

Biarpun demikian, bahasa Java juga banyak mendapat kritik. Kebanyakan orang mengatakan bahwa biarpun bahasa pemrograman Java dapat menghadirkan keempat prinsip di atas, ada satu aspek yang tidak dimiliki Java, yaitu peningkatan fungsi. Ada beberapa orang mengklaim bahwa bahasa Java cukup ketinggalan jika dibandingkan dengan beberapa bahasa pemrograman lain yang terus-menerus ditingkatkan. Java sendiri berusaha untuk menjawab kritik tersebut dengan meluncurkan **Java 9**, yaitu versi peningkatan dari bahasa Java yang dilengkapi dengan banyak peningkatan dan inovasi baru.

Tujuan

- Membiasakan untuk dapat membuat sebuah program aplikasi.
- Mengidentifikasi kesalahan pada sebuah program.
- Memahami tentang cara kerja NetBeans IDE.

Manfaat

- Setiap aplikasi maupun program yang dibuat dengan menggunakan dasar Bahasa Pemrograman Java mempunyai kemampuan yang sangat baik untuk dilakukan pengembangan lebih lanjut. Hal ini akan sangat membantu para programmer-programmer dan developer untuk lebih baik lagi dalam mengembangkan satu aplikasi yang berbasis Java.
- Sifatnya Multi-Platform, alias Universal dan dapat digunakan dalam platform apapun. Hal ini membuat banyak sekali para pengembang aplikasi yang menggunakan basis bahasa pemrograman Java ini untuk membuat aplikasi yang diinginkan oleh programmer tersebut.
- Kemampuan aplikasi – aplikasi yang dibuat dengan menggunakan atau berbasis Java yang mampu bekerja di platform manapun. Hal ini berhubungan dengan usability, atau kegunaan dari suatu aplikasi.
- Bahasa pemrograman Java adalah salah satu bentuk atau jenis bahasa pemrograman yang berorientasi terhadap objek. Itu artinya setiap aplikasi yang dibuat dengan menggunakan bahasa pemrograman java akan disesuaikan dengan objek atau dapat juga dengan tampilan dan interface dari aplikasi tersebut.
- Sifat dinamis dari bahasa pemrograman Java ini sangat berkaitan dengan kemampuan dari bahasa pemrograman Java yang sangat mudah untuk dikembangkan. Struktur kodenya dapat dengan mudah dimodifikasi dan dikembangkan, sesuai dengan kebutuhan dari user.

BAB II

DASAR TEORI

Java Methods

Method adalah blok kode yang hanya berjalan ketika dipanggil. Kita dapat mengirimkan data, yang dikenal sebagai parameter, ke dalam suatu Method. Method digunakan untuk melakukan tindakan tertentu, dan mereka juga dikenal sebagai fungsi. Mengapa menggunakan Method? Untuk menggunakan kembali kode: tentukan kode sekali, dan gunakan berkali-kali.

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.

Untuk memanggil Method di Java, tulis nama Method diikuti oleh dua tanda kurung () dan tanda titik koma ;

Java Methods Parameter

Informasi dapat dikirimkan ke Method sebagai **Parameter**. Parameter bertindak sebagai variabel di dalam Method. Parameter ditentukan setelah nama Method, di dalam tanda kurung. Anda dapat menambahkan sebanyak mungkin parameter yang Anda inginkan, cukup pisahkan dengan koma.

Method yang menggunakan **String** yang disebut **fname** sebagai Parameter. Ketika method dipanggil, kami memberikan nama depan, yang digunakan di dalam method untuk mencetak nama lengkap.

Kata kunci **void**, menunjukkan bahwa method tersebut tidak boleh mengembalikan nilai. Jika kita ingin method mengembalikan nilai, kita bisa menggunakan tipe data primitif (seperti **int**, **char**, dll.) Sebagai gantinya **void**, dan gunakan kata kunci **return** di dalam method. Ketika parameter dilewatkan ke method, itu disebut **argument**.

Kita dapat memiliki banyak parameter yang kita inginkan (Multiple Parameter). ketika kita bekerja dengan Multiple Parameter, pemanggilan method harus memiliki jumlah **argument** yang sama karena ada parameter, dan **argument** harus diteruskan dalam urutan yang sama.

Java Methods Overloading

Dengan Method Overloading, beberapa method dapat memiliki nama yang sama dengan parameter yang berbeda. Daripada mendefinisikan dua method yang harus melakukan hal yang sama, lebih baik membebani satu method. kami membebani method **plusMethod** agar bekerja baik untuk **int** maupun **double**. Beberapa method dapat memiliki nama yang sama selama jumlah dan atau tipe parameternya berbeda.

Java OOP

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

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

Pemrograman berorientasi objek memiliki beberapa keunggulan dibandingkan pemrograman prosedural:

- OOP lebih cepat dan lebih mudah untuk dieksekusi
- OOP menyediakan struktur yang jelas untuk program-program tersebut
- OOP membantu menjaga kode Java **DRY "Don't Repeat Yourself"**, dan membuat kode lebih mudah untuk mempertahankan, memodifikasi dan men-debug.

OOP memungkinkan untuk membuat aplikasi yang dapat digunakan kembali secara penuh dengan kode yang lebih sedikit dan waktu pengembangan yang lebih singkat. Prinsip "**Don't Repeat Yourself**" (**DRY**) adalah tentang mengurangi pengulangan kode. Anda harus mengekstrak kode yang umum untuk aplikasi, dan menempatkannya di satu tempat dan menggunakannya kembali alih-alih mengulanginya.

Java Classes / Objects

Classes dan **Objects** adalah dua aspek utama dari **Pemrograman Berorientasi Objek**. Jadi, **Classes** adalah template untuk **Objects**, dan **Objects** adalah turunan dari **Classes**. Ketika **Objects** individu dibuat, mereka mewarisi semua variabel dan method dari **Classes**.

Segala sesuatu di Java dikaitkan dengan **Classes** dan **Objects**, bersama dengan atribut dan methodnya. Sebagai contoh : dalam kehidupan nyata, mobil adalah **Objects**. Mobil memiliki atribut, seperti berat dan warna, dan metode, seperti drive dan rem. Untuk membuat **Classes**, gunakan kata kunci **class**. Suatu kelas harus selalu dimulai dengan huruf pertama huruf besar,

dan bahwa nama file java harus cocok dengan nama kelas. Di Java, **Objects** dibuat dari kelas. Untuk membuat objek **MyClass**, tentukan nama **Classes**, diikuti dengan nama **Objects**, dan gunakan kata kunci **new**. Kita juga dapat membuat beberapa objek dari satu kelas. Anda juga bisa membuat objek kelas dan mengaksesnya di kelas lain. Ini sering digunakan untuk organisasi kelas yang lebih baik (satu kelas memiliki semua atribut dan metode, sedangkan kelas lainnya memegang metode **main()** (**kode yang akan dieksekusi**)).

Java Class Attributes

Class Attributes adalah variabel dalam suatu kelas. Istilah lain untuk **Class Attributes** adalah **fields**. Kita bisa mengakses atribut dengan membuat objek kelas, dan dengan menggunakan sintaks **dot** (.). Kita juga dapat mengubah nilai atribut atau menimpa nilai yang ada dan jika kita tidak ingin kemampuan untuk menimpa nilai yang ada, nyatakan atribut sebagai **final**. Kata kunci **final** berguna ketika Anda ingin variabel selalu menyimpan nilai yang sama, seperti PI (3.14159 ...). Kata kunci **final** disebut dalam "Modifier". Jika kita membuat beberapa objek dari satu kelas, Anda bisa mengubah nilai atribut di satu objek, tanpa memengaruhi nilai atribut di yang lain dan kita dapat menentukan atribut sebanyak yang kita inginkan.

Java Class Methods

Methods dideklarasikan dalam kelas, dan mereka digunakan untuk melakukan tindakan tertentu. **myMethod()** mencetak teks (aksi), ketika **dipanggil**. Untuk memanggil **Methods**, tulis nama **Methods** diikuti oleh dua tanda **kurung** () dan tanda **titik koma** ; Anda akan sering melihat program Java yang memiliki atribut dan method **static** atau **public**. metode **static**, yang berarti dapat diakses tanpa membuat objek kelas, tidak seperti **public**, yang hanya dapat diakses oleh objek. **dot** (.) Digunakan untuk mengakses atribut dan method objek.

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

Java Constructors

Constructors di Java adalah **method khusus** yang digunakan untuk menginisialisasi objek. **Constructors** dipanggil ketika objek kelas dibuat. Ini dapat digunakan untuk mengatur nilai awal untuk atribut objek. Nama **Constructors** harus cocok

dengan nama kelas, dan tidak boleh memiliki tipe kembali (seperti **void**). Perhatikan juga bahwa **Constructors** dipanggil saat objek dibuat. Semua kelas memiliki **Constructors** secara default: jika kita tidak membuat konstruktor kelas sendiri, Java membuat satu untuk kita. Namun, maka kita tidak dapat menetapkan nilai awal untuk atribut objek. **Constructors** juga dapat mengambil **parameter**, yang digunakan untuk menginisialisasi atribut.

Kita dapat menambahkan parameter **int** ke **Constructors**. Di dalam **Constructors** kita atur **x** ke **y** (**x = y**). Ketika kita memanggil **Constructors**, kita meneruskan **parameter** ke **Constructors** (5), yang akan menetapkan nilai **x** ke **5**. Kita juga dapat memiliki banyak **parameter** yang kita inginkan.

Java Modifiers

Kata kunci **public** adalah **Access Modifiers**, artinya digunakan untuk mengatur tingkat akses untuk **classes**, **attributes**, **method**, dan **constructors**.

Kami membagi **Modifier** menjadi dua kelompok:

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

a) Access Modifiers

Untuk **classes**, Anda dapat menggunakan **public** atau **default**.

public - **classes** dapat diakses oleh kelas lain

default - **classes** hanya dapat diakses oleh kelas dalam paket yang sama. Ini digunakan ketika Anda tidak menentukan **Modifiers**.

Untuk **classes**, **attributes**, **method**, dan **constructors**, kita dapat menggunakan salah satunya. **public** - kode ini dapat diakses untuk semua kelas

private - kode hanya dapat diakses di dalam kelas yang dideklarasikan

default - kode hanya dapat diakses dalam paket yang sama. Ini digunakan ketika Anda tidak menentukan **Modifier**

protected - kode ini dapat diakses dalam paket dan **subclasses** yang sama.

b) Non-Access Modifiers

untuk **classes**, Anda dapat menggunakan **final** atau **abstract**.

final - **classes** tidak dapat diwarisi oleh kelas lain.

abstract - **classes** tidak dapat digunakan untuk membuat objek. Untuk mengakses kelas abstrak, itu harus diwarisi dari kelas lain.

Untuk **attributes** dan **method**, kita dapat menggunakan salah satunya.

final - **attributes** dan **method** tidak dapat diganti / dimodifikasi.

static - **attributes** dan **method** milik kelas, bukan objek.

abstract - hanya dapat digunakan dalam kelas **abstract**, dan hanya dapat digunakan pada **method**. **Method** ini tidak memiliki tubuh, misalnya **abstract void run();**. Tubuh disediakan oleh **subclass** (diwarisi dari).

transient - **attributes** dan **method** dilewati saat membuat serialisasi objek yang memuatnya.

synchronized - **method** yang hanya dapat diakses oleh satu utas pada satu waktu.

volatile - nilai dari suatu **attributes** tidak di-cache thread-lokal, dan selalu dibaca dari "memori utama".

Jika kita tidak ingin kemampuan untuk menimpa nilai **attributes** yang ada, nyatakan **attributes** sebagai **final**. **Method static** berarti dapat diakses tanpa membuat objek kelas, tidak seperti **public**. **Method abstract** milik kelas **abstract**, dan tidak memiliki tubuh. Tubuh disediakan oleh subclass.

Java Encapsulation

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

mendeklarasikan variabel / atribut kelas sebagai **private**.

memberikan **method get** dan **set public** untuk mengakses dan memperbarui nilai variabel **private**. variabel **private** hanya dapat diakses di dalam kelas yang sama (kelas luar tidak memiliki akses ke sana). Namun, dimungkinkan untuk mengaksesnya jika kami menyediakan **method** mendapatkan dan mengatur **public**.

Method get mengembalikan nilai variabel, dan **method set** menetapkan nilai.

Syntax untuk keduanya adalah bahwa mereka mulai dengan **get** atau **set**, diikuti dengan variabel **name**, dengan huruf pertama dalam huruf besar. **Method get** mengembalikan nilai variabel **name**.

Method yang ditetapkan mengambil parameter (**newName**) dan menetakannya ke variabel **name**. Kata kunci **This** digunakan untuk merujuk ke objek saat ini.

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

Jika variabel dinyatakan sebagai **public**, kita akan mendapatkan output yang benar. Namun, saat kita mencoba mengakses variabel **private**, kita mendapatkan kesalahan. Sebagai gantinya, kita menggunakan method **getName()** dan **setName()** untuk mengakses dan memperbarui variabel. Kelebihan **Encapsulation** :

- Kontrol **attributes** dan **method** kelas yang lebih baik
- **Attributes** kelas dapat dibuat **read-only** (jika Anda hanya menggunakan method **get**), atau **write-only** (jika Anda hanya menggunakan method **set**).
- Fleksibel : programmer dapat mengubah satu bagian kode tanpa mempengaruhi bagian lainnya
- Peningkatan keamanan data.

Java Packages / API

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

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

a) Built-in Packages

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

Perpustakaan berisi komponen untuk mengelola input, pemrograman basis data, dan banyak lagi lainnya. Daftar lengkap dapat ditemukan di situs web Oracles: <https://docs.oracle.com/javase/8/docs/api/>.

Perpustakaan dibagi menjadi beberapa **packages** dan **classes**. Berarti kita 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, kita perlu menggunakan kata kunci **import**.

Jika kita menemukan kelas yang ingin kita gunakan, misalnya, kelas **Scanner**, yang digunakan untuk mendapatkan input pengguna.

java.util adalah sebuah paket, sedangkan **Scanner** adalah kelas dari paket **java.util**.

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

Ada banyak paket untuk dipilih. Paket ini juga berisi fasilitas tanggal dan waktu, generator nomor acak, dan kelas utilitas lainnya.

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

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

```
└── root
    └── mypack
        └── MyPackageClass.java
```

Untuk membuat paket, gunakan kata kunci **package**.

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

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

Kemudian kompilasi paket:

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

Ini memaksa kompiler untuk membuat paket "mypack".

Kata kunci **-d** menentukan tujuan tempat penyimpanan file kelas. Kita dapat menggunakan nama direktori apa saja, seperti c: / user (windows), atau, jika kita ingin menyimpan paket dalam direktori yang sama, kita dapat menggunakan tanda titik "." Nama paket harus ditulis dalam huruf kecil untuk menghindari konflik dengan nama kelas.

Ketika kita mengompilasi paket, folder baru dibuat, disebut "mypack".

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

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

Outputnya pasti akan seperti apa yang kita input tadi.

Java Inheritance

Di Java, dimungkinkan untuk mewarisi **attributes** dan **method** dari satu kelas ke kelas lain. Kita mengelompokkan "konsep pewarisan" ke dalam dua kategori:

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

Untuk mewarisi dari kelas, gunakan kata kunci **extends**. Jika kita tidak ingin kelas lain mewarisi dari suatu kelas, gunakan kata kunci **final**.

Java Polymorphism

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

Inheritance memungkinkan kita mewarisi atribut dan metode dari kelas lain. **Polymorphism** menggunakan metode-metode itu untuk melakukan tugas yang berbeda. Ini memungkinkan kita untuk melakukan satu tindakan dengan berbagai cara.

Kelebihan menggunakan "**Inheritance**" dan "**Polymorphism**" adalah berguna untuk penggunaan kembali kode: menggunakan kembali atribut dan method dari kelas yang ada saat kita membuat kelas baru.

Java Inner Classes

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

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

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

Jika kita mencoba mengakses kelas dalam **private** dari kelas luar (MyMainClass), kesalahan terjadi.

Kelas dalam juga bisa **static**, yang berarti bahwa kita dapat mengaksesnya tanpa membuat objek dari kelas luar.

seperti atribut dan method **static**, kelas dalam **static** tidak memiliki akses ke anggota kelas luar.

Satu keuntungan dari kelas dalam, adalah mereka dapat mengakses atribut dan method dari kelas luar.

Java Abstraction

Data Abstraction adalah proses menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna.

Abstraction dapat dicapai dengan **abstract classes** atau **interfaces**.

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

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

Abstract Class dapat memiliki metode abstrak dan reguler. Kelebihan Menggunakan **Abstract Class** dan **Abstract Method** adalah untuk mencapai keamanan (sembunyikan detail tertentu dan hanya tampilkan detail penting suatu objek).

Java Interface

Cara lain untuk mencapai **abstraction** di Java, adalah dengan **Interface**.

Interface adalah "**abstract class**" yang sepenuhnya digunakan untuk mengelompokkan method terkait dengan benda kosong.

Untuk mengakses method **Interface**, **Interface** harus "diimplementasikan" (agak seperti diwariskan) oleh kelas lain dengan kata kunci **implements** (sebagai gantinya **extends**). Tubuh method **Interface** disediakan oleh kelas "implement".

Interface:

Seperti **abstract class**, **Interface** tidak dapat digunakan untuk membuat objek (dalam contoh di atas, tidak mungkin membuat objek "Hewan" di MyMainClass)

Metode antarmuka tidak memiliki tubuh - tubuh disediakan oleh kelas "implement"

Pada implementasi **Interface**, Anda harus mengganti semua methodnya

Method **Interface** secara default **abstract** dan **public**

Atribut **Interface** secara default **public**, **static** dan **final**

Interface tidak dapat berisi **Constructor** (karena tidak dapat digunakan untuk membuat objek).

Kelebihan Menggunakan **Interface** :

- 1) Untuk mencapai keamanan - sembunyikan detail tertentu dan hanya tampilkan detail penting dari suatu objek.
- 2) Java tidak mendukung "**multiple inheritance**" (sebuah kelas hanya dapat diwarisi dari satu superclass). Namun, ini dapat dicapai dengan **Interface**, karena kelas dapat mengimplementasikan **Multiple Interfaces**. Untuk mengimplementasikan **Multiple Interfaces**, pisahkan dengan koma.

Java Enums

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

Untuk membuat **enum**, gunakan kata kunci **enum** (bukan **class** atau **interface**), dan pisahkan **constant** dengan koma. Perhatikan bahwa mereka harus dalam huruf besar.

Kita dapat mengakses **enum constant** dengan sintaks **dot**.

Enum adalah kependekan dari "**enumerasi**", yang berarti "terdaftar secara khusus". Kita juga dapat memiliki **enum** di dalam kelas. **Enum** sering digunakan dalam pernyataan **switch** untuk memeriksa nilai yang sesuai. Tipe **enum** memiliki method **values()**, yang mengembalikan array dari semua **enum constants**. Method ini berguna ketika kita ingin mengulangi **Enum Constant**.

Perbedaan antara **Enums** dan **Class** :

Enum dapat seperti halnya **Class**, memiliki atribut dan method. Satu-satunya perbedaan adalah bahwa **Enum Constanta** bersifat **public**, **static**, dan **final** (tidak dapat diubah - tidak dapat diganti). **Enum** tidak dapat digunakan untuk membuat objek, dan itu tidak dapat memperluas kelas lain (tetapi dapat mengimplementasikan **Interface**). Kelebihan menggunakan **Enum** adalah dapat digunakan ketika kita memiliki nilai yang kita tahu tidak akan berubah, seperti bulan, hari, warna, setumpuk kartu, dll.

Java User Input

Scanner Class digunakan untuk mendapatkan **User Input**, dan ditemukan dalam **package java.util**. Untuk menggunakan **Scanner Class**, buat objek kelas dan gunakan salah satu method yang tersedia yang ditemukan dalam dokumentasi **Scanner Class**.

Java Date

Java tidak memiliki kelas **Date** bawaan, tetapi kita dapat mengimpor paket **java.time** untuk bekerja dengan API tanggal dan waktu. Paket termasuk banyak kelas tanggal dan waktu.

LocalDate : Merupakan tanggal (tahun, bulan, hari (yyyy-MM-dd))

LocalTime : Merupakan waktu (jam, menit, detik, dan milidetik (HH-mm-dt-zzz))

LocalDateTime : Merupakan tanggal dan waktu (yyyy-MM-dd-HH-mm-ss.zzz)

DateTimeFormatter : Formatter untuk menampilkan dan mem-parsing objek tanggal-waktu.

Untuk menampilkan tanggal saat ini, impor kelas **java.time.LocalDate**, dan gunakan method **now()**. Untuk menampilkan waktu saat ini (jam, menit, detik, dan milidetik), impor kelas **java.time.LocalTime**, dan gunakan method **now()**. Untuk menampilkan tanggal dan waktu saat ini, impor kelas **java.time.LocalDateTime**, dan gunakan method **now()**. Kita bisa menggunakan kelas **DateTimeFormatter** dengan metode **ofPattern()** dalam paket yang sama untuk memformat atau mem-parsing objek tanggal-waktu. Metode **ofPattern()** menerima semua jenis nilai, jika kita ingin menampilkan tanggal dan waktu dalam format yang berbeda.

Java ArrayList

ArrayList class adalah resizable array, yang dapat ditemukan dalam paket **java.util**. Perbedaan antara array bawaan dan **ArrayList** di Java, adalah bahwa ukuran array tidak dapat dimodifikasi (jika kita ingin menambah atau menghapus elemen ke / dari array, kita harus membuat yang baru). Sementara elemen dapat ditambahkan dan dihapus dari **ArrayList** kapan pun kita mau. Sintaksnya juga sedikit berbeda. **ArrayList class** memiliki banyak method yang berguna. Misalnya, untuk menambahkan elemen ke **ArrayList**, gunakan method **add()**. Untuk mengakses elemen di **ArrayList**, gunakan method **get()** dan lihat nomor indeks. Indeks array dimulai dengan 0: [0] adalah elemen pertama. [1] adalah elemen kedua, dll. Untuk memodifikasi elemen, gunakan method **set()** dan lihat nomor

indeks. Untuk menghapus elemen, gunakan method **remove()** dan lihat nomor indeks. Untuk menghapus semua elemen di **ArrayList**, gunakan method **clear()**. Untuk mengetahui berapa banyak elemen yang dimiliki **ArrayList**, gunakan method **size**. Loop melalui elemen **ArrayList** dengan **for** loop, dan gunakan method **size()** untuk menentukan berapa kali loop harus dijalankan. Kita juga dapat mengulang melalui **ArrayList** dengan **for-each** loop. Elemen dalam **ArrayList** sebenarnya adalah objek. Untuk menggunakan tipe lain, seperti **int**, kita harus menentukan **wrapper classes**: **Integer**. Untuk tipe primitif lainnya, gunakan: Boolean untuk **boolean**, Karakter untuk **char**, Double untuk **double**, dll. Kelas lain yang berguna dalam paket **java.util** adalah kelas **Collections**, yang mencakup method **sort()** untuk menyortir daftar berdasarkan abjad atau numerik.

Java HashMap

HashMap, menyimpan item dalam pasangan "**kunci / nilai**", dan kita dapat mengaksesnya dengan indeks jenis lain (mis. **String**). Satu objek digunakan sebagai kunci (indeks) ke objek lain (nilai). Ini dapat menyimpan berbagai jenis: Kunci **String** dan nilai **Integer**, atau tipe yang sama, seperti: Kunci **String** dan nilai **String**. **HashMap class** memiliki banyak method yang berguna. Misalnya, untuk menambahkan item ke dalamnya, gunakan method **put()**. Untuk mengakses nilai di **HashMap**, gunakan method **get()** dan lihat kuncinya. Untuk menghapus item, gunakan method **remove()** dan lihat kuncinya. Untuk menghapus semua item, gunakan method **clear()**. Untuk mengetahui berapa banyak item yang ada, gunakan method **size**. Ulangi item-item dari **HashMap** dengan **for-each** loop. Gunakan method **keySet()** jika kita hanya menginginkan kunci, dan gunakan method **values()** jika kita hanya menginginkan nilai. Kunci dan nilai dalam **HashMap** sebenarnya adalah objek. Untuk menggunakan tipe lain, seperti **int**, kita harus menentukan wrapper class yang setara: **Integer**. Untuk jenis primitif lainnya, gunakan: Boolean untuk **boolean**, Karakter untuk **char**, Double untuk **double**, dll.

Java Wrapper Classes

Wrapper Classes menyediakan cara untuk menggunakan tipe data primitif (**int**, **boolean**, dll.) sebagai objek. Di bawah ini menunjukkan tipe primitif dan **Wrapper Classes** yang setara.

- byte - Byte
- int - Integer
- long - Long

float - Float

double - Double

boolean - Boolean

char - Character

Kadang-kadang kita harus menggunakan **Wrapper Classes**, misalnya saat bekerja dengan objek Koleksi, seperti **ArrayList**, di mana tipe primitif tidak dapat digunakan (daftar hanya dapat menyimpan objek). Untuk membuat objek wrapper, gunakan **Wrapper Classes** alih-alih tipe primitif. Untuk mendapatkan nilai, kita cukup mencetak objek. Karena kita sekarang bekerja dengan objek, kita dapat menggunakan method tertentu untuk mendapatkan informasi tentang objek tertentu.

Sebagai contoh, method berikut ini digunakan untuk mendapatkan nilai yang terkait dengan objek wrapper yang sesuai: **intValue()**, **byteValue()**, **shortValue()**, **longValue()**, **floatValue()**, **doubleValue()**, **charValue()**, **booleanValue()**. Method lain yang bermanfaat adalah method **toString()**, yang digunakan untuk mengubah objek wrapper menjadi string.

Java Exceptions

Saat menjalankan kode Java, kesalahan yang berbeda dapat terjadi: kesalahan pengkodean yang dibuat oleh programmer, kesalahan karena input yang salah, atau hal-hal yang tidak terduga lainnya. Ketika kesalahan terjadi, Java biasanya akan berhenti dan menghasilkan pesan kesalahan. Istilah teknis untuk ini adalah: Java akan melempar **exception** (melempar kesalahan).

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

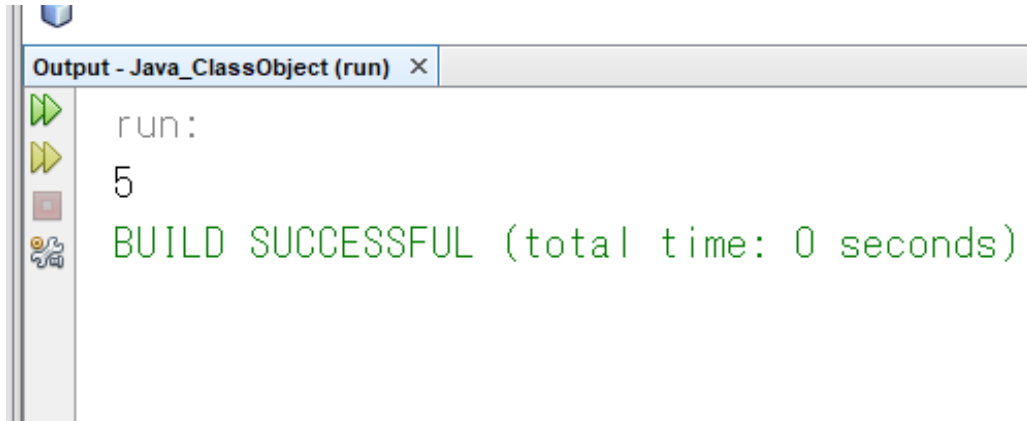
Pernyataan **catch** memungkinkan kita untuk menentukan blok kode yang akan dieksekusi, jika kesalahan terjadi di blok coba.

Kata kunci **try** dan **catch** berpasangan. Jika terjadi kesalahan, kita dapat menggunakan **try...catch** untuk menangkap kesalahan dan menjalankan beberapa kode untuk menanganinya. Pernyataan **finally** memungkinkan kita menjalankan kode, setelah **try...catch**, terlepas dari hasilnya. Pernyataan **throw** memungkinkan kita untuk membuat kesalahan khusus. Pernyataan **throw** digunakan bersama dengan **exception type**. Ada banyak jenis pengecualian yang tersedia di Java: **ArithmeticException**, **FileNotFoundException**, **ArrayIndexOutOfBoundsException**, **SecurityException**, dll.

BAB III

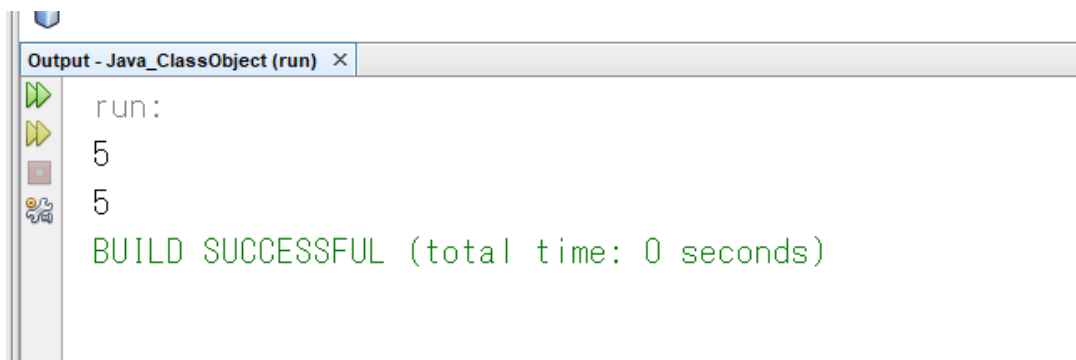
HASIL PERCOBAAN

3.1 Object



a)

```
public class Java_ClassObject {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Java_ClassObject myObj = new Java_ClassObject();  
        System.out.println(myObj.x);  
    }  
}
```



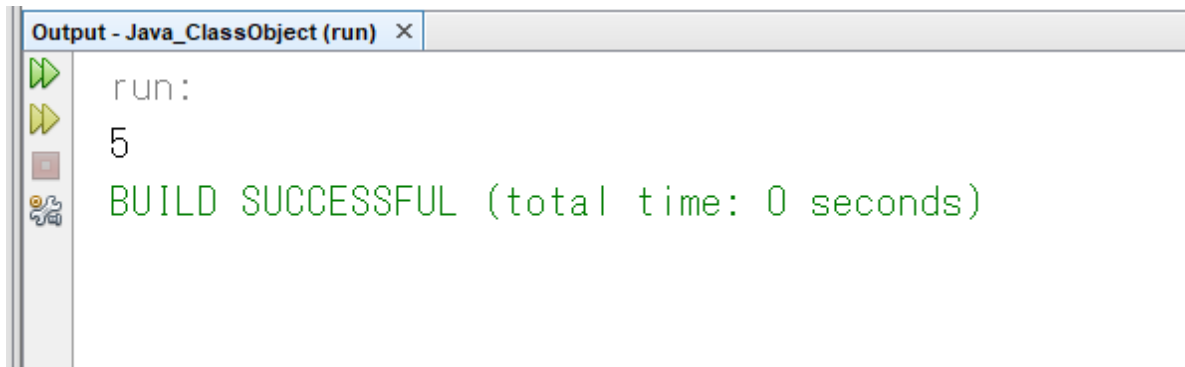
b)

```
public class Multiple_Object {  
    int x = 5;
```

```

public static void main(String[] args) {
    Multiple_Object myObj1 = new Multiple_Object(); // Object 1
    Multiple_Object myObj2 = new Multiple_Object(); // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
}
}

```



c)

//MainClass: OtherClass.java

```

class OtherClass {
    public static void main(String[] args) {
        Multiple_Class myObj = new Multiple_Class();
        System.out.println(myObj.x);
    }
}

```

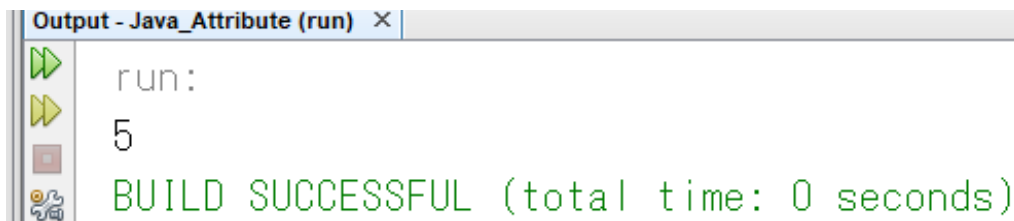
//SecondClass: Multiple_Class.java

```

public class Multiple_Class {
    int x = 5;
}

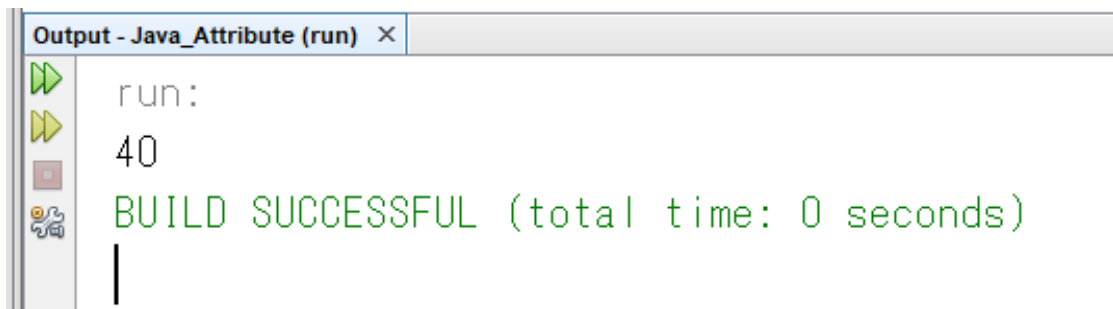
```


3.2 Attribute



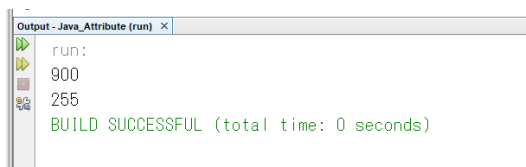
a)

```
public class Acc_attribute {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Acc_attribute myObj = new Acc_attribute();  
        System.out.println(myObj.x);  
    }  
}
```



b)

```
public class Mod_attribute {  
    int x;  
  
    public static void main(String[] args) {  
        Mod_attribute myObj = new Mod_attribute();  
        myObj.x = 40;  
        System.out.println(myObj.x);  
    }  
}
```



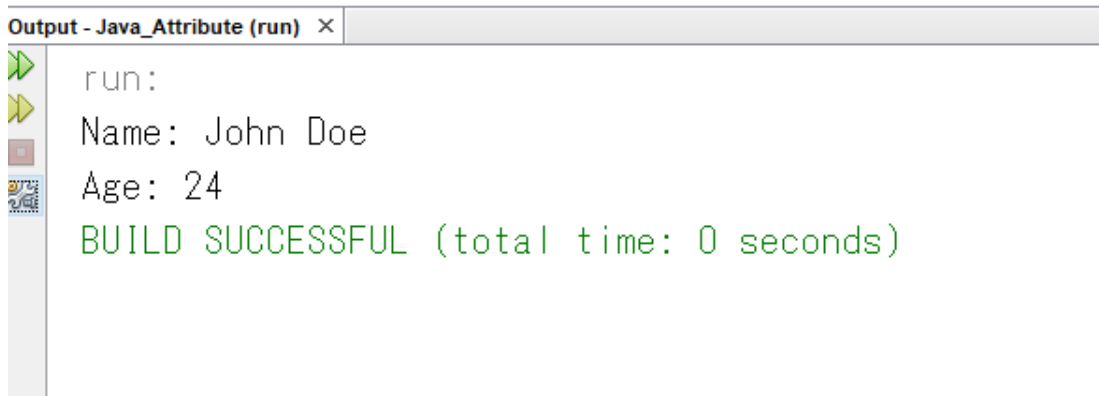
c)

```
public class Multiple_object {  
    int x = 900;  
  
    public static void main(String[] args) {  
        Multiple_object myObj1 = new Multiple_object(); // Object 1  
        Multiple_object myObj2 = new Multiple_object(); // Object 2
```

```

myObj2.x = 255;
System.out.println(myObj1.x); // Outputs 900
System.out.println(myObj2.x); // Outputs 255
}
}

```



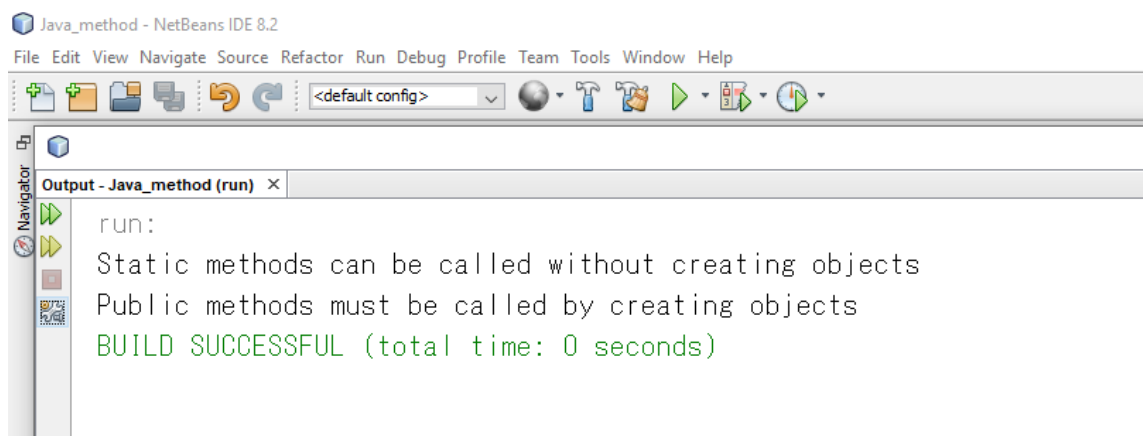
d)

```

public class Multiple_Attribute {
    String fname = "John";
    String lname = "Doe";
    int age = 24;
    public static void main(String[] args) {
        Multiple_Attribute myObj = new Multiple_Attribute();
        System.out.println("Name: " + myObj.fname + " " + myObj.lname);
        System.out.println("Age: " + myObj.age);
    }
}

```

3.3 Method



a)

```

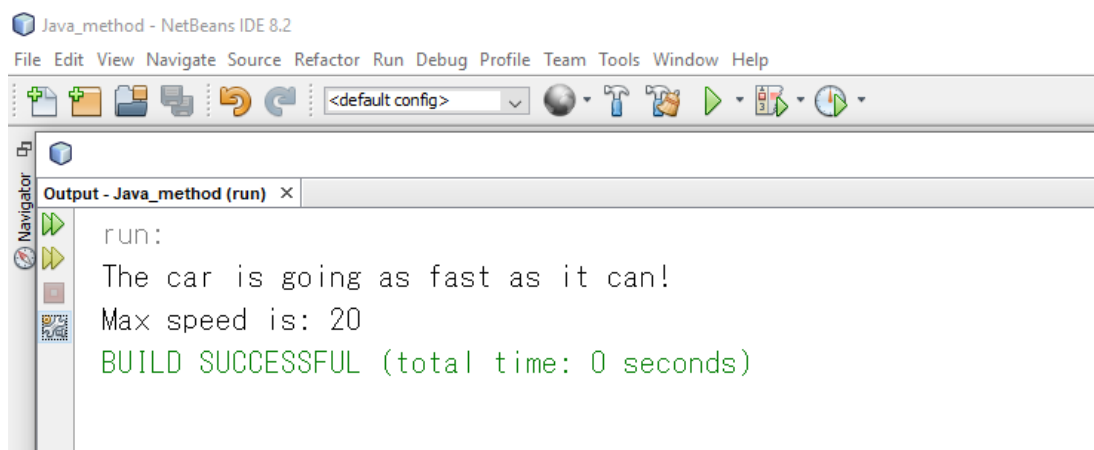
public class MyClass {
    // Static method
    static void myStaticMethod() {
        System.out.println("Static methods can be called without creating objects");
    }

    // 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
    }
}

```



b)

```

// Create a Car class
public class Car {

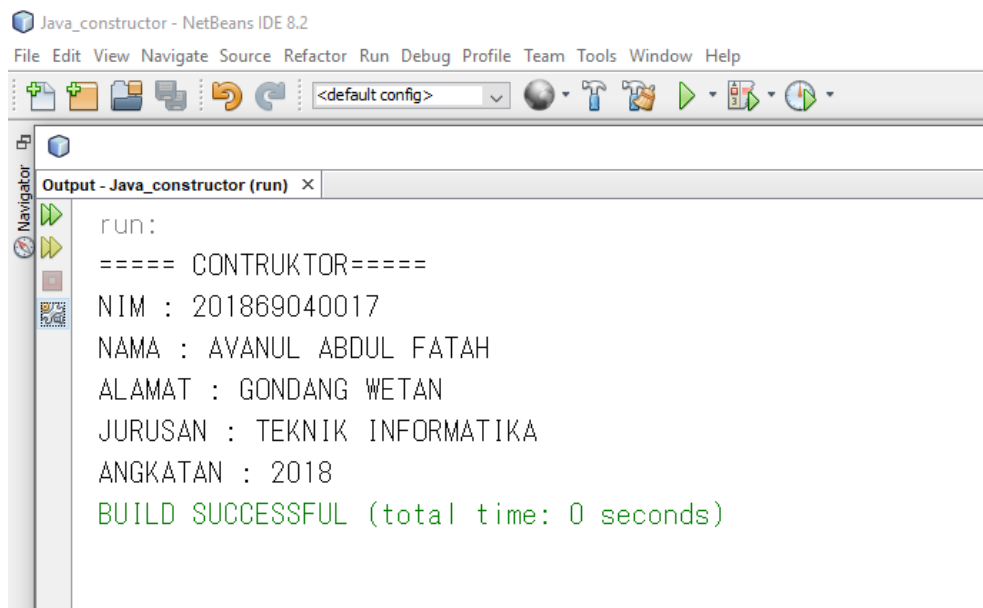
    // Create a fullThrottle() method
    public void fullThrottle() {
        System.out.println("The car is going as fast as it can!");
    }

    // 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(20); // Call the speed() method
    }
}

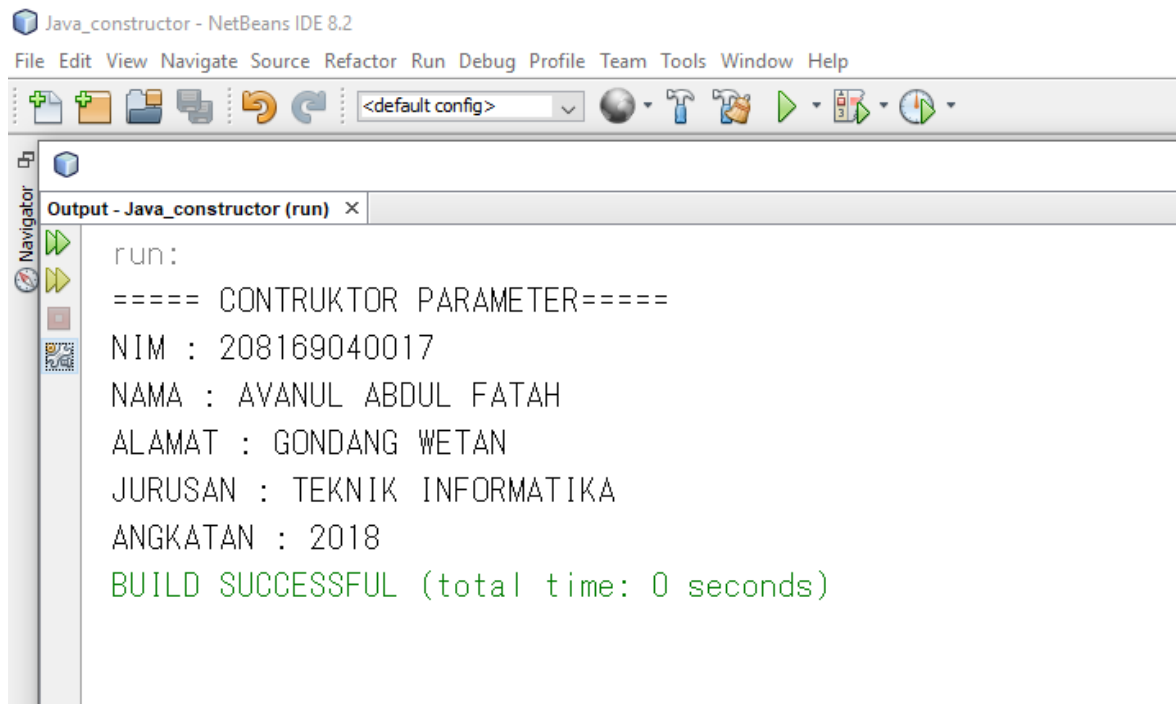
```

3.4 Constructor



a)

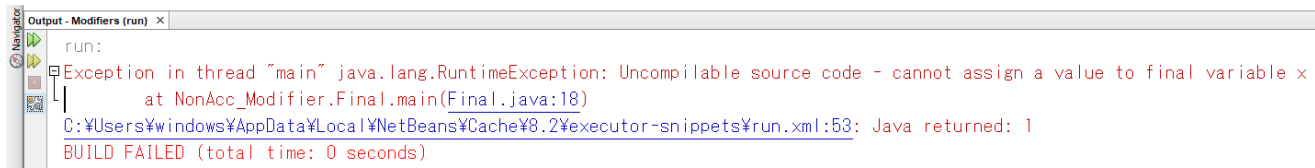
```
public class Constructor {  
    long nim;  
    String jurusan, nama, alamat;  
    int angkatan;  
    public Constructor(){  
        nim=201869040017L;  
        nama="AVANUL ABDUL FATAH";  
        alamat="GONDANG WETAN";  
        jurusan="TEKNIK INFORMATIKA";  
        angkatan=2018;  
    }  
  
    public static void main (String[] args){  
        Constructor data=new Constructor();  
        //CONSTRUCTOR  
        System.out.println("===== CONTRUKTOR=====");  
        System.out.print("NIM : "+data.nim+  
            "\nNAMA : "+data.nama+  
            "\nALAMAT : "+data.alamat+  
            "\nJURUSAN : "+data.jurusan+  
            "\nANGKATAN : "+data.angkatan+"\n");  
    }  
}
```



b)

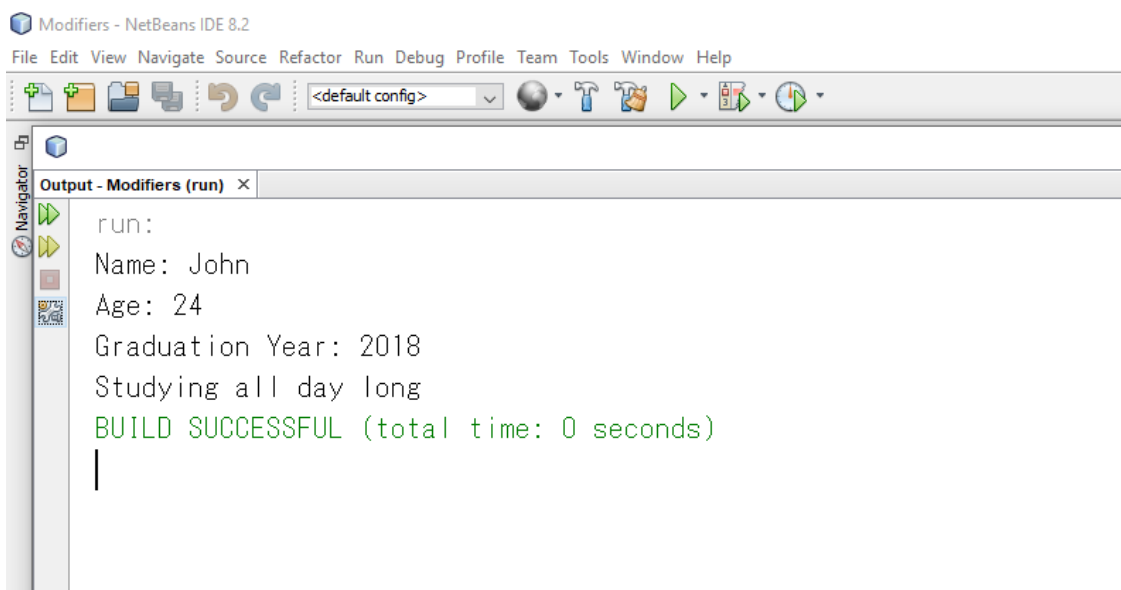
```
public class Constructor_Parameter {
    long nim;
    String jurusan, nama, alamat;
    int angkatan;
    public Constructor_Parameter(long NIM,String NAMA, String ALAMAT,String JUR,int ANGKATAN){
        nim=NIM;
        nama=NAMA;
        alamat=ALAMAT;
        jurusan=JUR;
        angkatan=ANGKATAN;
    }
    public static void main (String[] args){
        //CONSTRACKTOR PARAMETER
        Constructor_Parameter dataParameter=new
            Constructor_Parameter(208169040017L,"AVANUL          ABDUL          FATAH","GONDANG
WETAN","TEKNIK INFORMATIKA",2018);
        System.out.println("===== CONTRUKTOR PARAMETER=====");
        System.out.print(
            "NIM : "+dataParameter.nim+
            "\nNAMA : "+dataParameter.nama+
            "\nALAMAT : "+dataParameter.alamat+
            "\nJURUSAN : "+dataParameter.jurusan+
            "\nANGKATAN : "+dataParameter.angkatan+"\n");
    }
}
```

3.5 Modifier



a)

```
public class Final {  
    final int x = 10;  
    final double PI = 3.14;  
  
    public static void main(String[] args) {  
        Final myObj = new Final();  
        myObj.x = 50; // will generate an error: cannot assign a value to a final variable  
        myObj.PI = 25; // will generate an error: cannot assign a value to a final variable  
        System.out.println(myObj.x);  
        System.out.println(myObj.PI);  
    }  
}
```



b)

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  
    }  
}
```

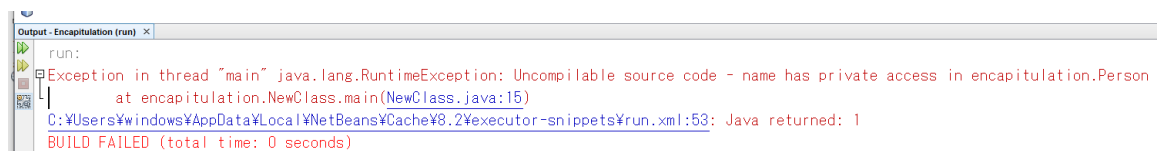
```
}
}
```

Person.java :

```
// abstract class
abstract class Person {
    public String fname = "John";
    public int age = 24;
    public abstract void study(); // abstract method
}

// Subclass (inherit from Person)
class Student extends Person {
    public int graduationYear = 2018;
    public void study() { // the body of the abstract method is provided here
        System.out.println("Studying all day long");
    }
}
```

3.6 Encapsulation

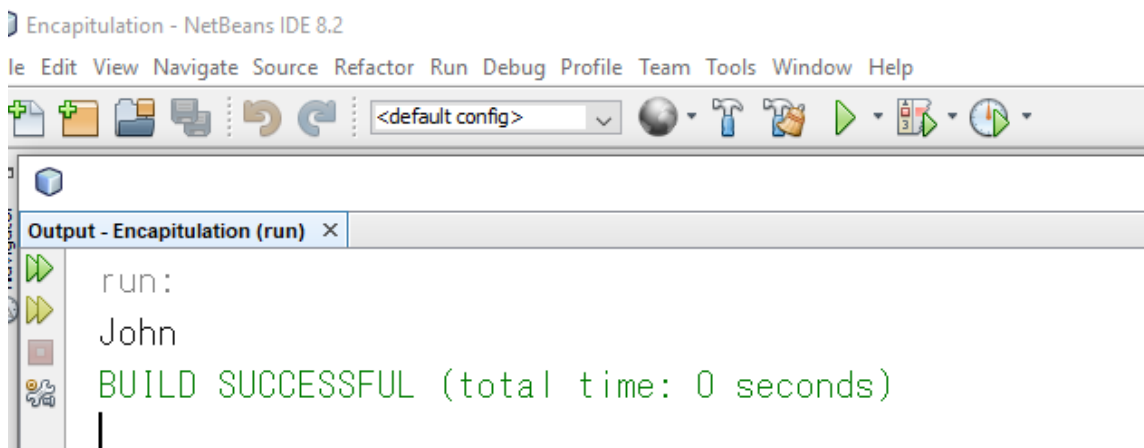


Output - Encapitulation (run) x

```
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - name has private access in encapitulation.Person
    at encapitulation.NewClass.main(NewClass.java:15)
C:\Users\%windows\AppData\Local\%NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

a)

```
public class MyClass {
    public static void main(String[] args) {
        Person myObj = new Person();
        myObj.name = "John";
        System.out.println(myObj.name);
    }
}
```



Encapitulation - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Output - Encapitulation (run) x

```
run:
John
BUILD SUCCESSFUL (total time: 0 seconds)
```

b)

MyClass.java :

```
public class MyClass {
    public static void main(String[] args) {
        Person myObj = new Person();
        myObj.setName("John");
        System.out.println(myObj.getName());
    }
}
```

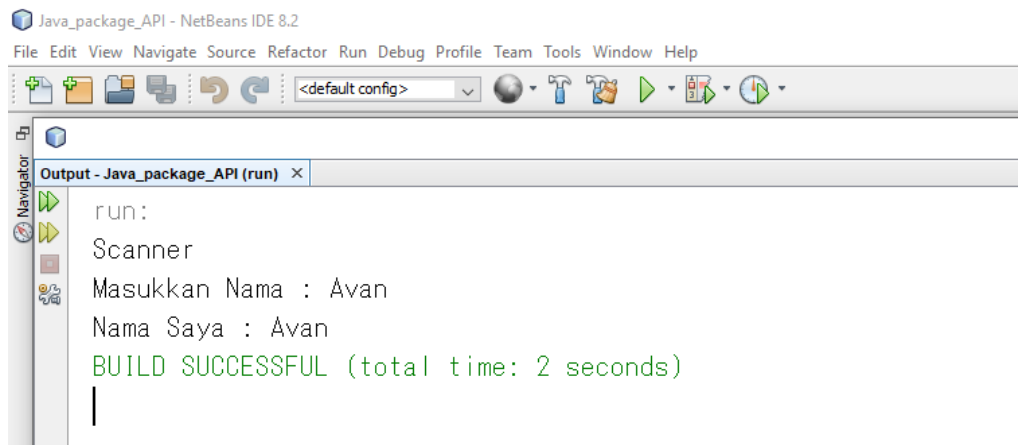
Person.java :

```
public class Person {
    private String name;

    // Getter
    public String getName() {
        return name;
    }

    // Setter
    public void setName(String newName) {
        this.name = newName;
    }
}
```

3.7 Package/API



a)

```
import java.util.Scanner;
```

```
/**
```

```
 *
```

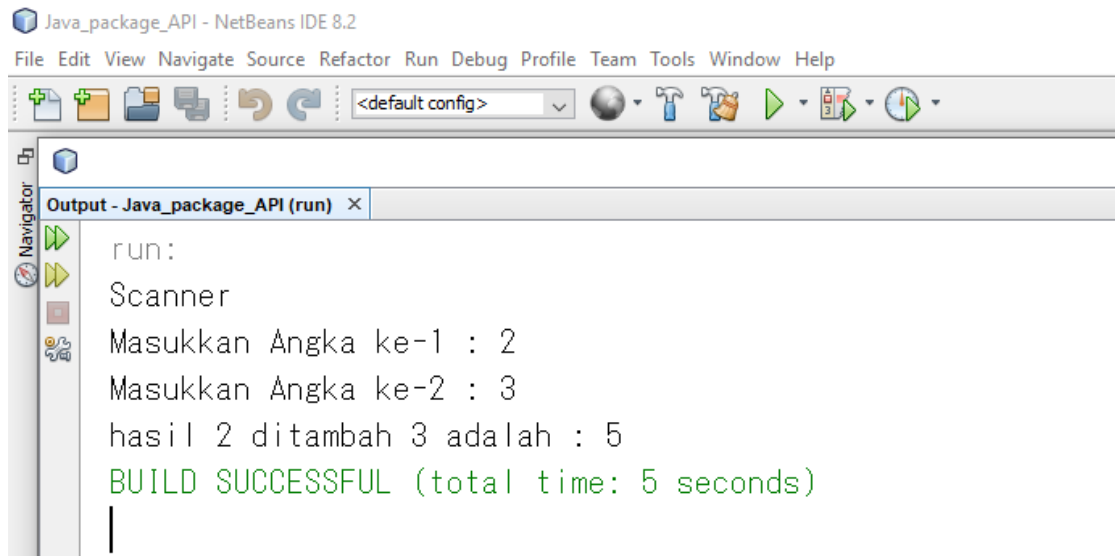
```
 * @author windows
```

```
 */
```

```
public class ContohScanner1 {
    public static void main(String [] args){

        System.out.println("Scanner");

        Scanner input= new Scanner(System.in);
        String nama;
        System.out.print("Masukkan Nama : ");
        nama =input.nextLine();
        System.out.println("Nama Saya : "+nama);
    }
}
```

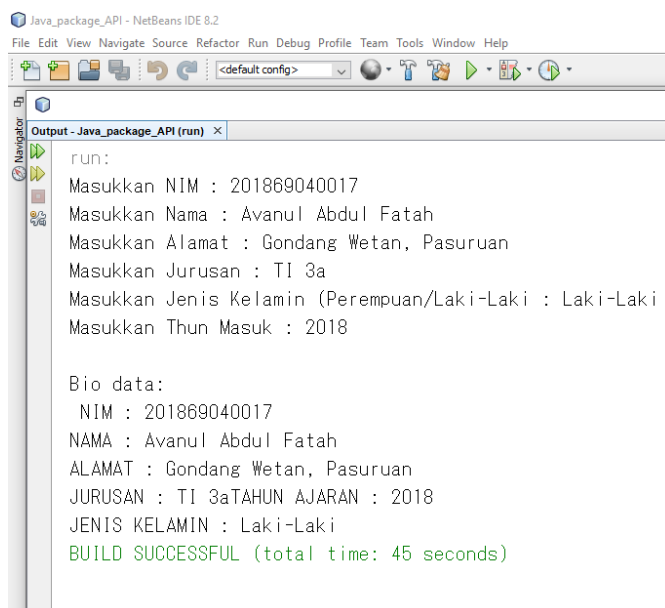



b)

```
import java.util.Scanner;
```

```
/**
 *
 * @author windows
 */
public class ContohScanner2 {
    public static void main(String [] args){

        System.out.println("Scanner");
        Scanner input= new Scanner(System.in);
        int satu,duwa,hasil;
        System.out.print("Masukkan Angka ke-1 : ");
        satu =input.nextInt();
        System.out.print("Masukkan Angka ke-2 : ");
        duwa =input.nextInt();
        hasil=satu+duwa;
        System.out.println("hasil "+satu+" ditambah "+duwa+" adalah : "+hasil);
    }
}
```



c)

```

package API;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
/**
 *
 * @author windows
 */
public class Read {
    public static void main(String []args){
        //BufferedReader tidak bisa menggunakan type data selain String
        BufferedReader input1= new BufferedReader(new InputStreamReader(System.in));
        long NIM;
        int THN;
        String nim, nama, alamat, gender, jurusan, thn;
        try{

            System.out.print("Masukkan NIM : ");
            nim =input1.readLine();

            System.out.print("Masukkan Nama : ");
            nama =input1.readLine();

            System.out.print("Masukkan Alamat : ");
            alamat =input1.readLine();

            System.out.print("Masukkan Jurusan : ");
            jurusan =input1.readLine();

            System.out.print("Masukkan Jenis Kelamin (Perempuan/Laki-Laki : ");
            gender =input1.readLine();

            System.out.print("Masukkan Thun Masuk : ");
            thn =input1.readLine();

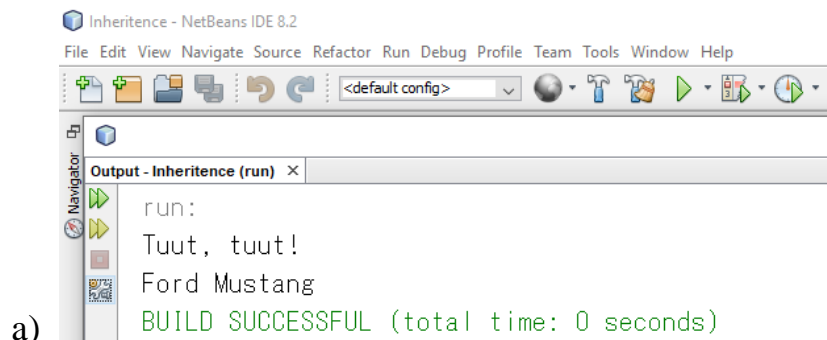
            System.out.println("\nBio data: \n NIM : "+nim+"\nNAMA : "+nama+"\nALAMAT : "+alamat+"\nJURUSAN : "+jurusan+"TAHUN AJARAN : "+thn+"\nJENIS KELAMIN : "+gender);

        }catch(IOException ex){

            System.out.println("Terjadi kesalahan");
        }
    }
}

```

3.8 Inheritance

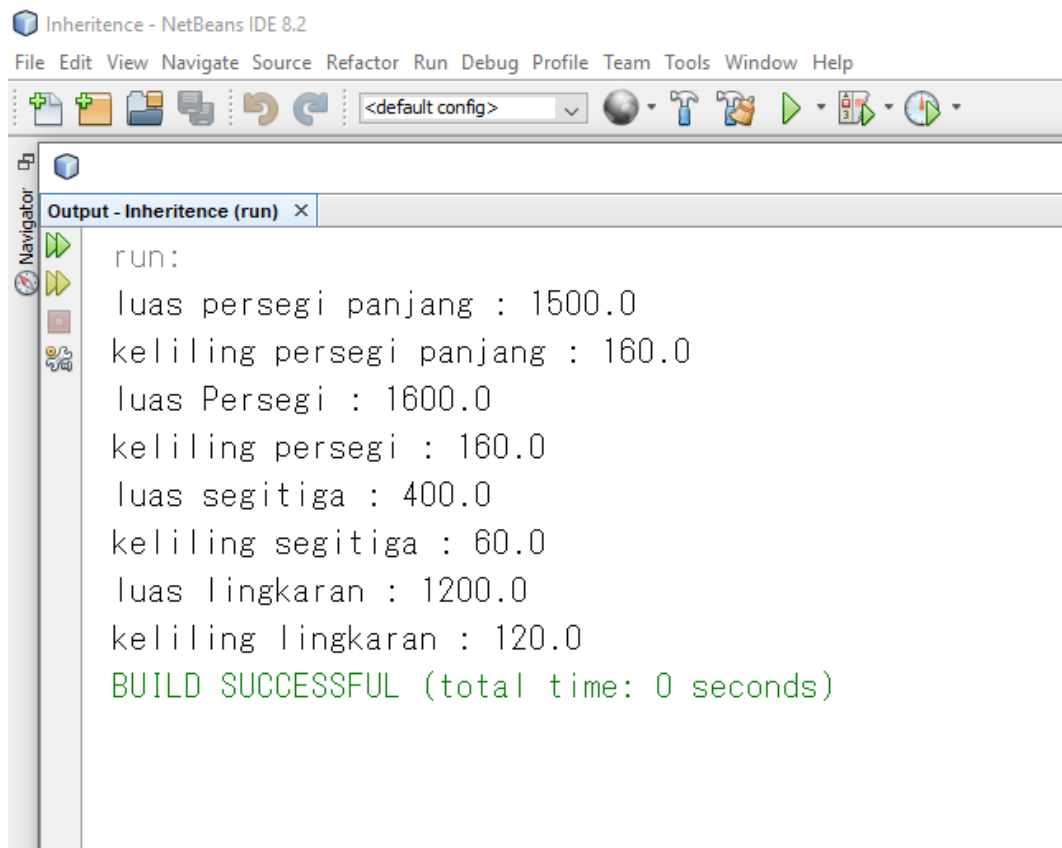


```

class Vehicle {
    protected String brand = "Ford";
    public void honk() {
        System.out.println("Tuut, tuut!");
    }
}

class Car extends Vehicle {
    private String modelName = "Mustang";
    public static void main(String[] args) {
        Car myFastCar = new Car();
        myFastCar.honk();
        System.out.println(myFastCar.brand + " " + myFastCar.modelName);
    }
}

```



b)

```

- Lingkaran.Java
class Lingkaran extends bangun_datar {
    float jari;
    float phi=22/7;
    public void luas(){
        Lingkaran ll =new Lingkaran();
        ll.luas=phi*(jari*jari);
        System.out.println("luas lingkaran : "+ll.luas);
    }
    public void keliling(){
        Lingkaran ll =new Lingkaran();
        ll.keliling=2*phi*jari;
        System.out.println("keliling lingkaran : "+ll.keliling);
    }
}

```

```

-- Persegi.java
class Persegi extends bangun_datar {
    float sisi;
    public void luas(){
        Persegi p1 = new Persegi();
        p1.luas=sisi*sisi;
        System.out.println("luas Persegi : "+p1.luas);
    }
    public void keliling(){
        Persegi p1 = new Persegi();
        p1.keliling= 4*sisi;
        System.out.println("keliling persegi : "+p1.keliling);
    }
}

-- persegi_panjang.java
class Persegi_Panjang extends bangun_datar {
    float panjang;
    float lebar;
    public void luas(){
        Persegi_Panjang pp =new Persegi_Panjang();
        pp.luas=panjang*lebar;
        System.out.println("luas persegi panjang : "+pp.luas);
    }
    public void keliling(){
        Persegi_Panjang pp =new Persegi_Panjang();
        pp.keliling=2*(panjang+lebar);
        System.out.println("keliling persegi panjang : "+pp.keliling);
    }
}

-- Segitiga.java
class Segitiga extends bangun_datar {
    float tinggi;
    float alas;
    public void luas(){
        Segitiga ss = new Segitiga();
        ss.luas=alas*tinggi/2;
        System.out.println("luas segitiga : "+ss.luas);
    }
    public void keliling(){
        Segitiga ss = new Segitiga();
        ss.keliling=alas*3;
        System.out.println("keliling segitiga : "+ss.keliling);
    }
}

-- Bangun_Datar.java
class bangun_datar {
    float luas;
    float keliling;

    public static void main(String[] args) {
        // TODO code application logic here
        Persegi_Panjang pp = new Persegi_Panjang();
        pp.lebar=30;
        pp.panjang=50;
        Segitiga s = new Segitiga();
        s.alas=20;
        s.tinggi=40;
        Persegi p = new Persegi();
        p.sisi=40;
        Lingkaran l= new Lingkaran();
    }
}

```

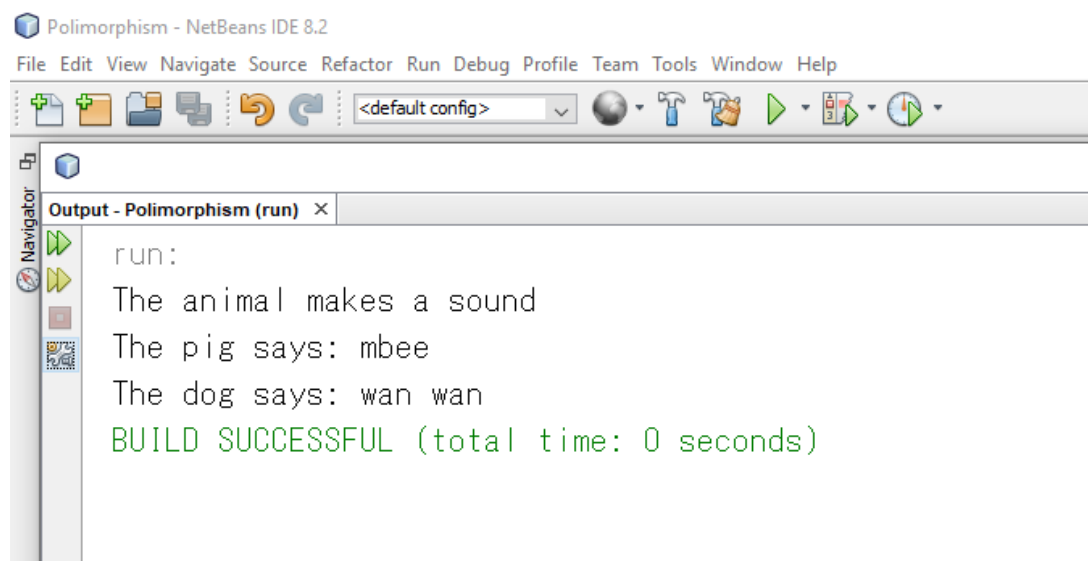
```

l.jari=20;

pp.luas();
pp.keliling();
p.luas();
p.keliling();
s.luas();
s.keliling();
l.luas();
l.keliling();
}
}

```

3.9 polymorphism



```

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: mbee");
    }
}

```

```

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: wan wan");
    }
}

```

```

class MyMainClass {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myPig = new Pig();
        Animal myDog = new Dog();

        myAnimal.animalSound();
    }
}

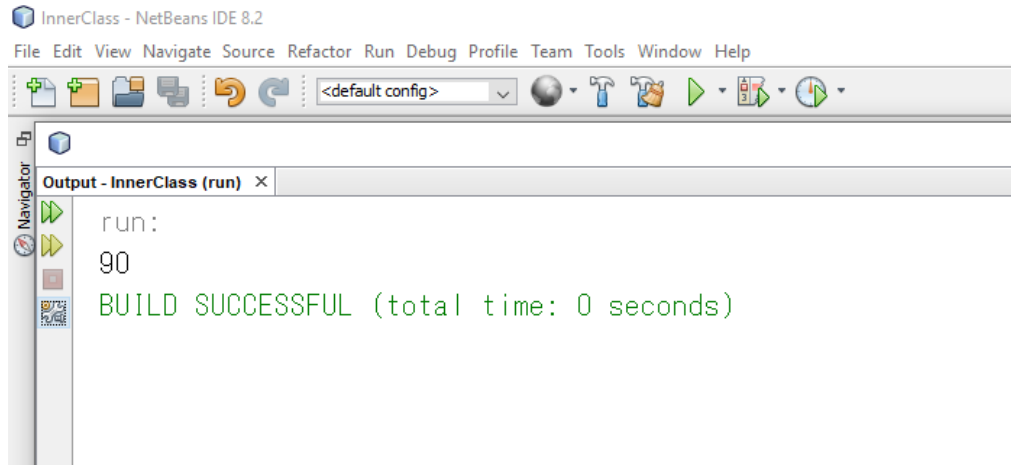
```

```

    myPig.animalSound();
    myDog.animalSound();
}
}

```

3.10 InnerClass



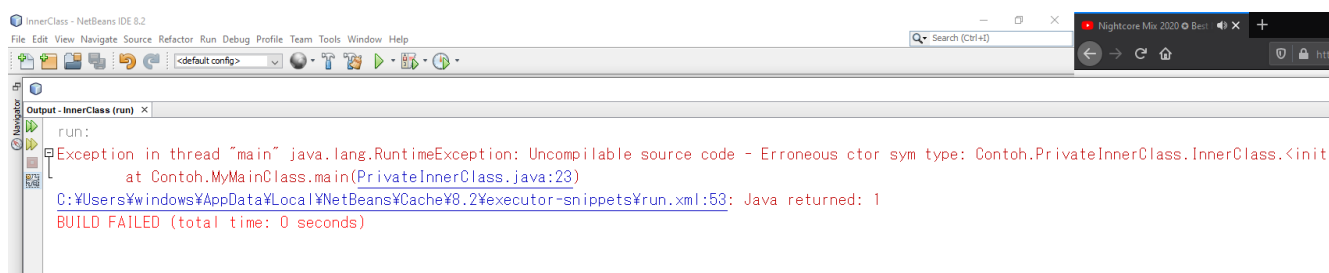
```

class OuterClass {
    int x = 10;

    class InnerClass {
        int y = 90;
    }
}

public class MyMainClass {
    public static void main(String[] args) {
        OuterClass myOuter = new OuterClass();
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
        System.out.println(myInner.y + myOuter.x);
    }
}

```



```

class OuterClass {
    int x = 10;

    private class InnerClass {
        int y = 5;
    }
}

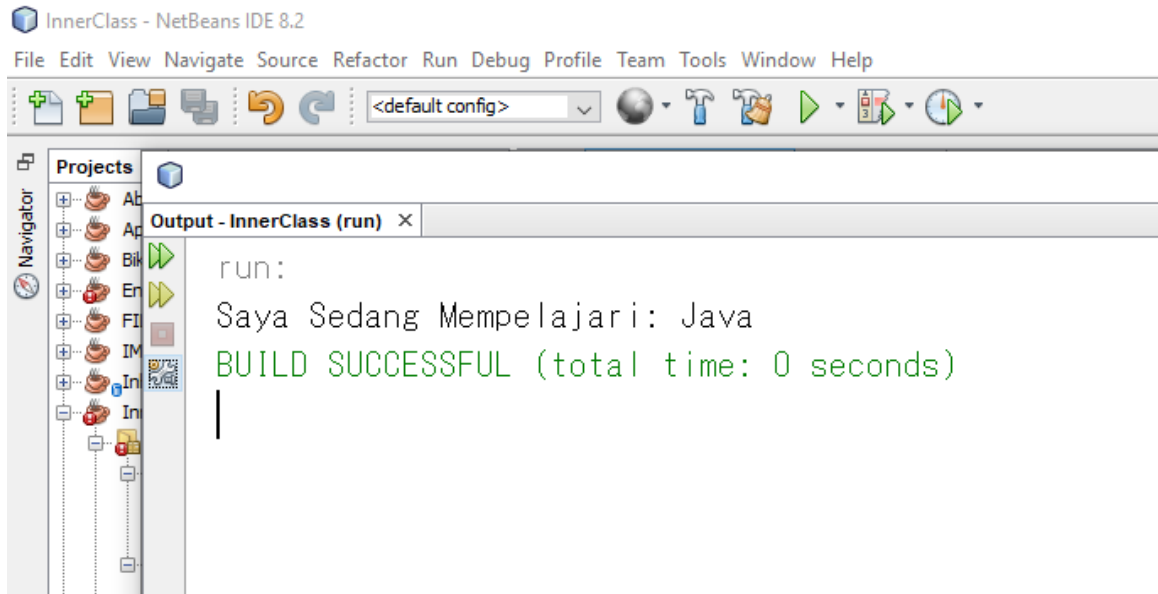
public class MyMainClass {
    public static void main(String[] args) {

```

```

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

```



c)

```
public class InnerClass {
```

```
//Class dalam/Inner Class Static
```

```
private static class Programming{
```

```
    private String language;
```

```
    private void setLanguage(String language){
```

```
        this.language = language;
```

```
    }
```

```
    private String getLanguage(){
```

```
        return language;
```

```
    }
```

```
}
```

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

```
    //Membuat Instance dari Kelas Dalam (Programming)
```

```
    InnerClass.Programming MyLanguage = new InnerClass.Programming();
```

```
    //Memasukan Nilai/Value
```

```
    MyLanguage.setLanguage("Java");
```

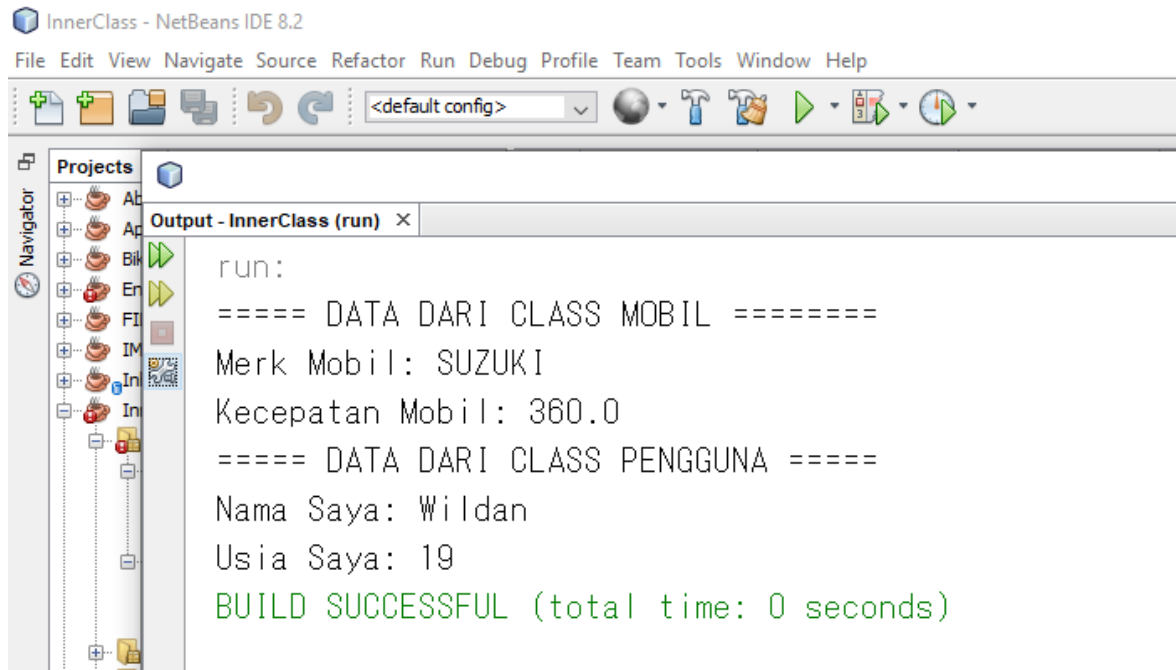
```
    //Menampilkan Hasil Output
```

```
    System.out.println("Saya Sedang Mempelajari: "+MyLanguage.getLanguage());
```

```

}
}

```



d)

```

public class KelasLuar {

    //Class dalam/Inner Class Pertama
    private class Mobil{
        private String merk = "SUZUKI";
        private float kecepatan = 360.0f;
        private void jalankan(){
            System.out.println("Merk Mobil: "+merk);
            System.out.println("Kecepatan Mobil: "+kecepatan);
        }
    }

    //Class dalam/Inner Class Kedua
    private class Pengguna{
        private String nama = "Wildan";
        private int umur = 19;
        private void identitas(){
            System.out.println("Nama Saya: "+nama);
            System.out.println("Usia Saya: "+umur);
        }
    }

    public static void main(String[] args){

```



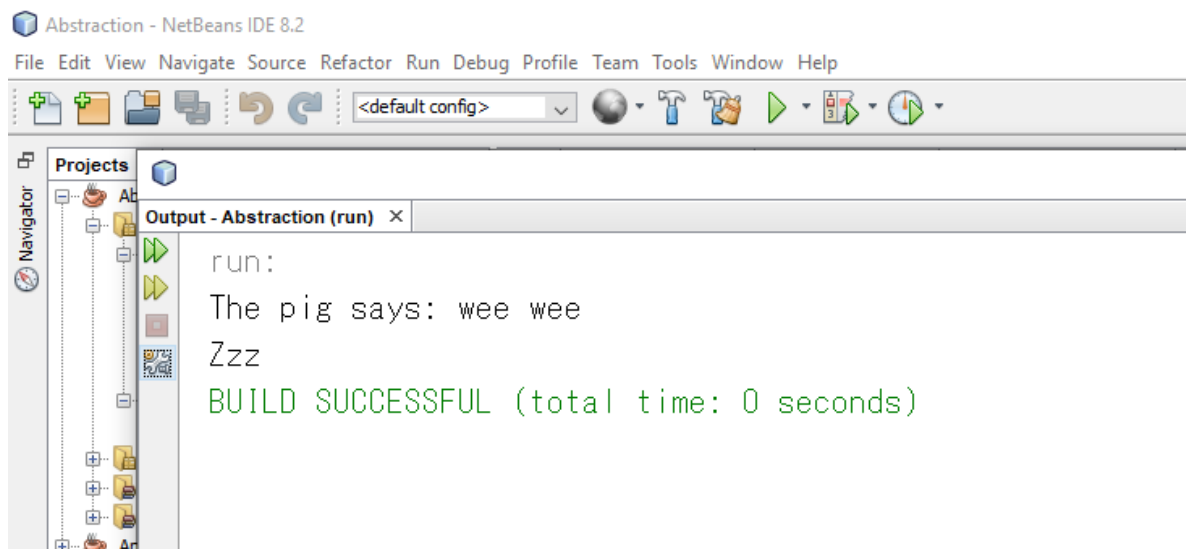
```

//Membuat Instance dari KelasLuar
KelasLuar outerclass = new KelasLuar();
//Membuat Instance dari KelasDalam (Mobil)
KelasLuar.Mobil data1 = outerclass.new Mobil();
//Membuat Instance dari KelasDalam (Pengguna)
KelasLuar.Pengguna data2 = outerclass.new Pengguna();

//Menampilkan Hasil Output
System.out.println("===== DATA DARI CLASS MOBIL =====");
data1.jalankan();
System.out.println("===== DATA DARI CLASS PENGGUNA =====");
data2.identitas();
}
}

```

3.11 Abstraction



```

// Abstract class
abstract class Animal {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep() {
        System.out.println("Zzz");
    }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
}

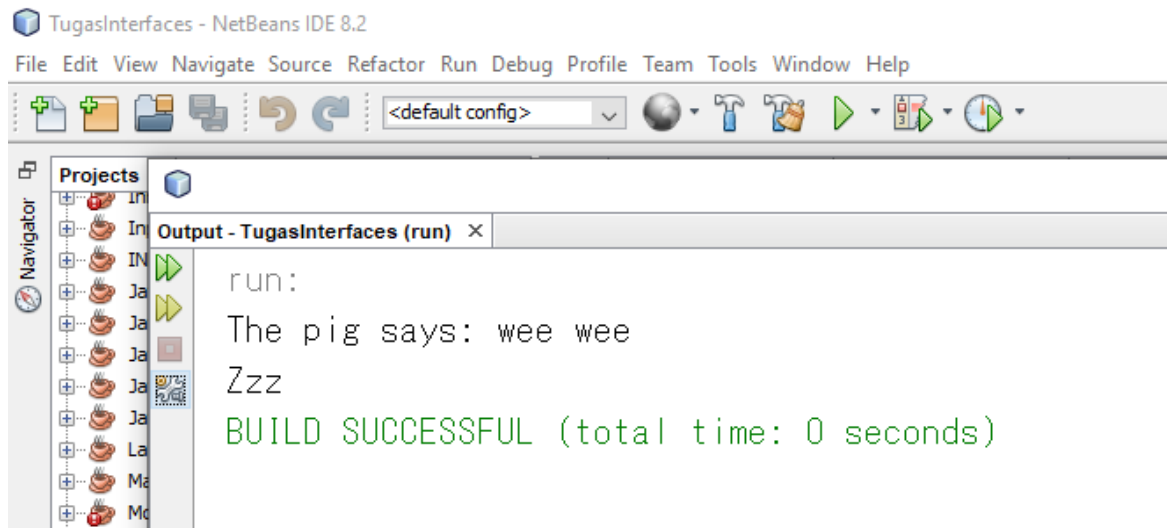
```

```

class MyMainClass {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}

```

3.12 Interface



a)

```

interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void sleep(); // interface method (does not have a body)
}

```

```

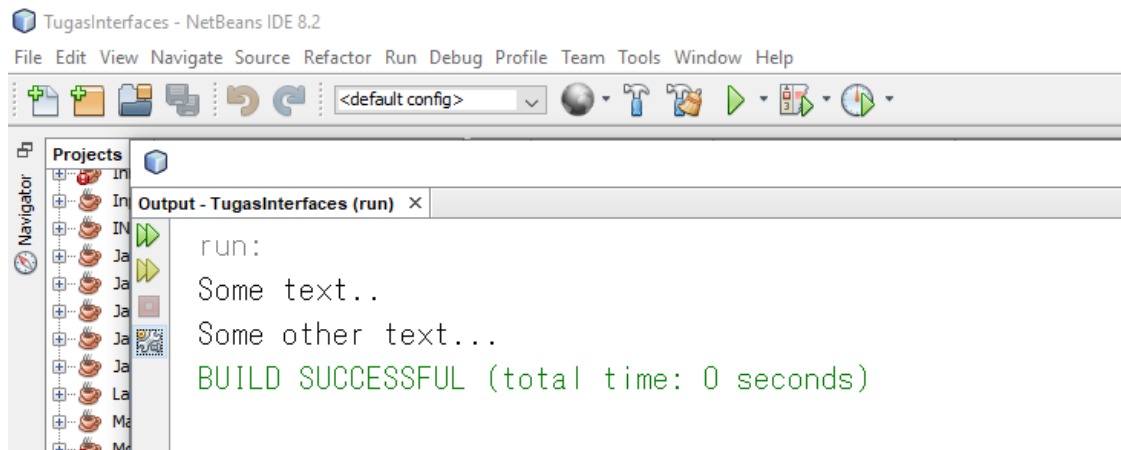
class Pig implements Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        System.out.println("Zzz");
    }
}

```

```

class MyMainClass {
    public static void main(String[] args) {
        Pig myPig = new Pig();
        myPig.animalSound();
        myPig.sleep();
    }
}

```



b)

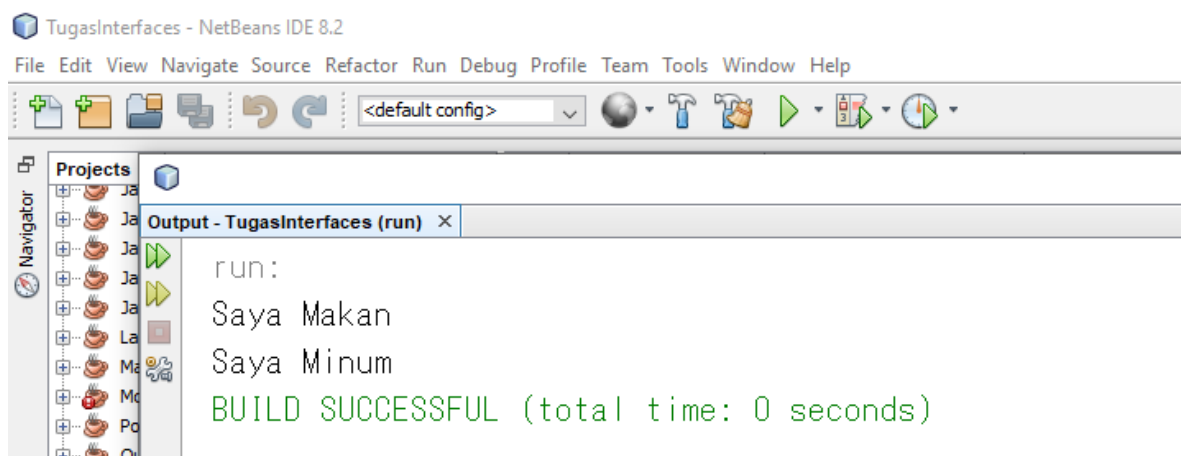
```
interface FirstInterface {
    public void myMethod(); // interface method
}
```

```
interface SecondInterface {
    public void myOtherMethod(); // interface method
}
```

// DemoClass "implements" FirstInterface and SecondInterface

```
class DemoClass implements FirstInterface, SecondInterface {
    public void myMethod() {
        System.out.println("Some text..");
    }
    public void myOtherMethod() {
        System.out.println("Some other text...");
    }
}
```

```
class MyMainClass {
    public static void main(String[] args) {
        DemoClass myObj = new DemoClass();
        myObj.myMethod();
        myObj.myOtherMethod();
    }
}
```



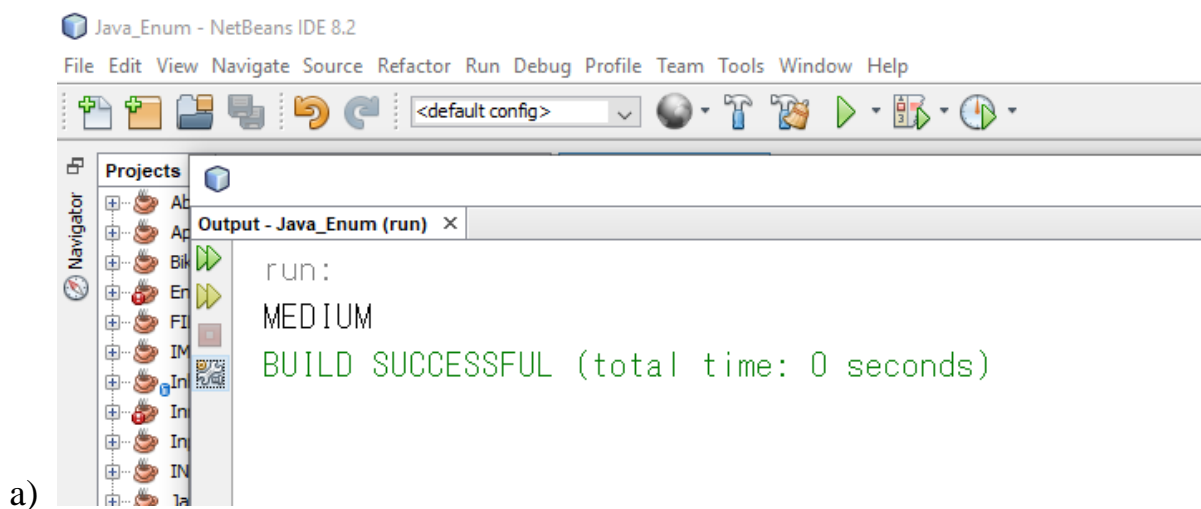
c)

```

interface makan {
    public void Ma();
}
interface minum {
    public void Mi();}
class lapar implements makan, minum {
    public void Ma() {
        System.out.println("Saya Makan");
    }
    public void Mi() {
        System.out.println("Saya Minum");
    }
}
public class MAIN {
    public static void main(String[] args) {
        lapar M = new lapar();
        M.Ma();
        M.Mi(); }
}

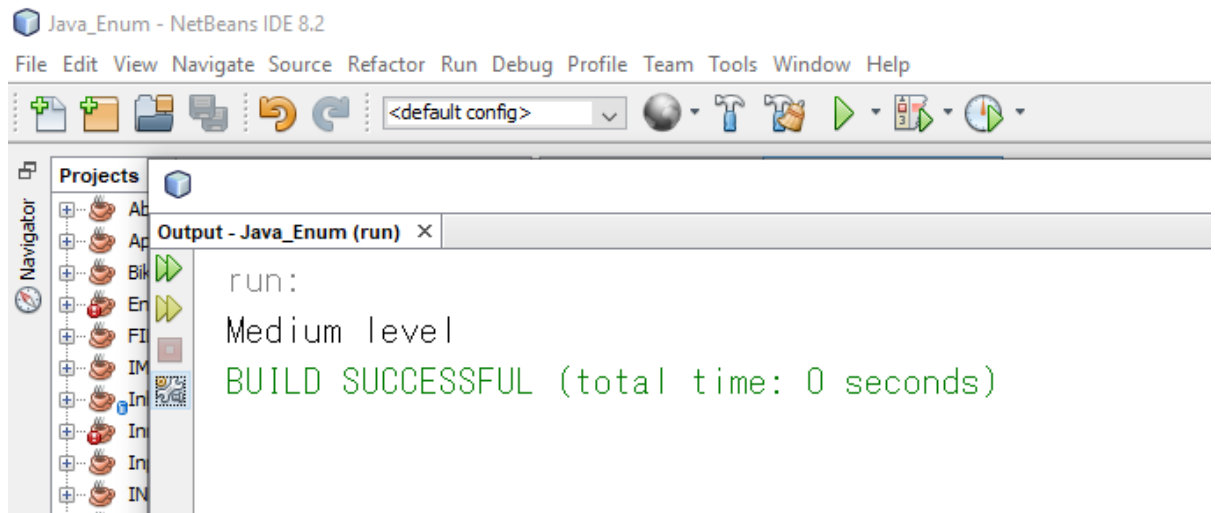
```

3.13 Enum



```
enum Level {
    LOW,
    MEDIUM,
    HIGH
}

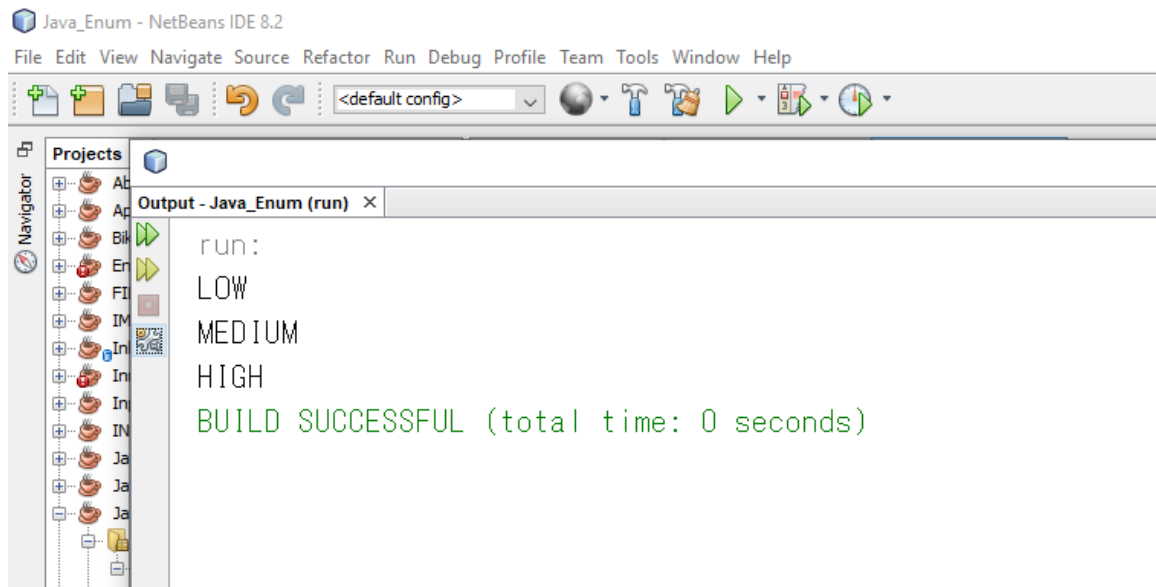
public class MyClass {
    public static void main(String[] args) {
        Level myVar = Level.MEDIUM;
        System.out.println(myVar);
    }
}
```



```
enum Level {
    LOW,
    MEDIUM,
    HIGH
}

public class MyClass {
    public static void main(String[] args) {
        Level myVar = Level.MEDIUM;

        switch(myVar) {
            case LOW:
                System.out.println("Low level");
                break;
            case MEDIUM:
                System.out.println("Medium level");
                break;
            case HIGH:
                System.out.println("High level");
                break;
        }
    }
}
```

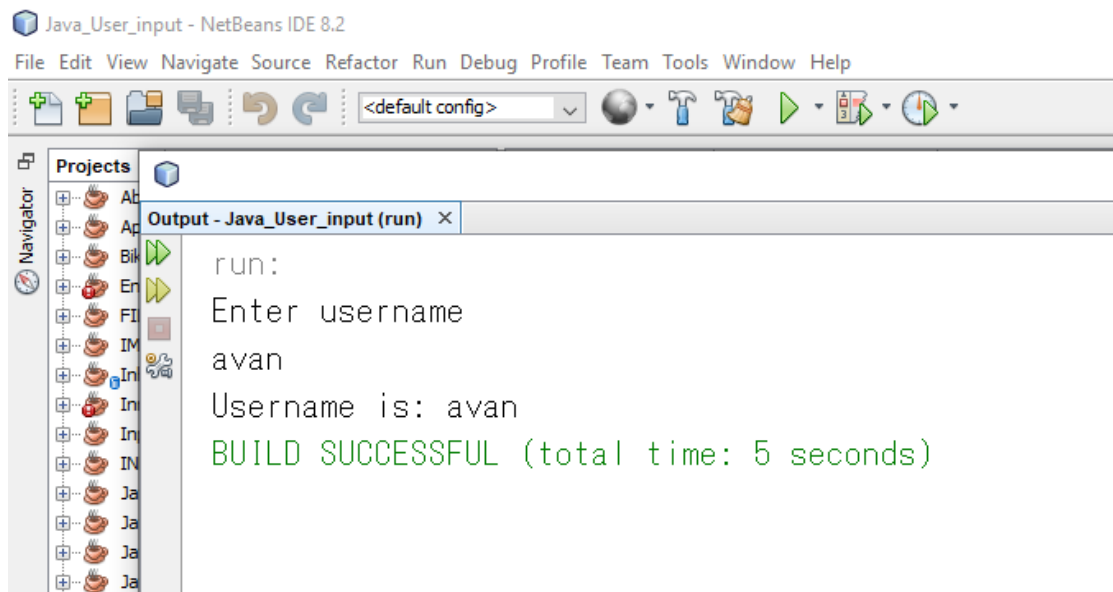


c)

```
enum Level {
    LOW,
    MEDIUM,
    HIGH
}

public class MyClass {
    public static void main(String[] args) {
        for (Level myVar : Level.values()) {
            System.out.println(myVar);
        }
    }
}
```

3.14 User input



a)

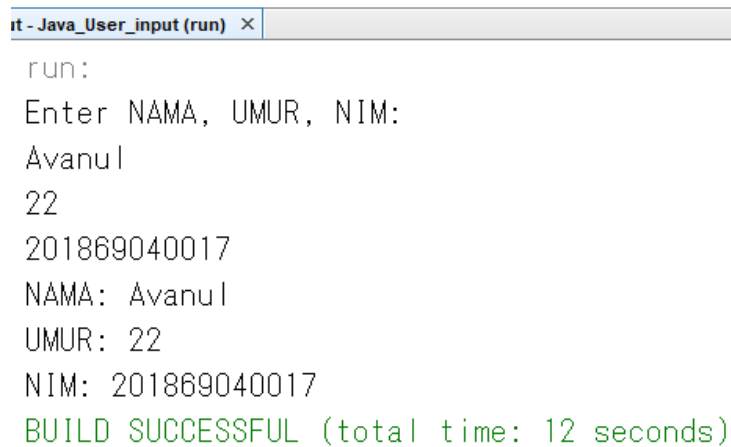
```

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

public class Java_User_input {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in); // Create a Scanner object
        System.out.println("Enter username");

        String userName = myObj.nextLine(); // Read user input
        System.out.println("Username is: " + userName); // Output user input
    }
}

```



```

run:
Enter NAMA, UMUR, NIM:
Avanul
22
201869040017
NAMA: Avanul
UMUR: 22
NIM: 201869040017
BUILD SUCCESSFUL (total time: 12 seconds)

```

b)

```

import java.util.Scanner;

public class Input_Type {

    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter NAMA, UMUR, NIM:");

        // String input
        String name = myObj.nextLine();

        // Numerical input
        int umur = myObj.nextInt();
        long nim = myObj.nextLong();

        // Output input by user
        System.out.println("NAMA: " + name);
    }
}

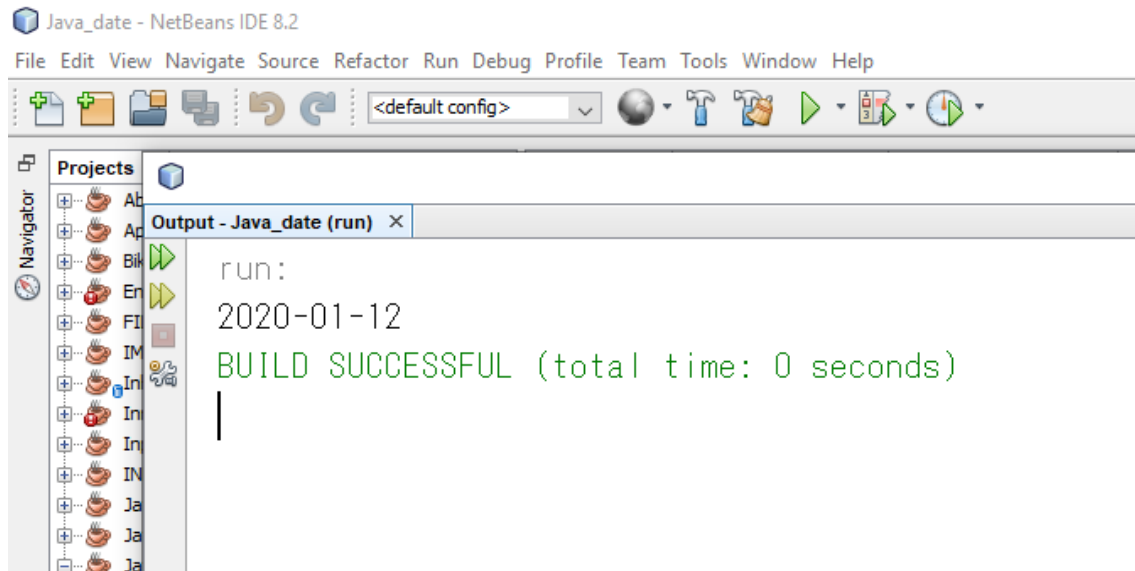
```

```

System.out.println("UMUR: " + umur);
System.out.println("NIM: " + nim);
}
}

```

3.15 Date



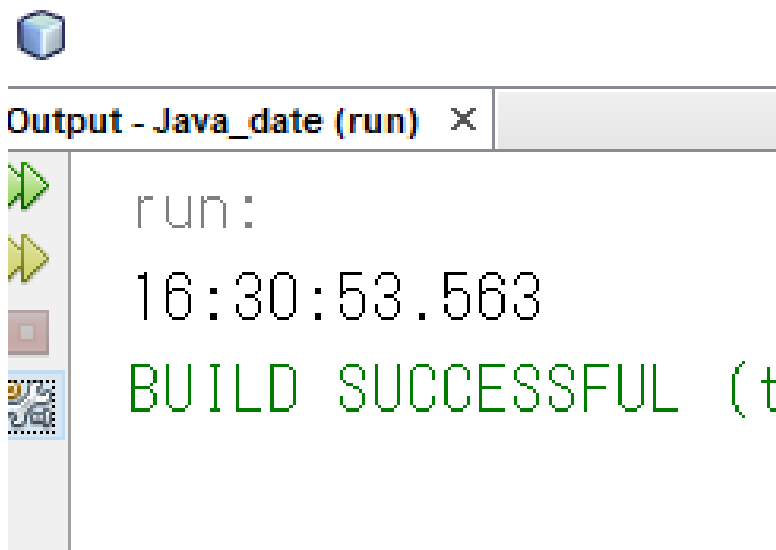
a)

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

```

public class MyClass {
    public static void main(String[] args) {
        LocalDate myObj = LocalDate.now(); // Create a date object
        System.out.println(myObj); // Display the current date
    }
}

```



b)

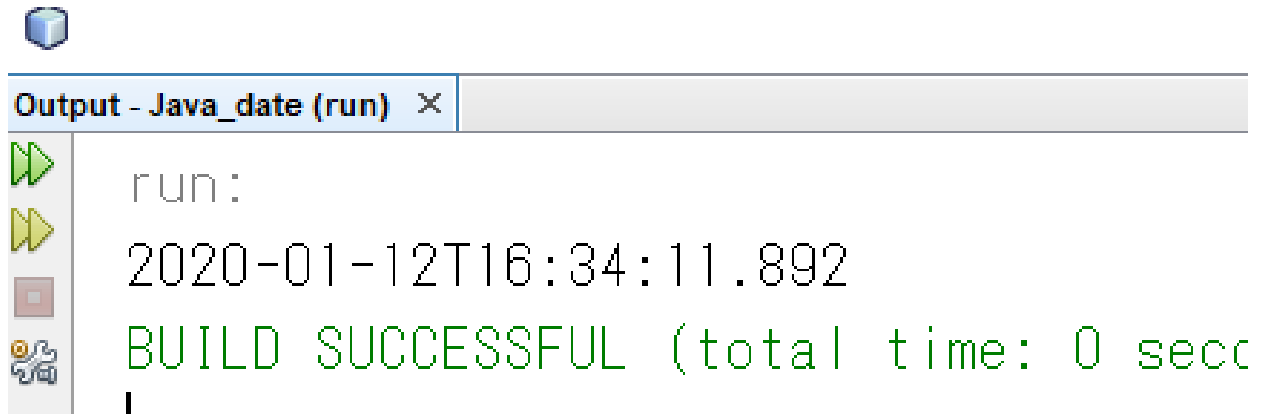
```
import java.time.LocalTime; // import the LocalTime class
```



```

public class MyClass {
    public static void main(String[] args) {
        LocalDateTime myObj = LocalDateTime.now();
        System.out.println(myObj);
    }
}

```



c) |

```

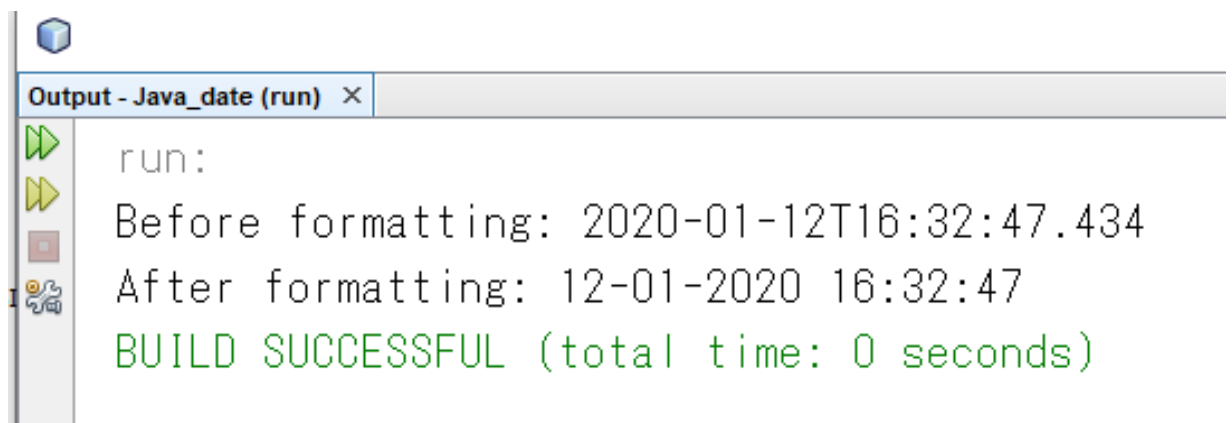
import java.time.LocalDateTime; // import the LocalDateTime class

```

```

public class MyClass {
    public static void main(String[] args) {
        LocalDateTime myObj = LocalDateTime.now();
        System.out.println(myObj);
    }
}

```



d)

```

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

```

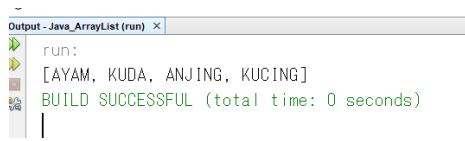
```

public class MyClass {
    public static void main(String[] args) {
        LocalDateTime myDateObj = LocalDateTime.now();
        System.out.println("Before formatting: " + myDateObj);
        DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

        String formattedDate = myDateObj.format(myFormatObj);
        System.out.println("After formatting: " + formattedDate);
    }
}

```

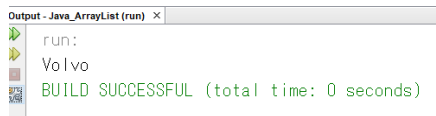
3.16 ArrayList



a) |

```
import java.util.ArrayList;
```

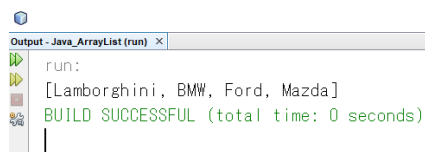
```
public class Java_ArrayList {
    public static void main(String[] args) {
        ArrayList<String> HEWAN = new ArrayList<String>();
        HEWAN.add("AYAM");
        HEWAN.add("KUDA");
        HEWAN.add("ANJING");
        HEWAN.add("KUCING");
        System.out.println(HEWAN);
    }
}
```



b) |

```
import java.util.ArrayList;
```

```
public class GET {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars.get(0));
    }
}
```



c) |

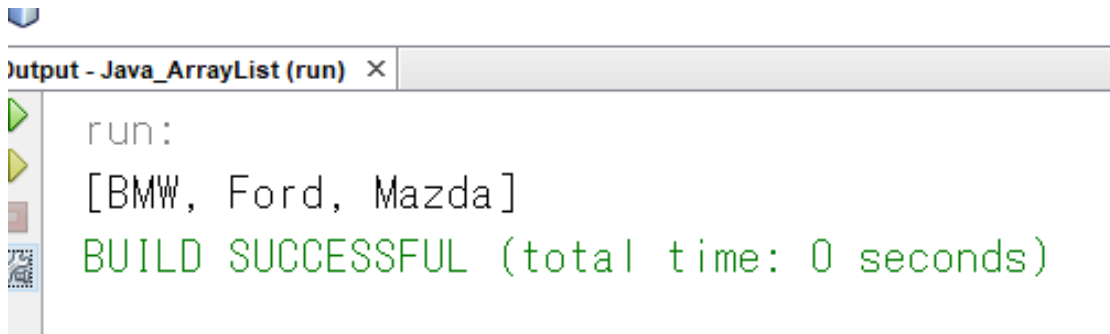
```
import java.util.ArrayList;
```

```
public class MyClass {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
    }
}
```

```

cars.set(0, "Lamborghini");
System.out.println(cars);
}
}

```



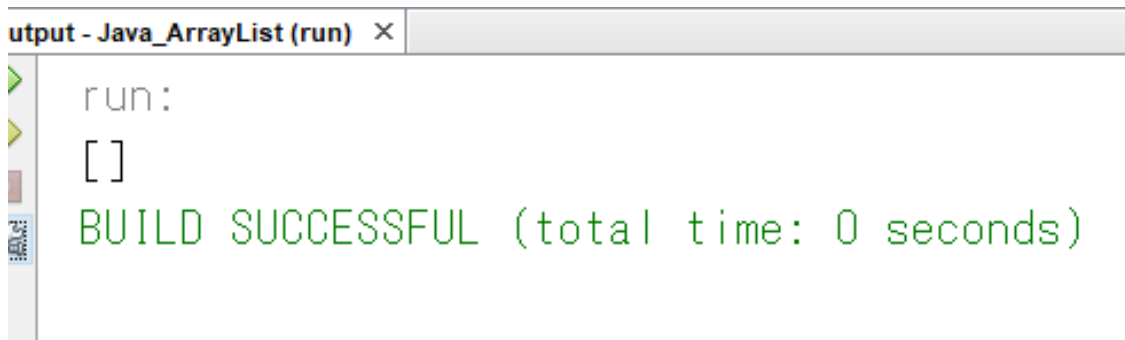
d)

```

import java.util.ArrayList;

public class Hapus {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        cars.remove(0);
        System.out.println(cars);
    }
}

```



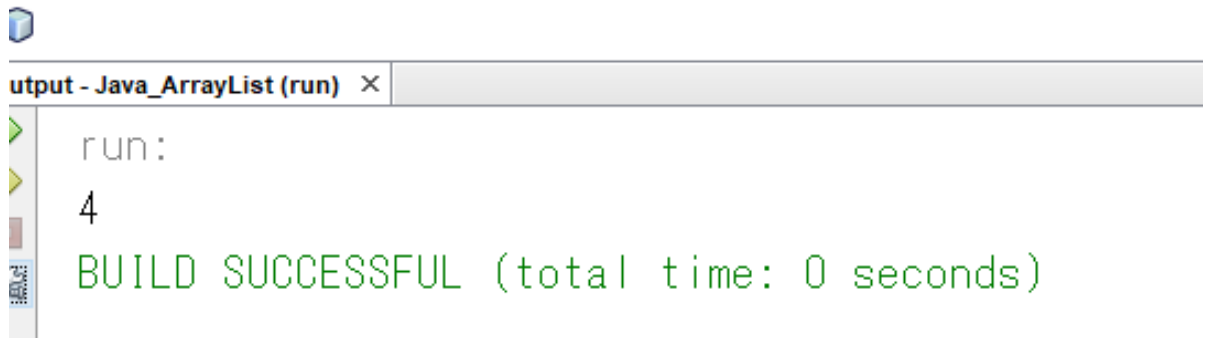
e)

```

import java.util.ArrayList;

public class Bersih {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        cars.clear();
        System.out.println(cars);
    }
}

```

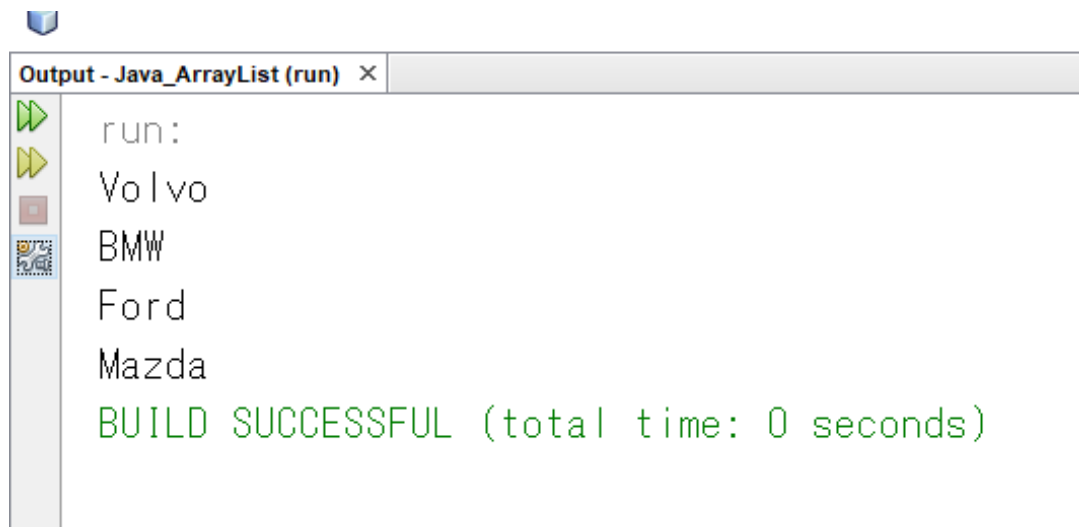


```
Output - Java_ArrayList (run) ×
run:
4
BUILD SUCCESSFUL (total time: 0 seconds)
```

f)

```
import java.util.ArrayList;

public class Ukuran {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars.size());
    }
}
```

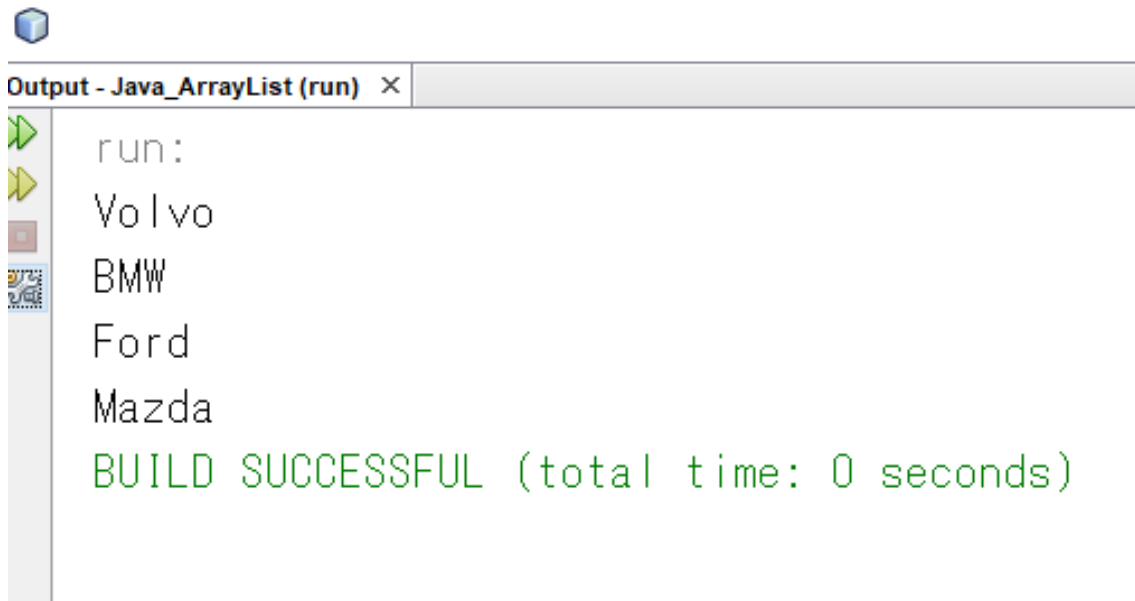


```
Output - Java_ArrayList (run) ×
run:
Volvo
BMW
Ford
Mazda
BUILD SUCCESSFUL (total time: 0 seconds)
```

g)

```
import java.util.ArrayList;

public class Pengulangan {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        for (int i = 0; i < cars.size(); i++) {
            System.out.println(cars.get(i));
        }
    }
}
```

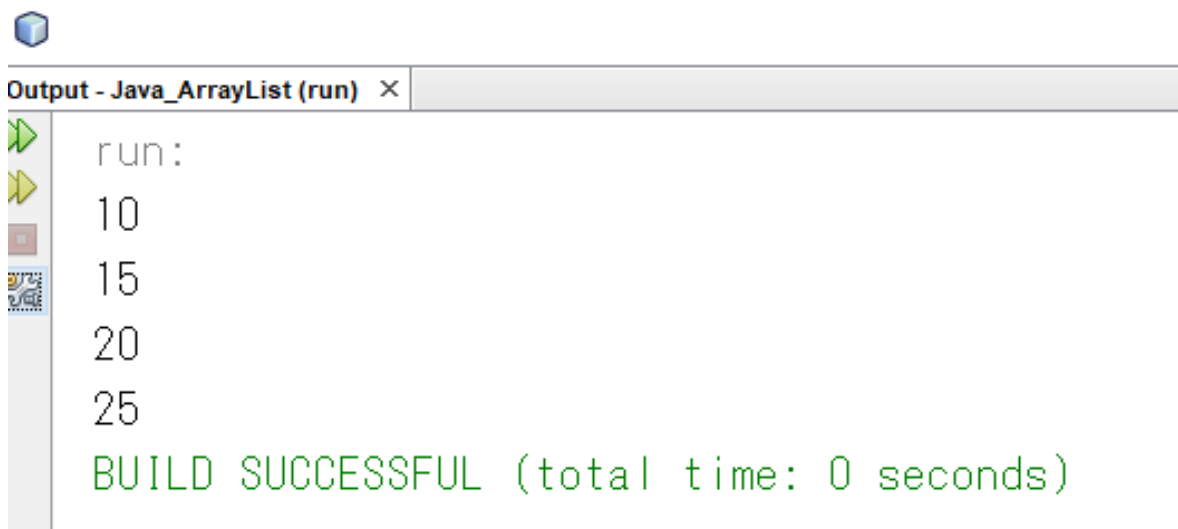


```
Output - Java_ArrayList (run) X
run:
Volvo
BMW
Ford
Mazda
BUILD SUCCESSFUL (total time: 0 seconds)
```

h)

```
import java.util.ArrayList;

public class Pengulangan2 {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        for (String i : cars) {
            System.out.println(i);
        }
    }
}
```



```
Output - Java_ArrayList (run) X
run:
10
15
20
25
BUILD SUCCESSFUL (total time: 0 seconds)
```

i)

```
import java.util.ArrayList;

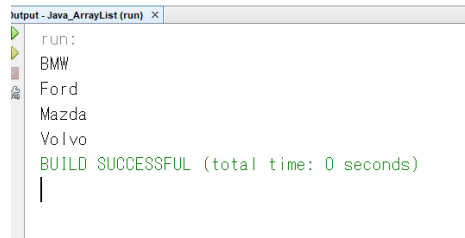
public class TipeLain {
    public static void main(String[] args) {
        ArrayList<Integer> myNumbers = new ArrayList<Integer>();
        myNumbers.add(10);
```

```

myNumbers.add(15);
myNumbers.add(20);
myNumbers.add(25);
for (int i : myNumbers) {
    System.out.println(i);
}
}
}

```

j)



```

run:
BMW
Ford
Mazda
Volvo
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

import java.util.ArrayList;
import java.util.Collections;

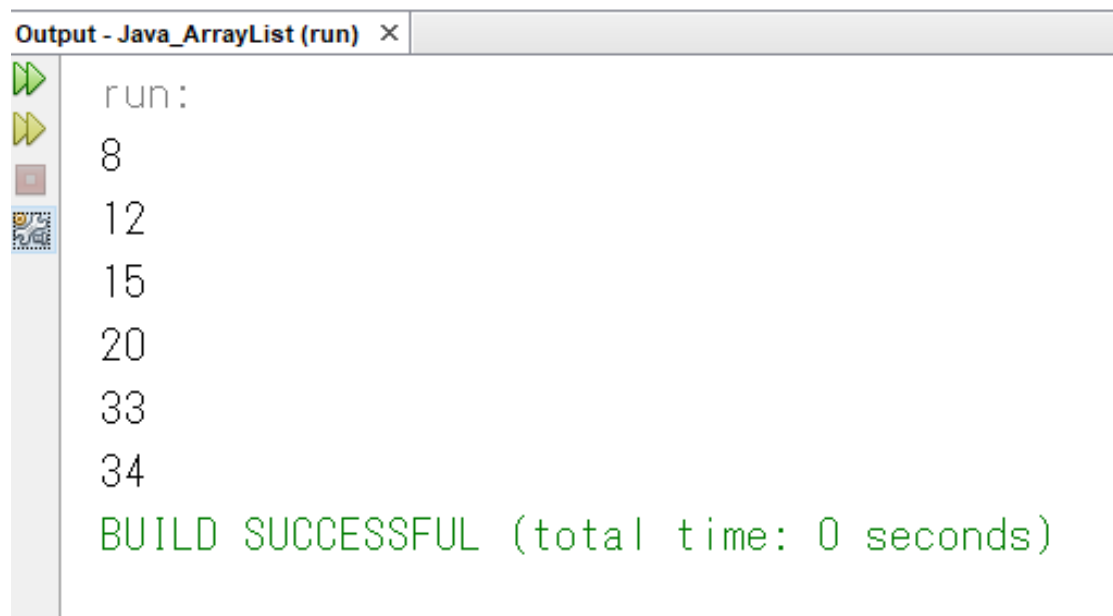
public class Mengurutkan {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");

        Collections.sort(cars);

        for (String i : cars) {
            System.out.println(i);
        }
    }
}

```

k)



```

run:
8
12
15
20
33
34
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

import java.util.ArrayList;
import java.util.Collections;

public class Mnengurutkan2 {
    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);

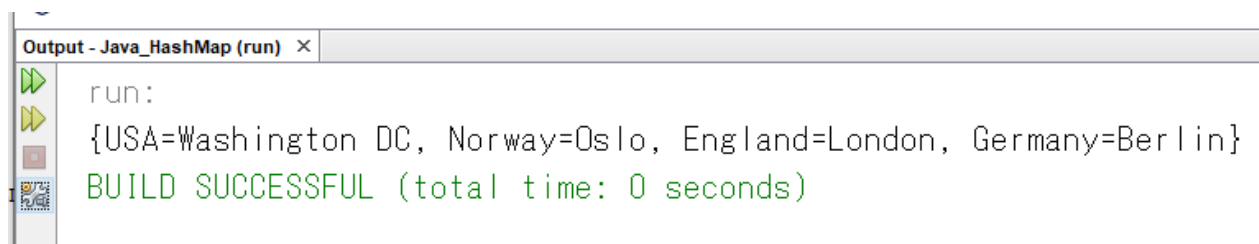
        Collections.sort(myNumbers);

        for (int i : myNumbers) {
            System.out.println(i);
        }
    }
}

```

3.17 HashMap

a)



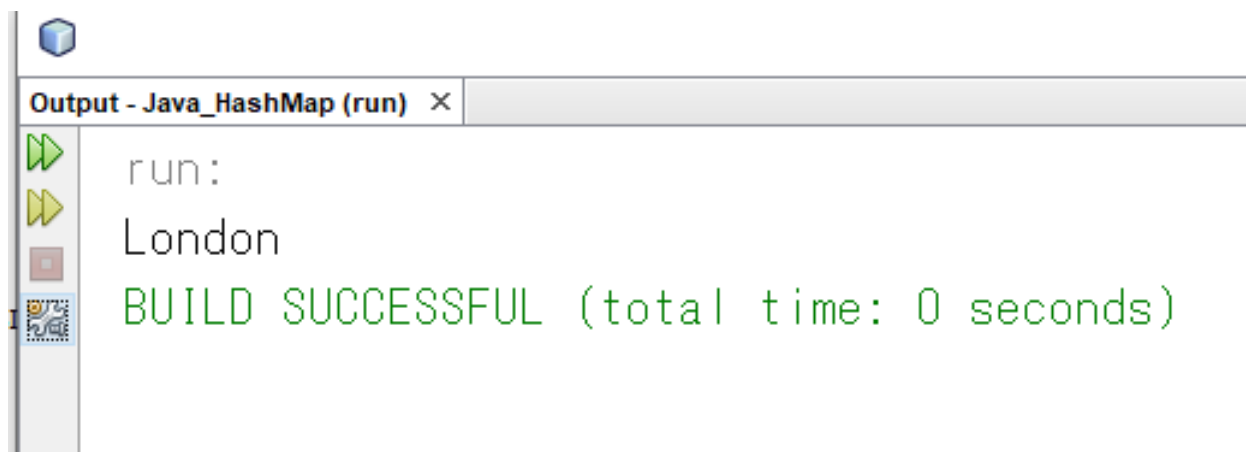
```
import java.util.HashMap;
```

```

public class Java_HashMap {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");
        System.out.println(capitalCities);
    }
}

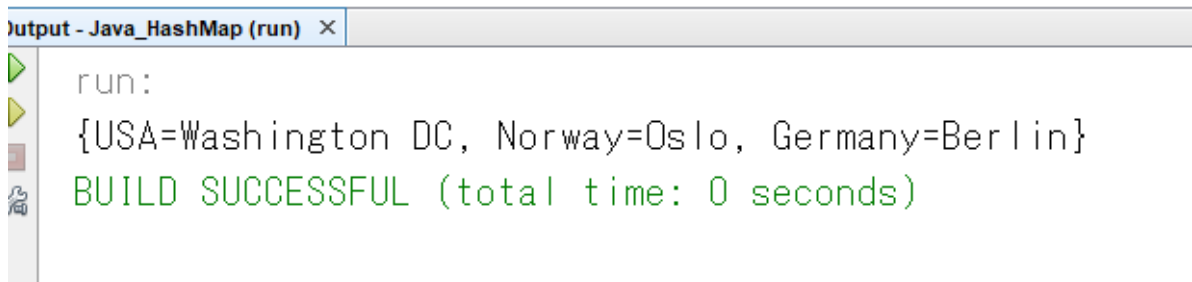
```

b)



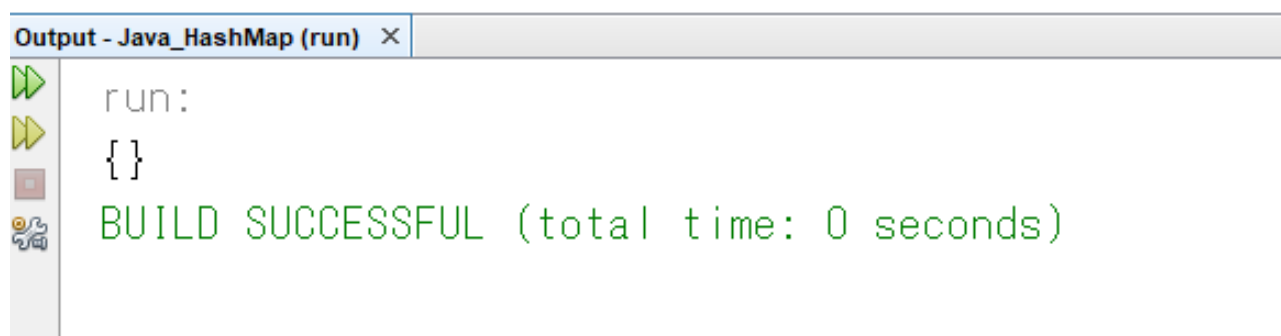
```
import java.util.HashMap;

public class MyClass {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");
        System.out.println(capitalCities.get("England"));
    }
}
```



c) import java.util.HashMap;

```
public class Hapus {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");
        capitalCities.remove("England");
        System.out.println(capitalCities);
    }
}
```

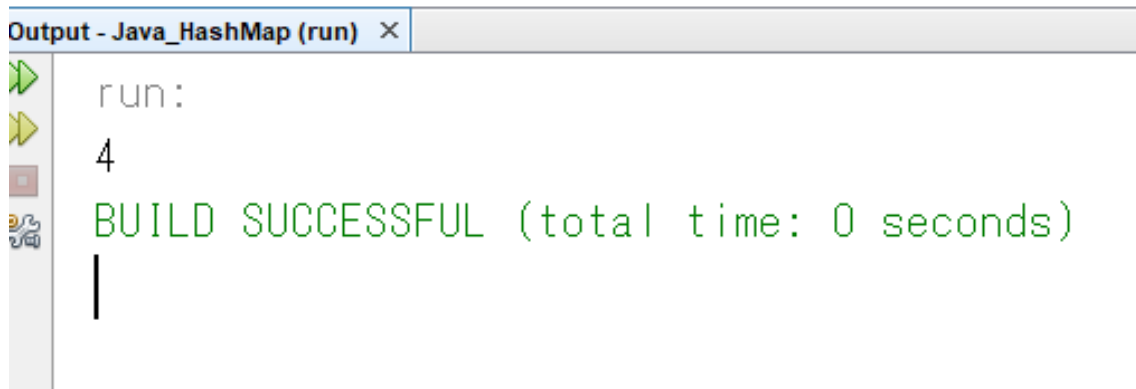


d) import java.util.HashMap;

```
public class Bersih {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");
        capitalCities.clear();
        System.out.println(capitalCities);
    }
}
```



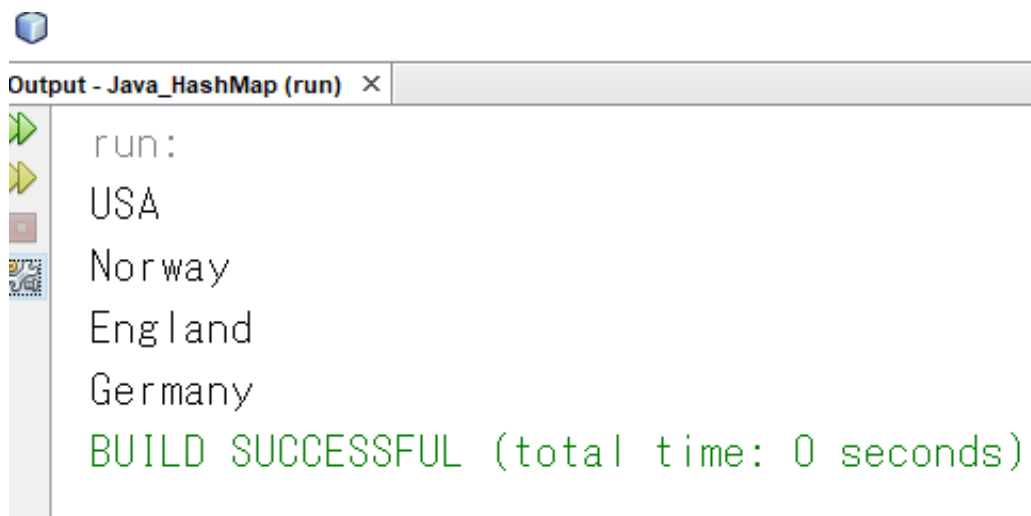
```
}  
}
```



```
Output - Java_HashMap (run) ×  
run:  
4  
BUILD SUCCESSFUL (total time: 0 seconds)
```

e)

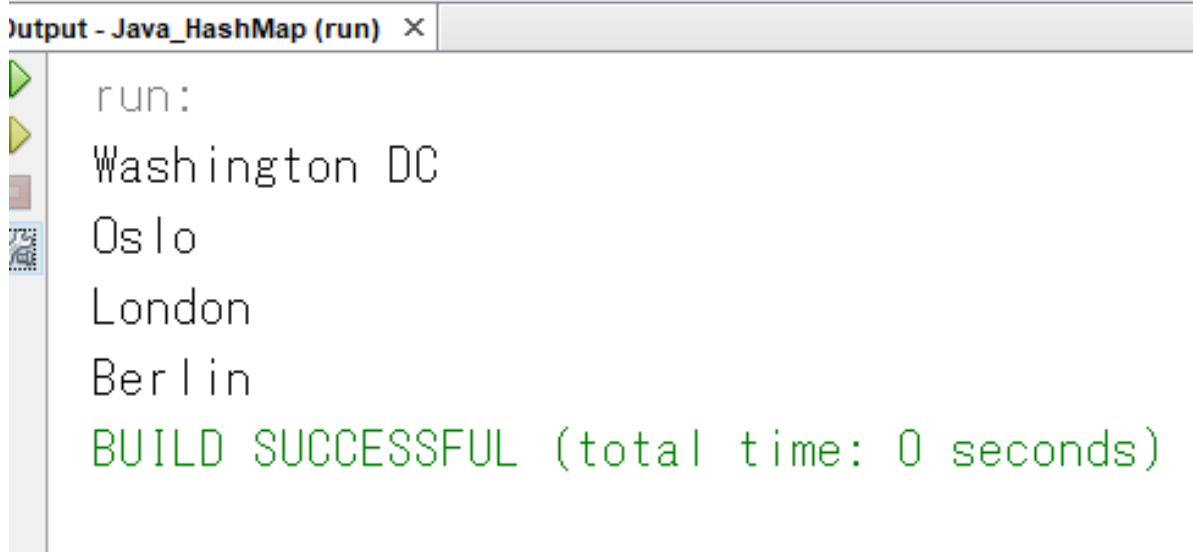
```
import java.util.HashMap;  
  
public class Ukuran {  
    public static void main(String[] args) {  
        HashMap<String, String> capitalCities = new HashMap<String, String>();  
        capitalCities.put("England", "London");  
        capitalCities.put("Germany", "Berlin");  
        capitalCities.put("Norway", "Oslo");  
        capitalCities.put("USA", "Washington DC");  
        System.out.println(capitalCities.size());  
    }  
}
```



```
Output - Java_HashMap (run) ×  
run:  
USA  
Norway  
England  
Germany  
BUILD SUCCESSFUL (total time: 0 seconds)
```

f)

```
import java.util.HashMap;  
  
public class MyClass {  
    public static void main(String[] args) {  
        HashMap<String, String> capitalCities = new HashMap<String, String>();  
        capitalCities.put("England", "London");  
        capitalCities.put("Germany", "Berlin");  
        capitalCities.put("Norway", "Oslo");  
        capitalCities.put("USA", "Washington DC");  
  
        for (String i : capitalCities.keySet()) {  
            System.out.println(i);  
        }  
    }  
}
```



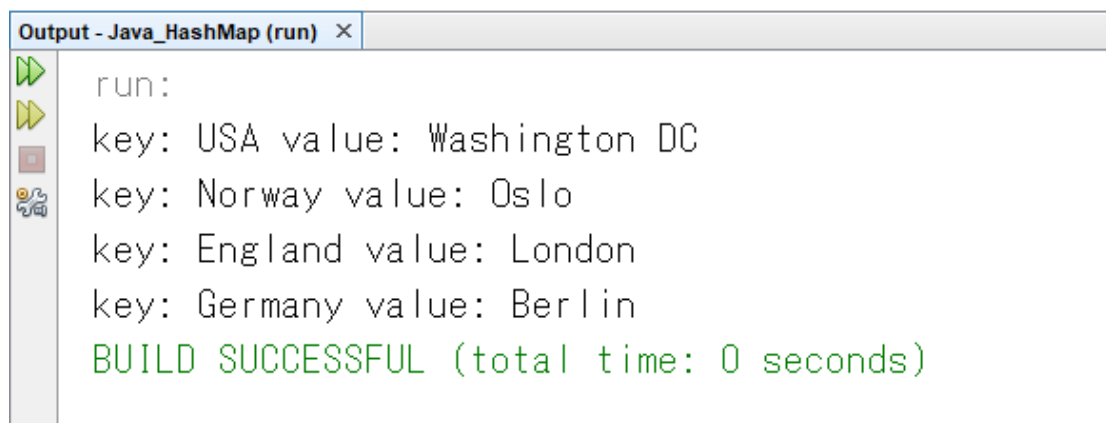
```
Output - Java_HashMap (run) x
run:
Washington DC
Oslo
London
Berlin
BUILD SUCCESSFUL (total time: 0 seconds)
```

g)

```
import java.util.HashMap;

public class MyClass {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");

        for (String i : capitalCities.values()) {
            System.out.println(i);
        }
    }
}
```



```
Output - Java_HashMap (run) x
run:
key: USA value: Washington DC
key: Norway value: Oslo
key: England value: London
key: Germany value: Berlin
BUILD SUCCESSFUL (total time: 0 seconds)
```

h)

```
import java.util.HashMap;

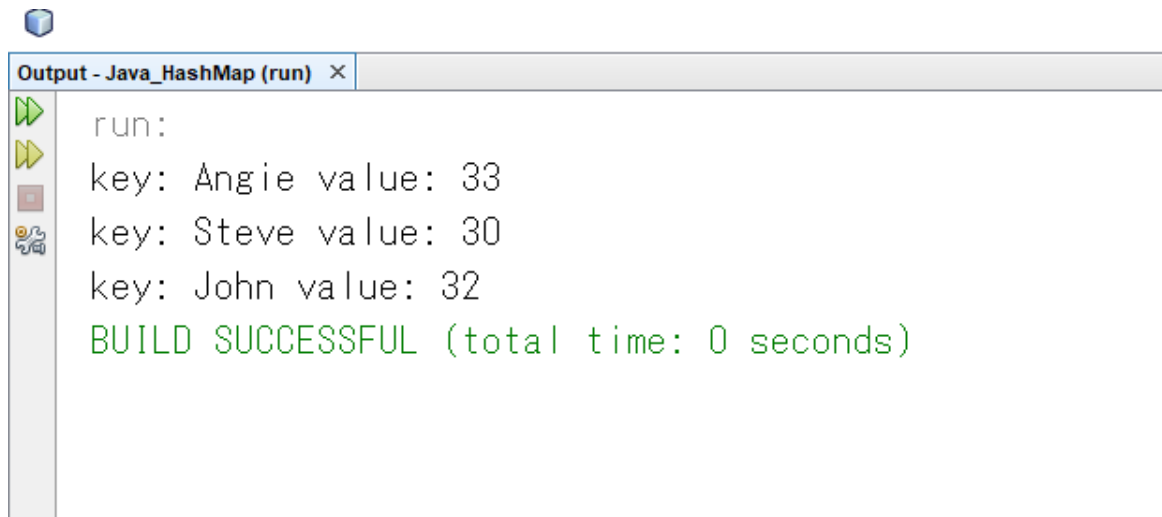
public class MyClass {
    public static void main(String[] args) {
        HashMap<String, String> capitalCities = new HashMap<String, String>();
        capitalCities.put("England", "London");
        capitalCities.put("Germany", "Berlin");
        capitalCities.put("Norway", "Oslo");
        capitalCities.put("USA", "Washington DC");

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

```

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

```



```

run:
key: Angie value: 33
key: Steve value: 30
key: John value: 32
BUILD SUCCESSFUL (total time: 0 seconds)

```

i)

```

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

public class TipeLain {
    public static void main(String[] args) {

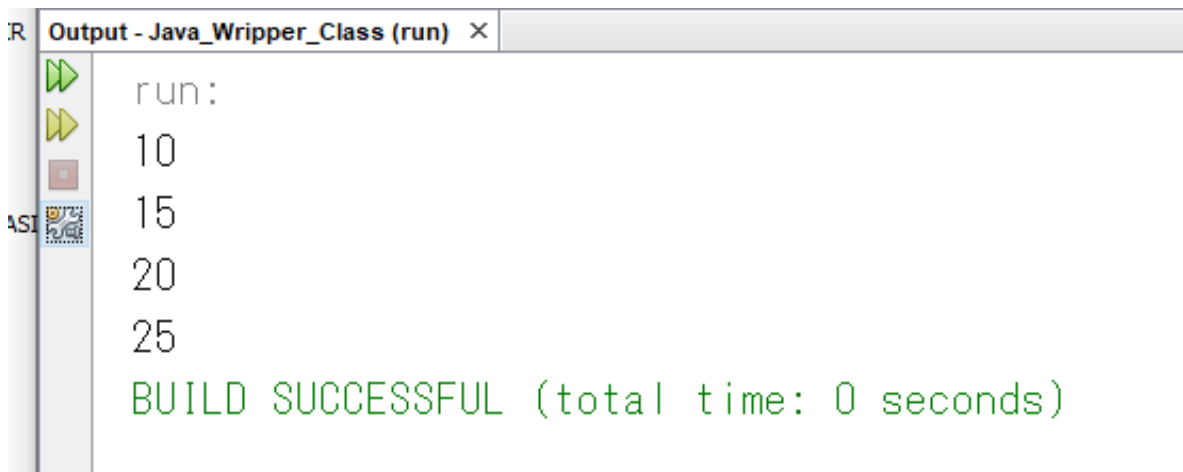
        // Create a HashMap object called people
        HashMap<String, Integer> people = new HashMap<String, Integer>();

        // 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));
        }
    }
}

```

3.18 Wrapper Class



```

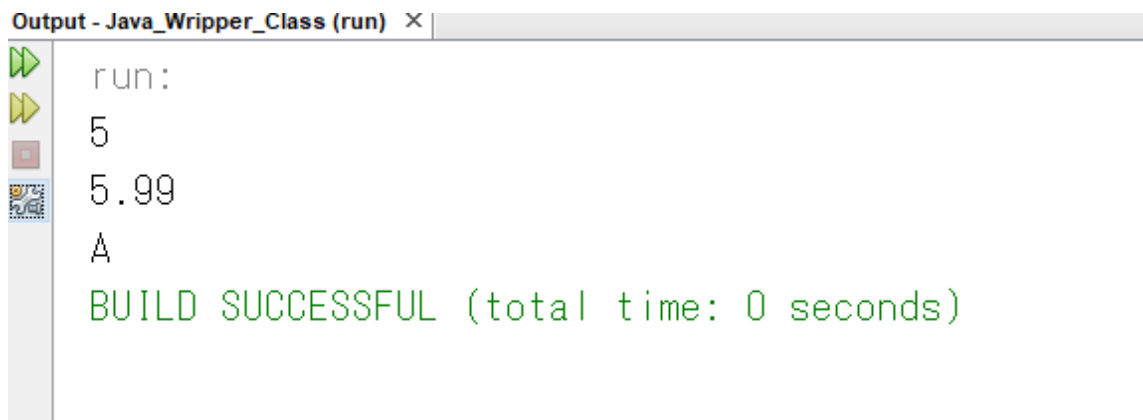
run:
10
15
20
25
BUILD SUCCESSFUL (total time: 0 seconds)

```

a)

```
import java.util.ArrayList;

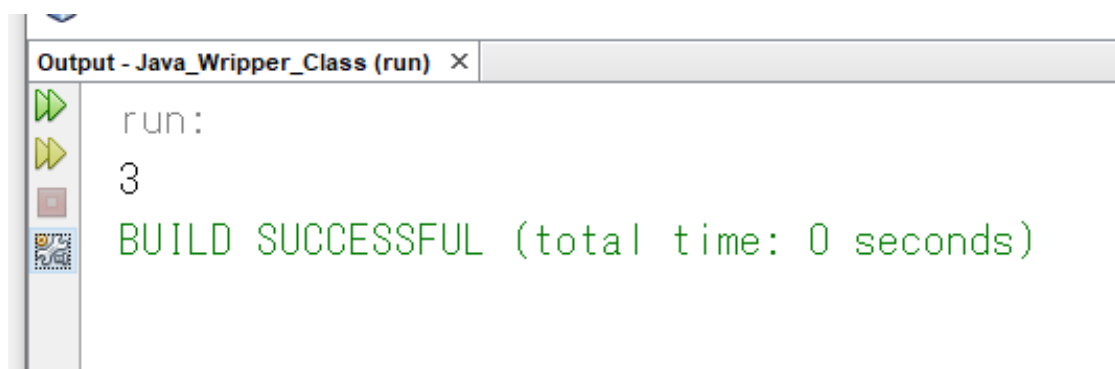
public class Java_Wripper_Class {
    public static void main(String[] args) {
        ArrayList<Integer> myNumbers = new ArrayList<Integer>();
        myNumbers.add(10);
        myNumbers.add(15);
        myNumbers.add(20);
        myNumbers.add(25);
        for (int i : myNumbers) {
            System.out.println(i);
        }
    }
}
```



```
Output - Java_Wripper_Class (run) x
run:
5
5.99
A
BUILD SUCCESSFUL (total time: 0 seconds)
```

b)

```
public class WrapperOnject {
    public static void main(String[] args) {
        Integer myInt = 5;
        Double myDouble = 5.99;
        Character myChar = 'A';
        System.out.println(myInt);
        System.out.println(myDouble);
        System.out.println(myChar);
    }
}
```

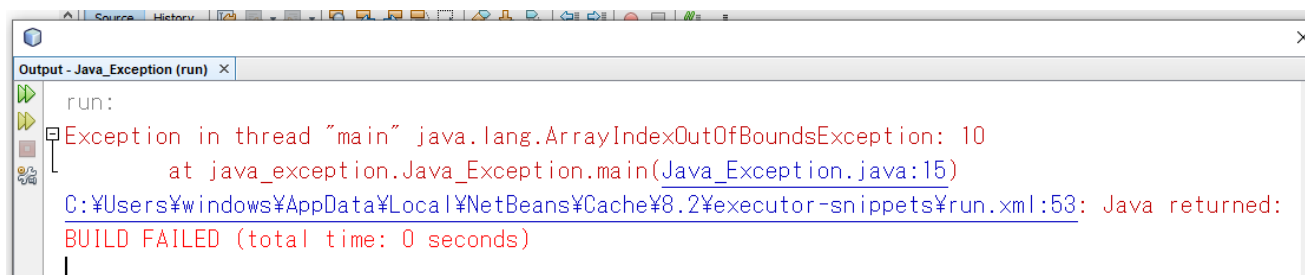


```
Output - Java_Wripper_Class (run) x
run:
3
BUILD SUCCESSFUL (total time: 0 seconds)
```

c)

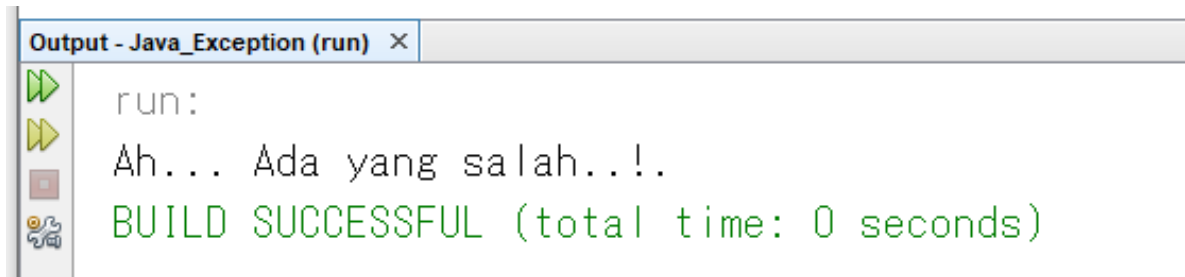
```
public class MyClass {
    public static void main(String[] args) {
        Integer myInt = 100;
        String myString = myInt.toString();
        System.out.println(myString.length());
    }
}
```

3.19 Exception



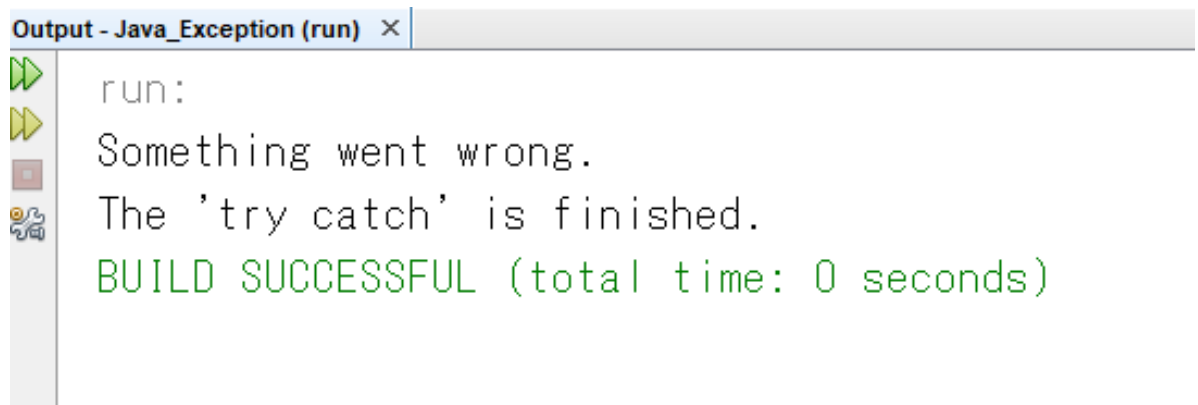
a)

```
public class MyClass {
    public static void main(String[] args) {
        int[] myNumbers = {1, 2, 3};
        System.out.println(myNumbers[10]);
    }
}
```



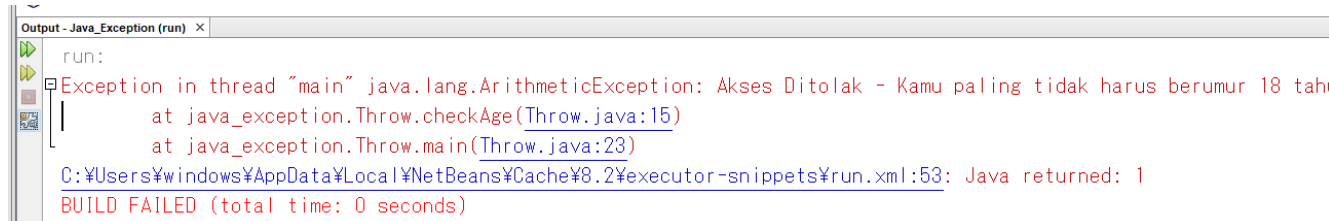
b)

```
public class Try_Catch {
    public static void main(String[] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Ah... Ada yang salah...!.");
        }
    }
}
```



c)

```
public class Finally {
    public static void main(String[] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        } finally {
            System.out.println("The 'try catch' is finished.");
        }
    }
}
```

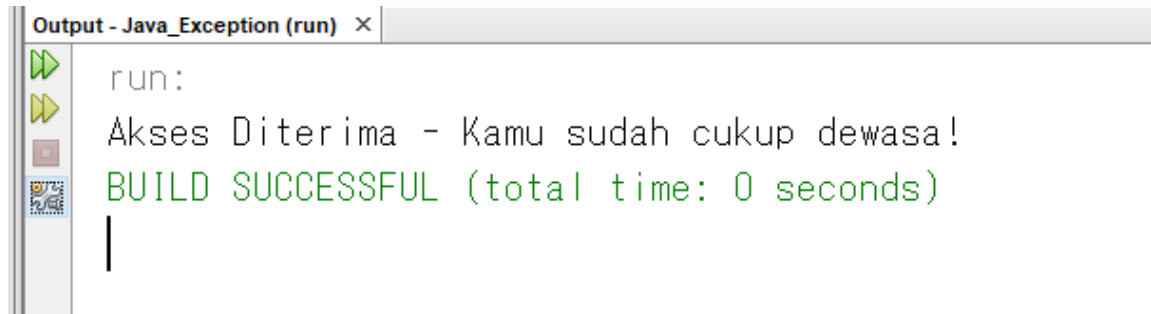


```
Output - Java_Exception (run) x
run:
Exception in thread "main" java.lang.ArithmeticException: Akses Ditolak - Kamu paling tidak harus berumur 18 tahun
    at java_exception.Throw.checkAge(Throw.java:15)
    at java_exception.Throw.main(Throw.java:23)
C:\Users\Windows\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

d)

```
public class Throw {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Akses Ditolak - Kamu paling tidak harus berumur 18 tahun.");
        }
        else {
            System.out.println("Akses Diterima - Kamu sudah cukup dewasa!");
        }
    }

    public static void main(String[] args) {
        checkAge(15); // Set age to 15 (which is below 18...)
    }
}
```



```
Output - Java_Exception (run) x
run:
Akses Diterima - Kamu sudah cukup dewasa!
BUILD SUCCESSFUL (total time: 0 seconds)
```

e)

```
public class Throw2 {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Akses Ditolak - Kamu paling tidak harus berumur 18 tahun.");
        }
        else {
            System.out.println("Akses Diterima - Kamu sudah cukup dewasa!");
        }
    }

    public static void main(String[] args) {
        checkAge(20);
    }
}
```

BAB IV

ANALISA HASIL PERCOBAAN

4.1 Object

a) Objects, Ketika kita membuat kelas kita akan mengisinya dengan objek, dan objek tersebut terdiri dari nama kelas, nama objek dan kata kunci new. Kita akan mengisi kelas dengan variabel x berupa Integer dengan hasil 5, dan membuat objek baru dengan kata kunci new diikuti nama kelas. Maka outputnya akan menghasilkan 5.

b) Kita bisa membuat beberapa object menggunakan kelas yang sama.

c) Kita juga bisa memanggil attribute kelas lain menggunakan object.

4.2 Attribute

a) Attribute sama dengan Object, Ketika kita membuat kelas kita akan mengisinya dengan objek, dan objek tersebut terdiri dari nama kelas, nama objek dan kata kunci new. Kita akan mengisi kelas dengan variabel x berupa Integer dengan hasil 5, dan membuat objek baru dengan kata kunci new diikuti nama kelas. Maka outputnya akan menghasilkan 5.

b) Atribut bisa diisi nilainya menggunakan object.

c) Jika Kita membuat beberapa objek dari satu kelas, Kita bisa mengubah nilai atribut di satu objek, tanpa memengaruhi nilai atribut di yang lain

d) Kita dapat menentukan atribut sebanyak yang Kita inginkan

4.3 Method

a) Java Method a. Ketika membuat kelas, pastinya akan membutuhkan sebuah objek, hal ini tidak berlaku pada static method dimana tidak membutuhkan sebuah objek untuk menampilkan output, beda dengan non-static method (public) yang harus membutuhkan sebuah objek, dan jika tidak ada objek, akan menampilkan pesan error ketika menggunakan public method.

b) Method with an Object, Kita membuat sebuah kelas baru bernama MyCar, dan tambahkan method fullThrottle() lalu isi ("The Car is going as fast as it can!") dan buat parameter maxSpeed yang berupa integer lalu isi ("Max speed is : " + maxSpeed). Kemudian buatlah Objek baru dengan diikuti kata kunci new dan nama kelas, lalu dibawahnya isi dengan nama objek.method fullThrottle() yang akan dipanggil, dan yang terakhir isi

dibawahnya dengan nama objek.parameter `maxSpeed()` isi dengan nilai integer (mis. 200). Maka outputnya akan seperti ini :

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

4.4 Constructor

a) Constructor, kita akan membuat Class Constructor dengan cara buatlah dahulu kelas baru dan isi dengan class atribut dengan `x` yang berupa integer, kemudian buatlah class constructor dibawahnya dengan method `public` diikuti nama kelas, lalu dibawahnya isi dengan hasil `x` tadi dengan angka 5. Setelah selesai, lanjutkan dengan membuat objek baru. Maka outputnya akan menghasilkan angka 5.

Catatan : nama konstruktor harus cocok dengan nama kelas, dan tidak boleh memiliki kata kunci `return` (seperti `void`). Konstruktor akan dapat dipanggil saat objek dibuat.

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

b) Multiple Constructor Parameter. Untuk constructor parameter kita lewati langsung ke multiple parameter, karena perbedaannya hanya pada jumlah parameter yang ada pada konstruktor itu sendiri. Disini saya memakludisi kasus data diri. Yang membuat ini menjadi multiple parameter adalah banyaknya parameter yang ada pada constructor (`long NIM,String NAMA, String ALAMAT,String JUR,int ANGKATAN`). Untuk mengisi parameter parameter tersebut kita harus mengisinya padasaan membuat object parameter tersebut.

4.5 Modifier

a) Modifier, seperti yang kita tahu bahwa, penempatan atribut final akan menghasilkan pesan error pada output jika dijalankan, karena tidak dapat mengubah nilai atribut yang sudah ditentukan sejak awal kita membuatnya.

b) Abstract class, kita akan membuat 2 class yang berbeda (`Person.java` dan `MyClass.java`).

`Person.java` ~ buatlah kelas bernama `Person.java` dan isi dengan class atribut `fname` yang berupa String "John", age yang berupa Integer yaitu 24, dan abstract class (`public abstract void study`). Kemudian kita buat subclass `Student`, extends dari `Person`. Dengan menggunakan

Public graduationYear sebagai Integer yaitu 2018, diikuti dibawahnya isi dengan public void study ("Studying all day long").

MyClass.java ~ buatlah kelas bernama MyClass.java dan buatlah object dari Student (subclass Person) dan dilanjutkan dibawahnya isi dengan ("Name : " memanggil dari fname),("Age : " memanggil dari age),("Graduation Year :" memanggil graduationYear). Lalu objek.study (untuk memanggil method abstract). Maka hasilnya akan seperti ini :

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

4.6 Encapsulation

a) Encapsulation, yang perlu kita ketahui adalah variabel name bersifat private, dan didalam praktik modul 8a, menggunakan method Public. Maka akan menghasilkan pesan error saat dijalankan. Agar bisa dijalankan, gunakan method getName() dan setName() yang akan dijelaskan pada modul 8b.

b) Method di Encapsulation, Seperti yang telah dibahas pada Modul 8a, kita harus menggunakan method getName() dan setName() agar bisa menjalankan method public dengan variabel name.

- get - mengembalikan nilai nama variabel.
- set - mengambil parameter (newName) dan menetapkan ke variabel nama

Untuk penempatannya getName() yaitu setelah nama objek. dan sebelum variabel name, dan untuk penempatan setName() yaitu dibawah getName(), setelah nama objek. didalam tanda kurung ().

4.7 Package/API

a) Package/API, ketika kita ingin mengimpor suatu class dari suatu paket java.util, kita akan menggunakan Scanner class. Untuk membuat Scanner class, buatlah sebuah objek baru dari sebuah class, dan kita isi dengan perintah yang akan muncul untuk mengisi manual pada output yang akan dijalankan nanti, coba kita isi "Enter Username".Kemudian gunakan method nextLine() untuk menampilkan apa yang telah kita ketik manual saat output dijalankan ("Username is : " memanggil userName).

b) Import a Package, kita akan mengimpor suatu package, kedalam kelas yang sudah kita buat dengan cara ketik `java.util*`; maka akan otomatis mengimpor package `java.util`.

c) User-defined Package, yang berarti membuat Paket sendiri, tidak mengimpor seperti `java.util`. Caranya, temukan terlebih dahulu folder mana yang berisi paket yang akan kita buat sendiri nanti (ekstensi `.java`). Setelah ketemu, gunakan kata kunci package kemudian nama folder kita. Setelah selesai, simpan dan compile dengan perintah `javac YourPackage.java` kemudian `javac -d . YourPackage.java`.

4.8 Inheritance

a) Java Inheritance, digunakan ketika kita ingin mewariskan sesuatu. Dalam praktik Modul 10a, saya menggunakan contoh class `Vehicle` akan diwariskan ke Class `Car`, class `Vehicle` mempunyai brand "Ford" yang berbunyi "Tuut tuut!", dan class `Car` mempunyai modelName "Mustang". Dengan urutan honk, kemudian dibawahnya berupa brand tadi dilanjutkan dengan modelName, maka akan mempunyai output seperti ini : Tuut tuut! Ford Mustang.

b) Final in Java Inheritance, seperti yang sudah kita bahas, jika kita tidak ingin memberikan / mewariskan sesuatu ke kelas lain, gunakan kata kunci final.

4.9 Polimorphism

Polymorphism, hampir sama dengan Inheritance, tetapi Polymorphism mempunyai cara yang berbeda seperti method `animalSound()` untuk diwariskan ke suatu hewan / binatang.

4.10 InnerClass

a) Java Inner Class, digunakan ketika kita ingin saling mengakses dari luar maupun dalam, dengan cara membuat objek dari kelas luar dan juga membuat objek dari kelas dalam.

b) Private Inner Class, Inner Class tidak seperti kelas yang biasanya, Inner Class tidak dapat dijalankan ketika menggunakan Private akan menjadikan output menampilkan pesan error.

c) Static Inner Class, di Inner Class dapat dibuat tanpa menggunakan objek dengan method static. Tetapi tidak dapat diakses dari kelas luar.

d) Access Outer Class from Inner Class, kita dapat mengakses kelas dalam dari kelas luar dengan cara masukkan Inner Class ke dalam Outer Class, dan juga public class untuk diisi. Berilah atribut x berupa Integer yang mempunyai nilai 10 untuk Outer Class, kemudian gunakan method return untuk dipanggil didalam public class lagi. Maka outputnya akan menghasilkan angka 10.

4.11 Data Abstraction

Data Abstraction, cara menyembunyikan detail tertentu dan hanya menampilkan informasi yang penting - penting saja. Kita juga tidak dapat membuat objek kelas animal. Data abstraction juga menggunakan method Inheritance.

4.12 Interface

a) Interfaces, Sinonim dari Abstraction, perbedaannya jika Abstraction menggunakan extends dari Inheritance, Interfaces menggunakan kata kunci implements. Dan di Interfaces juga dapat dibuat Multiple Interfaces yang akan dijelaskan pada Modul 14b.

b) Multiple Interfaces, cukup pisahkan dengan koma.

4.13 Enum

a) Enum, terdaftar secara khusus, itulah arti dari Enum. Untuk membuat enum, cukup ketik kata kunci enum dan tambahkan koma, saat memisahkan konstanta. Kita juga dapat mengakses konstanta enum dengan titik (.)

b) Enum inside a Class, enum juga dapat dimasukkan kedalam kelas, dengan cara buatlah kelas baru dan tambahkan kata kunci enum, dan jangan lupa memisahkan konstanta dengan koma.

c) Enum in a Switch Statement, caranya sama seperti Modul 15b, cukup tambahkan kata kunci enum dan pisahkan konstanta dengan koma.

4.14 User Input

a) Java User Input, kita dipertemukan lagi dengan java.util.Scanner, java.util.Scanner kembali digunakan pada java user input, buatlah objek dan gunakan method dari Scanner class, contohnya nextLine() digunakan untuk membaca tipe data String. Dan saat dijalankan, kita akan diberi perintah untuk memasukkan input yang sesuai dengan perintah yang tertulis.

b) Different Method Java User Input, sama dengan modul 16a, hanya saja menambahkan berbagai varian method seperti `nextInt()`, `nextDouble()`, dll.

4.15 Date

a) `Date`, digunakan untuk menampilkan Tanggal-Bulan-Tahun saat ini sekarang juga.

b) `Display Current Time`, digunakan untuk menampilkan Waktu (Jam, Menit, Detik, Mili-detik) saat ini sekarang juga.

c) `Display Current Date and Time`, digunakan untuk menampilkan Tanggal dan Waktu secara bersamaan saat ini sekarang juga.

d) `Formatting Date and Time`, digunakan untuk memformat tata letak Tanggal dan Waktu.

4.16 ArrayList

`ArrayList`, berbeda dengan array biasa, `ArrayList` memiliki sejumlah operasi yang lebih lengkap dan mudah digunakan dibandingkan dengan array biasa. `ArrayList` merupakan collection yang menjadi bagian dari `java.util`. Seperti biasa, `ArrayList` dapat menambah data baru secara dinamis tanpa harus menentukan ukurannya di awal. Berbagai operasi dapat Anda lakukan terhadap `ArrayList` seperti berikut:

- `size()`, untuk mencari panjang *ArrayList*
- `add()`, untuk menambah elemen baru
- `get()`, untuk mengambil elemen pada indeks tertentu
- `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 menimpa nilai pada indeks tertentu
- `remove()`, untuk menghapus nilai pada indeks tertentu

4.17 HashMap

`HashMap`, sebuah collection yang berbasis hash di Java. `HashMap` mengimplimentasikan interface `java.util.Map`. Namun, `HashMap`, menyimpan item dalam pasangan "key / value", dan Anda dapat mengaksesnya dengan indeks jenis lain (mis. `String`).

HashMap juga memiliki performa yang constant time. Dan yang terpenting HashMap unggul dalam performanya, karena HashMap tidak tersinkronisasi. Satu objek digunakan sebagai key (indeks) ke objek lain (value). Ini dapat menyimpan berbagai jenis, Key string dan value Integer, atau tipe yang sama, seperti Key string dan value String.

4.18 Wrapper Class

Wrapper Classes, Wrapper Classes memberitahu kita bagaimana caranya menggunakan tipe data primitif (int, boolean, dll.) menjadi objek. Ada kalanya dimana kita harus menggunakan wrapper classes, mis. saat digunakan di objek collection, tipe data primitif tidak dapat digunakan di ArrayList.

4.19 Exception

Exceptions. Seperti artinya, ketika kesalahan terjadi, Java biasanya akan berhenti dan menghasilkan pesan error. Java akan menunjukkan Exception (bukan Error). Ada 2 macam Exception, Try and Catch.

- Try ~ Statements memungkinkan kita untuk menentukan blok kode yang akan diuji untuk error saat sedang dieksekusi.
- Catch ~ Statements memungkinkan kita untuk menentukan blok kode yang akan dieksekusi, jika error terjadi di blok try.
- Finally ~ Statements memungkinkan kita menjalankan kode, setelah try ... catch, tidak memunculkan hasilnya.
- Throw ~ Statements memungkinkan kita untuk membuat custom error.

BAB V PENUTUP

KESIMPULAN

Java adalah bahasa pemrograman tingkat tinggi yang sangat dinamis yang memungkinkan kita mengembangkan aplikasi web, Android, dll. Java telah menjadi bahasa yang sangat berlaku untuk Internet dan pemrograman sistem terdistribusi secara umum. Tetapi bidang aplikasinya bukan semata-mata Internet, salah satu kelebihan Java adalah dimaksudkan untuk sepenuhnya bebas dari perangkat keras ada mesin virtual Java untuk berbagai jenis komputer.

SARAN

Dari Laporan yang saya buat, diharapkan Laporan ini dapat bermanfaat bagi pembaca dalam memahami tentang Bahasa Pemrograman Java. Selain itu saya juga menyarankan untuk menerapkan apa yang baik dari Laporan ini dan juga mengingatkan saya apa yang dianggap pembaca kurang baik dari Laporan ini. Sebagai penyusun, saya akui tidak terlepas dari kesalahan dan keterbatasan. Karena itu penyusun mengharapkan kritik dan saran yang membangun untuk perbaikan penulisan Laporan selanjutnya. Semoga Laporan ini dapat bermanfaat bagi para pembaca.

Pasuruan, 5 Januari 2020

Achmad Syahrialdi Noor Akmal

DAFTAR PUSTAKA

https://www.w3schools.com/java/java_methods.asp
https://www.w3schools.com/java/java_methods_param.asp
https://www.w3schools.com/java/java_methods_overloading.asp
https://www.w3schools.com/java/java_oop.asp
https://www.w3schools.com/java/java_classes.asp
https://www.w3schools.com/java/java_class_attributes.asp
https://www.w3schools.com/java/java_class_methods.asp
https://www.w3schools.com/java/java_constructors.asp
https://www.w3schools.com/java/java_modifiers.asp
https://www.w3schools.com/java/java_encapsulation.asp
https://www.w3schools.com/java/java_packages.asp
https://www.w3schools.com/java/java_inheritance.asp
https://www.w3schools.com/java/java_polymorphism.asp
https://www.w3schools.com/java/java_inner_classes.asp
https://www.w3schools.com/java/java_abstract.asp
https://www.w3schools.com/java/java_interface.asp
https://www.w3schools.com/java/java_enums.asp
https://www.w3schools.com/java/java_user_input.asp
https://www.w3schools.com/java/java_date.asp
https://www.w3schools.com/java/java_arraylist.asp
https://www.w3schools.com/java/java_hashmap.asp
https://www.w3schools.com/java/java_wrapper_classes.asp
https://www.w3schools.com/java/java_try_catch.asp
[https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
[https://en.wikipedia.org/wiki/Java_\(software_platform\)](https://en.wikipedia.org/wiki/Java_(software_platform))
<https://en.wikipedia.org/wiki/Java>